PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE ABEJAS PARA LA PLANIFICACIÓN DE LA DISTRIBUCIÓN DE AYUDA EN CASO DE FENÓMENOS NATURALES EN EL PERÚ

Tesis para obtener el título profesional de Ingeniero Informático

AUTOR:

Jeison Tonny Romero Salinas

ASESOR:

Rony Cueva Moscoso

Lima, Febrero, 2025

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 IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE ABEJAS PARA LA PLANIFICACIÓN DE LA DISTRIBUCIÓN DE AYUDA EN CASO DE FENÓMENOS NATURALES EN EL PERÚ ……..,

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

Jeison Tonny Romero Salinas ……,

…….……,

…….……,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 20%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 30/01/2025.
- 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, 10 de febrero del 2025……

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

Resumen

La investigación aborda la implementación de un algoritmo de colonia de abejas para mejorar la planificación de distribución de ayuda humanitaria en situaciones de desastres naturales en Perú. Este estudio se fundamenta en la necesidad de mejorar la gestión y distribución de recursos en respuesta a eventos como terremotos e inundaciones, que suelen afectar gravemente a las poblaciones y donde la rapidez y eficiencia son cruciales. Por lo tanto, en situaciones de desastre, la escala y la velocidad de la respuesta humanitaria son cruciales para salvar vidas y aliviar el sufrimiento. El objetivo principal es implementar dicho algoritmo para lograr una distribución más eficiente en términos de cobertura y equidad, comparando su desempeño con el de un algoritmo genético previamente utilizado.

El estudio se basa en los principios de los algoritmos bioinspirados, que imitan el comportamiento colectivo de organismos para resolver problemas complejos. Se adopta un enfoque metodológico basado en el desarrollo y prueba del algoritmo propuesto, utilizando herramientas como C++, Python y técnicas de programación extrema. El procedimiento incluye la definición de variables clave, la construcción del pseudocódigo, y la experimentación numérica para comparar ambos algoritmos.

Los resultados esperados incluyen una distribución más eficiente y justa de la ayuda, con menor costo operativo. Finalmente, una vez culminado el trabajo de investigación se concluye que el algoritmo de colonia de abejas ofrece una solución prometedora para la optimización de la distribución en situaciones de emergencia, proporcionando mejoras en eficiencia y tiempo de respuesta en comparación con el algoritmo genético.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades

1.1 Problemática

La humanidad ha experimentado continuamente los efectos devastadores de desastres naturales como terremotos, tsunamis y tornados, cuyo impacto varía según la ubicación geográfica. Según la OMM (Organización Meteorológica Mundial), entre 1970 y 2019, estos eventos causaron más de 2 millones de vidas perdidas y un impacto económico valorado en 3,64 billones de dólares (OMM, 2021). Ante estos efectos, surge la necesidad de mitigar y prepararse para actuar antes, durante y después de estos desastres (Espinosa Bordón, 2008). Sin embargo, dado que los desastres son inevitables y difíciles de predecir (Kumar & Havey, 2013), las organizaciones enfrentan desafíos al coordinar esfuerzos logísticos, complicando la entrega puntual de ayuda en las zonas impactadas por obstrucciones en las vías y la escasez de recursos.

La logística humanitaria involucra múltiples sectores, cada uno con sus propias capacidades y limitaciones (Balcik et al., 2010; Kumar & Havey, 2013; Yadav & Barve, 2015; Regis-Hernández et al., 2017). Las dificultades de coordinación entre estos actores pueden demorar la entrega de ayuda, lo cual se agrava cuando las infraestructuras quedan destruidas y se requiere una gestión más eficiente de los recursos disponibles (CRED, 2020). Esto subraya la importancia de identificar necesidades prioritarias y optimizar la distribución, especialmente en los primeros días tras el desastre, cuando la supervivencia depende de los recursos esenciales (Aduviri Choque & Robert Alonso, 2019).

Además, los problemas logísticos en la entrega de alimentos, medicamentos y otros suministros tienen un impacto considerable en la salud pública, tanto en el corto como en el largo plazo, debido a la carencia de servicios básicos y la exposición a enfermedades (Arcos González et al., 2002). En situaciones de emergencia, muchas personas se ven obligadas a desplazarse buscando ayuda, lo cual también contribuye a la crisis (IDMC, 2020).

La planificación logística es esencial para la distribución de ayuda en grandes desastres, e implica resolver problemas de optimización, como ubicación de almacenes, distribución, y planeamiento de

inventarios. Esto resulta complejo debido a las limitaciones de capacidad y los costos (Toth & Vigo, 2002).

Por ejemplo, una de las soluciones para solucionar este problema de distribución fue propuesta por el ingeniero Robert Aduviri Choque en el año 2018, en el que implementó un algoritmo genético, considerado un método inspirado en la biología, que brindaba soluciones óptimas. Por tal motivo, mi tesis es solucionar este problema, pero mejorando la solución por medio de otro algoritmo bioinspirado para obtener mejores resultados y con un menor tiempo de respuesta.

1.2 Objetivos

1.2.1 Objetivo general

Implementar un algoritmo de colonia de abejas con el objetivo de optimizar la planificación de la distribución de ayuda humanitaria en situaciones de desastres naturales en Perú.

1.2.2 Objetivos específicos

O1. Identificar y seleccionar las variables y restricciones más relevantes en la distribución de ayuda humanitaria en caso de fenómenos naturales.

O2. Implementar el algoritmo de colonia de abeja spara abordar el desafío de la distribución de ayuda humanitaria.

O3. Comparar los resultados obtenidos mediante la experimentación numérica del algoritmo presentado en este trabajo contra la solución que emplea un algoritmo genético para la optimización en la distribución de ayuda humanitaria en caso de fenómenos naturales.

1.2.3 Resultados esperados

Resultados esperados del objetivo 1 (O1):

R 1. Identificación de las variables clave y las restricciones esenciales para la distribución de ayuda.

R 2. Diseño de la función objetivo destinada a optimizar la distribución de ayuda..

Resultados esperados del objetivo 2 (O2):

R 3. Especificación de las estructuras de datos que serán empleadas por el algoritmo de colonia de abejas propuesto.

R 4. Adaptación de un algoritmo inspirado en la colonia de abejas para resolver el problema de distribución de ayuda.

R 5. Codificación del algoritmo de colonia de abejas ajustado para abordar el problema de distribución de ayuda.

R 6. Codificación del algoritmo genético ajustado para resolver el problema de distribución de ayuda.

R 7. Interfaz gráfica diseñada para ejecutar el algoritmo basado en la colonia de abeja aplicado al problema de distribución de ayuda.

R 8. Ajustar las variables necesarias para la implementación de los algoritmos de colonia de abejas y genético.

Resultados esperados del objetivo 3 (O3):

R 9. Informe de análisis numérico destinado a evaluar la eficiencia de los algoritmos de colonia de abejas y genético.

1.2.4 Mapeo de objetivos, resultados y verificación

La Tabla 1 resume el objetivo específico, los resultados esperados, los medios de verificación y los indicadores verificables.

Tabla 1. Mapeo de objetivos, resultados y verificación

Objetivo (O1): Identificar y seleccionar las variables y restricciones más relevantes en la distribución de ayuda humanitaria en caso de fenómenos naturales		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**

(R1) Identificación de las variables clave y las restricciones esenciales para la distribución de ayuda	- Documento que incluye la descripción de las variables y las restricciones.	- Revisión y validación al 100% con respecto a la tesis a mejorar.
(R2) Diseño de la función objetivo destinada a optimizar la distribución de ayuda.	- Documento que incluye la definición de la función objetivo.	- Verificación y validación completa con respecto a la tesis a mejorar.
Objetivo (O2): Implementar el algoritmo de colonia de abeja para abordar el desafío de la distribución de ayuda humanitaria.		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
(R3) Especificación de las estructuras de datos que serán empleadas por el algoritmo de colonia de abejas propuesto.	- Documento que describe las estructuras de datos necesarias para la implementación del algoritmo de colonia de abejas.	- Verificación y aprobación completa por un especialista en diseño de algoritmos.
(R4) Adaptación de un algoritmo inspirado en la colonia de abejas para	- Documento que contiene el pseudocódigo del	- Verificación y aprobación completa de un especialista en diseño de algoritmos.

resolver el problema de distribución de ayuda.	algoritmo de colonia de abejas. - Documento que detalla la realización de pruebas relacionadas con el flujo de datos para el diseño del algoritmo.	
(R5) Codificación del algoritmo de colonia de abejas ajustado para abordar el problema de distribución de ayuda.	- Programa fuente del algoritmo de colonia de abejas y sus funciones auxiliares. - Documento que contiene el conjunto de pruebas unitarias realizadas a la codificación del algoritmo de colonia de abejas.	- 100% de las pruebas unitarias del algoritmo de colonia de abejas y sus funciones auxiliares ejecutadas con éxito. - Verificación y aprobación completa de un especialista en diseño de algoritmos.
(R6) Codificación del algoritmo genético ajustado para resolver el problema de distribución de ayuda.	- Programa fuente del algoritmo genético y sus funciones auxiliares. - Documento que recopila las pruebas	- 100% de las pruebas unitarias del algoritmo genético y sus funciones auxiliares ejecutadas con éxito.

	unitarias realizadas sobre la programación del algoritmo genético.	- Verificación y aprobación completa por un especialista en el diseño de algoritmos.
(R7) Interfaz gráfica diseñada para ejecutar el algoritmo basado en la colonia de abeja aplicado al problema de distribución de ayuda.	- Programa fuente de la interfaz para ejecutar el algoritmo. - Documento que recopila las pruebas unitarias realizadas a la interfaz gráfica	- Ejecución exitosa del 100% de las pruebas unitarias de la interfaz del algoritmo. - Verificación y aprobación completa por un especialista en el diseño de algoritmos
(R8) Ajustar las variables necesarias para la implementación de los algoritmos de colonia de abejas y genético.	- Documento que contiene las pruebas que se hicieron para calibrar las variables con los valores más óptimos.	- Verificación y aprobación completa de un especialista en el diseño de algoritmos.
Objetivo (O3): Comparar los resultados obtenidos mediante la experimentación numérica del algoritmo presentado en este trabajo contra la solución que emplea un algoritmo genético para la optimización en la distribución de ayuda humanitaria en caso de fenómenos naturales.		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**

(R9) Informe de análisis numérico destinado a evaluar la eficiencia de los algoritmos de colonia de abejas y genético.	- Documento que presenta un informe detallado de la experimentación numérica realizada sobre los algoritmos, incluyendo las conclusiones obtenidas.	- Verificación y aprobación completa por un especialista en el diseño de algoritmos, quien valida que la experimentación fue adecuada y los resultados correctos.

1.3 Métodos y Procedimientos

Esta sección detalla las herramientas, métodos y procedimientos que se emplearán en el presente proyecto de tesis, los cuales se encuentran resumidos en la Tabla 2 .

Tabla 2 Herramientas, métodos y procedimientos

Resultados esperados	**Herramientas, metodologías, métodos y procedimientos**
Identificación de las variables clave y las restricciones principales para la distribución de ayuda	- No aplica
Diseño de la función objetivo destinada a optimizar la distribución de ayuda.	- No aplica
Especificación de las estructuras de datos que serán empleadas por el algoritmo de colonia de abejas propuesto.	- No aplica

Adaptación de un algoritmo de colonia de abejas para el problema de distribución de ayuda.	- No aplica
Codificación del algoritmo de colonia de abejas ajustado para abordar el problema de distribución de ayuda.	- C++ - Visual Code - Python - Jupyter Lab - Extreme Programming - Pruebas unitarias
Codificación del algoritmo genético ajustado para resolver el problema de distribución de ayuda.	- C++ - Visual Code - Python - Jupyter Lab - Extreme Programming - Pruebas unitarias
Interfaz gráfica diseñada para ejecutar el algoritmo basado en la colonia de abeja aplicado al problema de distribución de ayuda.	- C++ - Visual Code - Python - Jupyter Lab - Extreme Programming - Pruebas unitarias
Ajustar las variables necesarias para la implementación de los algoritmos de colonia de abejas y genético.	- C++ - Visual Code
Informe de análisis numérico destinado a evaluar la eficiencia de los algoritmos de colonia de abejas y	- Prueba de Shapiro-Wilk

genético.	- Prueba F - Prueba Z - R Studio

1.3.1 Herramientas

1.3.1.1 C++

C++ es un poderoso lenguaje de programación avanzado integra funcionalidades de la programación orientada a objetos con las capacidades de bajo nivel del lenguaje C (Bjarne Stroustrup, 2013). C++ se destaca por su eficiencia, rendimiento y flexibilidad. Permite a los programadores controlar directamente los recursos del sistema y proporciona características como la sobrecarga de operadores, la herencia múltiple y los punteros, lo que lo convierte en un lenguaje muy versátil.

1.3.1.2 Visual Code

Visual Studio Code (Microsoft, 2018) es un entorno de desarrollo integrado (IDE) creado por Microsoft, que ofrece herramientas como análisis de código, detección de errores en tiempo real, sugerencias de corrección y avanzadas opciones de navegación en una plataforma liviana. Por ello, se seleccionó para este proyecto, tanto para la implementación del algoritmo final como para el desarrollo de la interfaz de usuario.

1.3.1.3 Python

Python es un lenguaje de programación interactivo, interpretado y orientado a objetos, que ofrece módulos, gestión de excepciones, tipos de datos avanzados y funcionalidad para trabajar con clases. (Python Software Foundation, 2018). Python es un lenguaje flexible y multifacético, ideal para crear desde scripts básicos hasta aplicaciones web avanzadas y realizar análisis de datos complejos. Cuenta con una gran cantidad de bibliotecas y frameworks que brindan funcionalidades adicionales y simplifican el desarrollo de proyectos.

1.3.1.4 Jupyter Lab

Jupyter Notebook es una plataforma web diseñada para crear y compartir documentos que combinan código, ecuaciones, gráficos y texto descriptivo (Jupyter, 2018). Jupyter Lab es una versión avanzada del entorno Jupyter Notebook, creada para simplificar la exploración, el análisis y la visualización de datos, además de apoyar el desarrollo de proyectos en ciencia de datos y aprendizaje automático. Una de las características destacadas de JupyterLab es su diseño modular y su capacidad de personalización. Permite a los usuarios organizar su espacio de trabajo mediante la combinación de diferentes componentes en pestañas y paneles.

1.3.1.5 R Studio

R es un lenguaje de programación y una plataforma enfocada en el análisis estadístico y la gestión de datos. Por su parte, R Studio es un IDE para R que proporciona una consola, un editor con resaltado de sintaxis y diversas herramientas para gestionar el espacio de trabajo (RStudio, 2020). Ofrece un entorno estructurado con una consola interactiva, un editor de código, un gestor de archivos y secciones dedicadas a la visualización de gráficos y resultados.

1.3.2 Metodologías, métodos y procedimientos

1.3.2.1 Extreme Programming

Extreme Programming (XP) es una metodología ágil centrada en el desarrollo de software, con énfasis en la entrega constante de soluciones de alta calidad. Su premisa central es enfrentar los retos del desarrollo de software a través de la implementación de valores y prácticas fundamentales.. Estos incluyen la retroalimentación continua, la simplicidad, la comunicación efectiva, la aceptación del cambio y el enfoque en la calidad del software. Una ventaja adicional de este marco es su conjunto de reglas sencillas (Extreme Programming, 2013), que resultan fáciles de aplicar y especialmente adecuadas para proyectos de corta duración.

1.3.2.2 Prueba de Shapiro-Wilk

El test de Shapiro-Wilk es una técnica estadística empleada para evaluar si una muestra de datos se ajusta a una distribución normal, comparando los valores observados con los esperados bajo normalidad

(Razali & Wah, 2011). Esta prueba es ampliamente utilizada en estadística y otras disciplinas para asegurar la validez de análisis que asumen normalidad, ayudando a determinar las pruebas estadísticas más adecuadas en un contexto específico.

1.3.2.3 Prueba F

El test F es un análisis estadístico que permite comprobar si las varianzas de dos muestras independientes son iguales, mediante la comparación entre la varianza entre los grupos con la varianza dentro de ellos (Fallas, 2012). Este método se utiliza frecuentemente en análisis de varianza (ANOVA), diseños experimentales y modelos de regresión, y facilita la identificación de diferencias significativas entre las medias de los grupos o los efectos de las variables independientes sobre la dependiente.

1.3.2.4 Prueba Z

El test Z es una herramienta estadística utilizada para determinar si la media de una muestra presenta una diferencia significativa con respecto a una hipótesis previamente planteada, calculando un estadístico Z que mide esta diferencia en términos de desviaciones estándar (Massey & Miller, 2016). Es ampliamente empleada en investigación científica, economía y psicología, permitiendo decisiones basadas en evidencia a partir de datos muestrales.

1.3.2.5 Pruebas unitarias

Las pruebas unitarias son fundamentales en el desarrollo de software, asegurando de forma independiente que cada componente del código opere correctamente antes de ser integrado al sistema completo. Estas pruebas cubren diferentes escenarios y casos límite, comparando los resultados esperados con los obtenidos. Entre sus beneficios, destacan la detección temprana de errores, lo que reduce costos, y la confianza que brindan al desarrollador en la estabilidad del código (Lindberg & Strandberg, 2006).

Capítulo 2. Marco Legal/Regulatorio/Conceptual/otros

2.1 Introducción

El propósito de esta sección es explicar ciertos conceptos clave para mejorar la comprensión de la problemática relacionada con la distribución de ayuda humanitaria en situaciones de desastres naturales. Con dichos conceptos se podrá entender mejor el contexto y las consideraciones que se tienen al abordar este problema.En consecuencia, se comenzará abordando conceptos vinculados a la logística humanitaria utilizada en este tipo de situaciones, luego conceptos acerca de los problemas o factores que se tienen en cuenta al momento de plantear la distribución de ayuda en este tipo de casos y finalmente, se desarrollarán conceptos acerca de los métodos o algoritmos que se emplean para resolver este tipo de casos.

2.2 Desarrollo del marco

2.2.1 Fenómeno natural

Un fenómeno natural se refiere a un suceso que tiene lugar en la naturaleza de manera independiente a la acción humana. Los fenómenos naturales tienen la capacidad de incluir desde eventos meteorológicos como huracanes, tornados, terremotos, inundaciones, erupciones volcánicas, hasta eventos astronómicos como eclipses, cometas, lluvias de meteoros, entre otros.Durante mucho tiempo, los desastres se han considerado como equivalentes a eventos naturales o actos divinos imposibles de controlar. Sin embargo, para que ocurra un desastre no es suficiente la presencia de un fenómeno natural; es necesario que existan condiciones de vulnerabilidad en las personas y en los asentamientos humanos (Cepal, 2014). La situación se complica debido a la falta de una planificación urbana en el desarrollo de muchas ciudades, además de actividades ilegales como el tráfico de terrenos y la habilitación de zonas urbanas en áreas inseguras (PuntoEdu, 2023).

2.2.2 Ayuda humanitaria

La ayuda humanitaria consiste en proporcionar apoyo a las personas afectadas por un desplazamiento, asegurando su acceso a servicios esenciales como alimentación, atención médica, agua potable o alojamiento. Estas situaciones de desplazamiento suelen originarse por desastres naturales, conflictos

bélicos o guerras, siendo conocidas como emergencias humanitarias (ACNUR, 2019). Las medidas y acciones pueden incluir la distribución de suministros de emergencia, la asistencia médica y psicológica, la protección de los derechos humanos, la atención a niños y mujeres en situación de vulnerabilidad y la creación de condiciones para la recuperación y el desarrollo sostenible.

2.2.3 Distribución de ayuda humanitaria

La distribución de ayuda humanitaria es el procedimiento mediante el cual se entregan suministros y servicios de emergencia a las personas afectadas en contextos de crisis humanitaria. Este proceso puede implicar la participación de diversas entidades, como organizaciones no gubernamentales, agencias de la ONU, gobiernos y otros actores involucrados en la ayuda humanitaria. La distribución de ayuda humanitaria comienza con una evaluación detallada de las necesidades en el área afectada. A partir de allí, se elabora un plan de respuesta que define qué tipo de ayuda se necesita y en qué cantidad. Los suministros pueden incluir alimentos, agua, medicamentos, kits de higiene personal, refugio y otros elementos esenciales (Nawazish et al., 2022). La distribución de ayuda humanitaria puede ser un proceso complejo y desafiante, especialmente en áreas afectadas por conflictos armados o desastres naturales.Asimismo, es fundamental asegurar que la distribución de los suministros se realice de forma justa e imparcial, sin ningún tipo de discriminación.

2.2.4 Tiempo de distribución de recursos (tiempo de viaje)

La rapidez en la distribución de recursos durante fenómenos naturales es un aspecto clave que puede determinar la supervivencia de las personas afectadas. En situaciones de emergencia, es esencial que los recursos lleguen a las personas que los necesitan lo más rápido posible. La distribución de recursos puede incluir desde alimentos y agua hasta suministros médicos, tiendas de campaña, mantas y otros suministros esenciales. En muchos casos, estos recursos son transportados por avión, barco o camión a través de terrenos difíciles o áreas afectadas por el desastre (Chen et al., 2020). La velocidad con la que los recursos llegan a las personas afectadas puede variar según diversos factores, como la magnitud y alcance del fenómeno, la disponibilidad de insumos y el nivel de coordinación entre los grupos encargados de su distribución.

2.2.5 Costos operativos de distribución de ayuda

Los costos operativos en una empresa son los gastos necesarios para mantener la actividad principal de producción y continuidad de la operación. Estos se clasifican en costos fijos, que no varían independientemente del nivel de producción, como el pago del alquiler, y costos variables, que fluctúan en función del volumen de producción, como los salarios durante periodos de alta demanda (Rabiei & Arias-Aranda, 2021). En el contexto de ayuda humanitaria, los costos operativos se enfocan en los gastos para llevar suministros a las zonas afectadas, incluyendo transporte, almacenamiento y personal. Además, estos costos pueden abarcar gastos de compra de suministros, apertura de plantas de fabricación y centros de distribución, necesarios para una respuesta efectiva en emergencias (Zavvar Sabegh et al., 2017).

2.2.6 Nivel de demanda

En contextos de desastres naturales, la demanda se refiere a la cantidad de recursos necesarios para atender la emergencia, incluyendo alimentos, agua, suministros médicos y equipos de rescate. Este nivel de necesidad puede variar considerablemente según la gravedad del desastre y el número de personas impactadas. Asimismo, la provisión de atención médica constituye un elemento esencial, con demandas específicas en áreas como saneamiento ambiental, control epidemiológico, nutrición y servicios asistenciales (Daniel Bitrán Bitrán, 1995). Para atender estas necesidades de forma efectiva y oportuna, es fundamental una planificación adecuada y una coordinación eficiente de los recursos disponibles. Cada punto de ayuda se evalúa y prioriza según su nivel de necesidad para asegurar que los recursos se distribuyan de manera adecuada y donde más se necesitan (Bozorgi-Amiri et al., 2012).

2.2.7 Problema de enrutamiento

El enrutamiento de vehículos con restricciones de capacidad y flota homogénea es un problema de optimización combinatoria clasificado dentro de los problemas NP-completos, caracterizados por la ausencia de una solución óptima en tiempo polinomial (Julio Mario Daza, Jairo R. Montoya, Francesco Narducci, 2009). Bajo el contexto de desastres naturales, este problema se refiere a la dificultad de encontrar rutas eficientes para el transporte de ayuda en áreas afectadas. Las carreteras pueden estar bloqueadas o dañadas, dificultando el acceso a los afectados. La solución requiere una planificación

detallada y una coordinación eficaz entre equipos de respuesta, ONG y autoridades locales, incluyendo la identificación de rutas alternativas y la evaluación de la seguridad y accesibilidad de las vías.

2.2.8 Múltiple demanda

Durante los desastres naturales, la necesidad de recursos materiales aumenta para satisfacer los requerimientos básicos de las personas impactadas. Entre los recursos más solicitados están los suministros médicos, alimentos, agua potable y refugios, que son esenciales para el tratamiento, la nutrición y la hidratación de los damnificados, especialmente en los primeros días (Chen et al., 2020). Además, estos eventos también crean una necesidad de recursos adicionales, como equipos de construcción para reconstruir infraestructuras, generadores de energía para los hospitales y refugios, y equipos de comunicación para coordinar eficazmente los esfuerzos de rescate. Sin embargo, existen varios desafíos, como la red vial dañada, la alta demanda de recursos, la escasez de suministros, sitios de múltiples demandas y una capacidad de transporte limitada, lo que complica la distribución (Chen et al., 2020).

2.2.9 Objetivos múltiples

Después de un fenómeno natural, las necesidades de los afectados requieren estrategias que aborden múltiples objetivos en la distribución de ayuda humanitaria. La metodología multiobjetivo facilita la optimización en la toma de decisiones en contextos complejos, abarcando elementos como la preservación de vidas humanas, la protección de infraestructuras clave, la reducción del impacto del desastre y la restauración de la normalidad en la comunidad (Mohammad Mohammadi & Naderi, 2017). La modelización y simulación computacional son herramientas clave en esta técnica, ya que permiten evaluar diversos escenarios en tiempo real. En conclusión, la técnica multiobjetivo es esencial para una toma de decisiones integral en emergencias, contemplando todos los factores relevantes para una respuesta efectiva.

2.2.10 Algoritmo bioinspirado

Los algoritmos inspirados en la biología representan una categoría de métodos que emulan procesos naturales y comportamientos de organismos vivos para abordar problemas complejos en distintas áreas (Instituto de Ingeniería del Conocimiento, 2021). Entre los más destacados se encuentran el algoritmo genético, que se fundamenta en los principios de la selección natural, y el algoritmo de colonia de

hormigas, diseñado para replicar el comportamiento de las hormigas en la búsqueda de rutas óptimas. También destacan otros como el enjambre de partículas, la optimización de colonias de abejas y la inteligencia de enjambre. Estos algoritmos son especialmente útiles en áreas como la ingeniería, la robótica y la computación, ya que ofrecen soluciones eficientes a problemas de optimización incluso cuando los datos son incompletos o inciertos. Además, su capacidad para operar en sistemas distribuidos los hace ideales para aplicaciones en línea.

2.2.11 Optimización estocástica

La optimización estocástica es una técnica matemática diseñada para encontrar soluciones óptimas en situaciones donde la incertidumbre es un factor clave. A diferencia de la optimización determinista, esta técnica integra la aleatoriedad y permite tomar decisiones más robustas y flexibles, ajustando rápidamente las soluciones según las condiciones cambiantes. En el ámbito de los fenómenos naturales, resulta valioso para la planificación de recursos y la toma de decisiones, particularmente en escenarios donde la información es limitada o poco precisa (Mamashli et al., 2021).

Por ejemplo, en casos de huracanes, la optimización estocástica ayuda a planificar evacuaciones al considerar factores inciertos, como la intensidad del huracán y la topografía del área, creando planes más efectivos y con menor impacto. También se aplica en la planificación de recursos para emergencias, optimizando la ubicación de equipos de rescate y suministros médicos, lo que permite una respuesta más rápida y eficiente al incorporar la incertidumbre en el modelo.

2.2.12 Algoritmo genético

Los algoritmos genéticos son métodos de optimización inspirados en los principios de la evolución biológica y la selección natural, utilizadas para obtener soluciones aproximadas en problemas complejos donde no es factible realizar una búsqueda exhaustiva. Estos algoritmos operan sobre un conjunto de soluciones potenciales, llamadas "individuos", y aplican operadores de selección, cruce y mutación para simular los procesos evolutivos. En el ámbito de los fenómenos naturales, los algoritmos genéticos destacan por su utilidad en la planificación de recursos y la toma de decisiones, especialmente en escenarios caracterizados por una elevada incertidumbre (Aduviri Choque, Robert Alonso, 2019). or ejemplo, durante un terremoto, estos algoritmos pueden mejorar la asignación de recursos de rescate teniendo en cuenta aspectos como la densidad poblacional y las capacidades de los equipos de

emergencia. También pueden utilizarse en la planificación de rutas de evacuación en casos de huracanes, tomando en cuenta aspectos como la topografía y la ubicación de refugios para crear planes robustos que minimicen el impacto del fenómeno y protejan vidas.

2.2.13 Algoritmo de colonia de abejas

El algoritmo Artificial Bee Colony (ABC), desarrollado por Dervis Karaboga en 2005, es un método de optimización basado en la conducta cooperativa de las abejas. En este algoritmo, cada solución es representada como una fuente de alimento que las abejas artificiales exploran en un espacio de búsqueda multidimensional. El algoritmo busca un balance entre exploración y explotación para identificar soluciones óptimas, tanto locales como globales. Comienza con soluciones generadas al azar, donde las abejas empleadas optimizan las opciones y las observadoras seleccionan y mejoran las mejores fuentes basándose en la información compartida. Las abejas exploradoras, por su parte, introducen nuevas soluciones reemplazando aquellas que no han mejorado en un tiempo determinado (Silvana Yanet García, 2018). La efectividad del ABC está determinada por parámetros como la cantidad de fuentes de alimento, el límite de intentos y el número de ciclos, los cuales afectan su eficiencia y capacidad para hallar soluciones óptimas, aunque ajustarlos correctamente puede resultar desafiante.

El pseudocódigo del algoritmo ABC clásico se detalla a continuación:

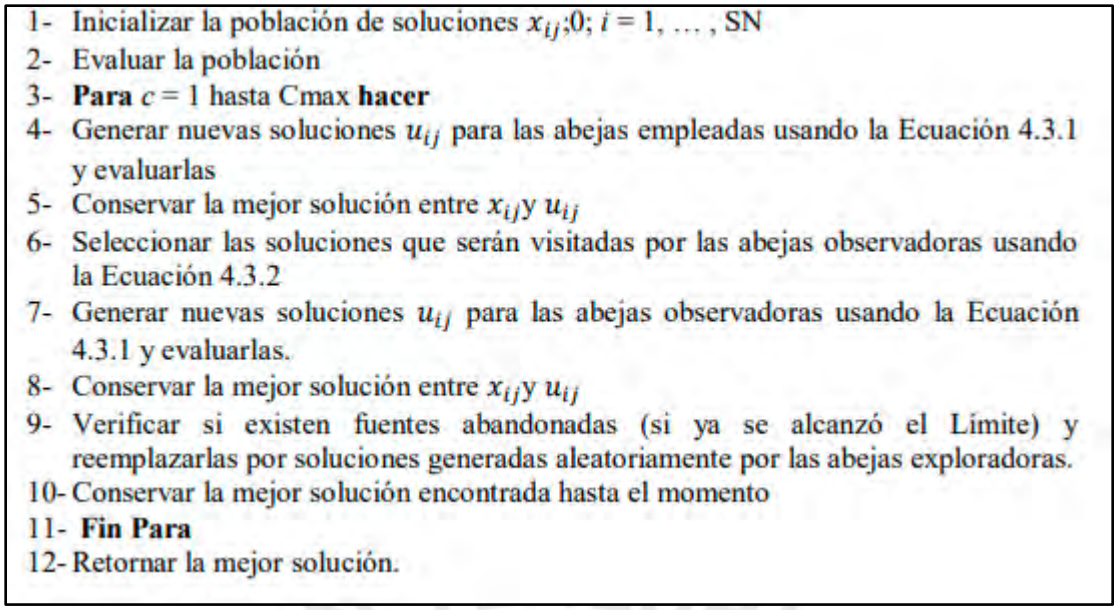


Figura 1. Pseudocódigo del algoritmo ABC. Recuperado de https://repo.unlpam.edu.ar/bitstream/handle/unlpam/7706/i_garopt963.pdf?sequence=1&isAllowed=y

En el algoritmo de colonia de abejas, SN indica las fuentes de alimento, Cmax es el total de ciclos del algoritmo, y el Límite define los ciclos permitidos sin mejoras antes de reemplazar la solución con una nueva de una abeja exploradora. Estos algoritmos aprovechan la inteligencia colectiva de las abejas artificiales para encontrar soluciones óptimas o cercanas a la óptima, y son eficientes en términos computacionales, ya que requieren menos evaluaciones de la función objetivo en comparación con otros métodos de optimización. Son especialmente útiles en problemas de múltiples variables que requieren muchos cálculos, como la optimización de nodos (Diego Isla López, 2018). No obstante, una desventaja es que el algoritmo puede quedar atrapado en mínimos locales, limitando su capacidad para explorar otras soluciones potencialmente mejores. La representación de soluciones y la elección de la función objetivo impactan el rendimiento del algoritmo, y una configuración inadecuada puede disminuir su eficacia. Para evitar mínimos locales, se emplean estrategias como reinicios aleatorios (Silvana Yanet García, 2018).

Capítulo 3. Estado del Arte

3.1 Introducción

La revisión de literatura es un enfoque sistemático que recopila y sintetiza estudios previos, formando un fundamento para el avance del conocimiento y la creación de teorías (Baumeister y Leary, 1997; Tranfield, Denyer y Smart, 2003; Webster & Watson, 2002). Este trabajo utilizará una revisión sistemática y comparativa de casos de estudio, la cual permite integrar hallazgos empíricos de múltiples estudios, ofreciendo una perspectiva amplia que ningún estudio individual puede proporcionar (Hannah Snyder, 2019). Una revisión sistemática es un proceso que busca, evalúa y analiza estudios existentes, sintetizando datos para obtener conclusiones claras sobre el estado del conocimiento y las áreas pendientes por investigar (Denyer & Tranfield, 2009; Kitchenham, 2004).

3.2 Objetivos de revisión

Los siguientes son los objetivos que motivan esta revisión de literatura donde el tipo de revisión que se va a realizar, puede ser teórica, empírica o histórica:

● Identificar los estudios realizados en el área de ciencias de la computación que se han realizado sobre el tema de optimización de la logística de distribución de ayuda humanitaria hasta el momento (distribución de recursos como medicamentos, alimentos, agua, entre otros recursos que la población afectada pueda necesitar)

● Identificar los factores logísticos relevantes considerados actualmente para la distribución de ayuda en estos fenómenos. Estos factores abarcan aspectos logísticos, sociales y computacionales, fundamentales para plantear el problema en general.

● Identificar los problemas que afrontan los procesos, sistemas y algoritmos de distribución de ayuda humanitaria.

● Identificar las soluciones que se están usando en la actualidad para el tema de la distribución de ayuda humanitaria en estos casos.

● Identificar los algoritmos empleados en la planificación y toma de decisiones para la distribución de ayuda ante fenómenos naturales.

3.3 Preguntas de revisión

Se utilizará el método PICOC para organizar los elementos clave de las preguntas de investigación, describiendo sus cinco componentes principales (Petticrew, M., & Roberts, H., 2006.).

A continuación se describen los elementos PICOC escogidos:

Tabla 3. Elementos PICOC

Elemento	**Responde a**	**Descripción**
Population o población	¿Quién/es?	Soluciones para la planificación de distribución de ayuda humanitaria.
Intervention o intervención	¿Qué o cómo?	Implementar un algoritmo bioinspirado para optimizar la planificación de la distribución de ayuda en caso de fenómenos naturales
Comparison o comparación	¿Comparado con qué?	Algoritmos de optimización para la distribución más usados en problemas similares
Outcome o salida	¿Qué se está tratando de lograr o mejorar?	Lograr una mejora de la planificación para la distribución de ayuda en estos tipos de fenómenos naturales.
Context o contexto	¿En qué tipo de organización / circunstancias?	Esta investigación toma lugar en el ámbito académico y social.

Tomando en cuenta estos aspectos se formulan las siguientes preguntas de investigación :

P1. ¿Cuáles son los factores logísticos, ambientales y sociales a considerar para el tema de distribución de recursos en fenómenos naturales?

P2. ¿Cuáles son los principales problemas en la planificación de la distribución de ayuda humanitaria?

P3. ¿Qué soluciones se están utilizando para el tema de la distribución de ayuda humanitaria en la actualidad?

P4. ¿Qué algoritmos se están utilizando en los sistemas actuales para la solución del problema?

3.4 Estrategia de búsqueda

A continuación, se detallan los factores clave de la estrategia de búsqueda a emplear.

3.4.1 Motores de búsqueda a usar

- Scopus
- Springer
- IEEE Xplore Digital Library
- ScienceDirect
- Repositorio Tesis PUCP

Estas fueron seleccionadas según estos criterios:

- Son bases de datos accesibles para estudiantes, docentes y tesistas de la Pontificia Universidad Católica del Perú
- Son bases de datos con documentos variados en áreas como computación, informática e ingeniería.

3.4.2 Cadenas de búsqueda a usar

Se definen las siguientes palabras claves por cada elemento de la metodología PICOC:

Tabla 4. Palabras clave PICOC

Valor	Palabras clave
Population o población	- factors - distribution - natural disaster - issues
Intervention o intervención	- algorithm - methods - system

Comparison o comparación	- algorithm - distribution - optimization
Outcome o salida	- solution - response

La tabla incluye cadenas de búsqueda adaptadas para cada motor tras analizar los resultados. Debido a las variaciones en la cantidad y relevancia de los resultados, se utilizaron cadenas diferentes en cada motor. En algunos casos, se aplicaron criterios más específicos para obtener resultados alineados con el propósito del trabajo, y se verificó que los temas seleccionados correspondieran con el objetivo del estudio.

En Scopus:

TITLE-ABS-KEY (algorithm AND natural AND disaster AND help AND distribution)

En Springer:

TITLE-ABS-KEY (algorithm AND natural AND disaster AND help AND distribution)

En Springer se aplicaron dos filtros adicionales para obtener resultados en las áreas de Ingeniería y Ciencias de la Computación. Aunque el trabajo se enfoca en Ciencias de la Computación, la revisión de trabajos de Ingeniería proporcionó conceptos útiles y soluciones relevantes para la problemática planteada, enriqueciendo el enfoque del estudio.

En IIEE:

TITLE-ABS-KEY (algorithm AND natural AND disaster AND help AND distribution)

En ScienceDirect:

TITLE-ABS-KEY (algorithm AND natural AND disaster AND help AND distribution)

En Repositorio Tesis PUCP:

En este caso se buscó directamente la tesis del ingeniero Aduviri, cuya tesis es en la que se basa este proyecto de tesis.

3.4.3 Criterios de inclusión/exclusión

Esta revisión considera estudios que cumplen con los siguientes criterios de inclusión:

- El estudio aborda como un tema relacionado a la distribución de ayuda humanitaria en cualquier forma.
- El estudio pertenece al área de informática o ciencias de la computación.

Si no se hallan suficientes estudios recientes, se tomará en cuenta a las investigaciones con más de cinco años de antigüedad con un máximo de quince años de antigüedad.

En la presente revisión se excluyen los estudios que cumplen con los siguientes criterios de exclusión:

- El estudio aborda temas de predicción de riesgos sísmicos y no de la situaciones luego de que suceda alguna tragedia natural.
- El estudio aborda el tema de monitoreo más no la toma de decisiones para distribuir algún tipo de ayuda a la sociedad.
- El estudio se enfoca en aspectos no relacionados a lo que se quiere investigar como por ejemplo: resiliencia de la población a fenómenos naturales, encuestas, entre otros.
- De igual forma se están excluyendo los estudios que estén relacionados con el tema de distribución de energía, dado que es un tema que se repite muchas veces en la distribución de ayuda luego de un fenómeno natural y que no está alineado con el objetivo de este trabajo. Por lo tanto se usará la cadena: AND NOT ENERGY.
- El estudio está escrito en inglés o español, dado que el inglés es un idioma universal y estos son los idiomas más comunes en las investigaciones.
- El estudio fue realizado con cinco años de antigüedad o menos, esto debido a que se busca ejecutar una revisión del contenido más actualizado posible.

3.4.4 Documentos encontrados

A continuación, se presenta la cantidad de documentos encontrados usando las cadenas de búsqueda en cada motor, sin aplicar aún criterios de exclusión.

Tabla 5. Cantidad de resultados de búsqueda

Motor	Cantidad de resultado
Scopus	43
Springer (con Engineering)	196
Springer (con Computer Science)	236
IEEE Xplore Digital Library	329
ScienceDirect	298
Repositorio Tesis PUCP	1

Esta sección presenta el proceso de selección de estudios

a) Selección de documentos relevantes

Esta sección expone los resultados obtenidos tras aplicar el protocolo de revisión. En total, se encontraron 1103 publicaciones en diversas bases de datos, de las cuales se eliminaron 75 documentos duplicados con ayuda de Mendeley. Luego, se eligieron publicaciones en inglés o español de los últimos cinco años, reduciendo el total a 584. Para refinar la búsqueda, se excluyeron documentos sobre distribución de energía, obteniendo un total de 252 publicaciones. Posteriormente, se revisaron títulos y sumarios para eliminar aquellos que no cumplían con los criterios, quedando 81, y finalmente se seleccionaron 18 publicaciones directamente relacionadas con la problemática en cuestión. La Tabla 6 muestra el total de documentos localizados en las cinco bases de datos consultadas, considerando también 3 publicaciones de más de cinco años por su relevancia

Tabla 6. Cantidad total de documentos en todos los motores de búsqu*eda*

Descripción	Scopus	Springer (con	Springer (con	IEEE Xplore	Science Direct	Repositorio Tesis PUCP	Cantidad Total

		Engineering)	Computer Science)	Digital Library			
Documentos encontrados en las bases de datos	43	196	236	329	298	1	1103
Documentos duplicados	7	9	12	28	19	0	75
Documentos tras aplicar los criterios de inclusión y exclusión	15	34	59	81	62	1	252
Documentos seleccionados luego de una revisión rápida	8	12	14	29	17	1	81
Documentos relevantes a tomar en cuenta para el presente trabajo	5	3	4	2	3	1	18

b) Llenado del formulario de revisión

Esta fase consiste en extraer los documentos relevantes indicados y, mediante el llenado de un formulario de revisión, lograr una aproximación más precisa a las respuestas de las preguntas de investigación. Los documentos seleccionados se muestran a continuación:

Tabla 7. Documentos relevantes seleccionados

Id	Título	Autores	Año
E001	Multi-objective optimization considering quality concepts in a green healthcare supply chain for natural disaster response: neural network approaches	Mohammad Hossein Zavvar Sabegh, Mohammad Mohammadi & Bahman Naderi	2017
E002	Production-Routing-Inventory in Post-Disaster Conditions: a Multi-Objective Mathematical Model and Two Algorithms	Shima Zargary & Parvaneh Samouei	2022
E003	A modified particle swarm optimization for disaster relief logistics under uncertain environment	Ali Bozorgi-Amiri, Mohammad Saeid Jabalameli, Mehdi Alinaghian & Mahdi Heydari	2012
E004	A heuristic-based simulated annealing algorithm for	Sina Nayeri, Reza Tavakkoli-	2022

	the scheduling of relief teams in natural disasters	Moghaddam, Zeinab Sazvar & Jafar Heydari	
E005	A heuristic-based multi-choice goal programming for the stochastic sustainable-resilient routing-allocation problem in relief logistics	Zakie Mamashli, Ali Bozorgi-Amiri, Iman Dadashpour, Sina Nayeri & Jafar Heydari	2021
E006	Supply allocation: bi-level programming and differential evolution algorithm for Natural Disaster Relief	Ying-xin Chen, Pandu R. Tadikamalla, Jennifer Shang & Yan Song	2020
E007	Metaheuristic algorithms to allocate and schedule of the rescue units in the natural disaster with fatigue effect	Sina Nayeri, Ebrahim Asadi-Gangraj & Saeed Emami	2019
E008	Emergency logistics network optimization with time window assignment	Wang, Yonga; Wang, Xiuwenb;Fan, Jianxinc;Wang, Zhengd;Zhen, Lu	2023
E009	A scenario-based collaborative problem for a relief supply chain during post-disaster under uncertain parameters: a real case study in Dorud	Bakhshi, Alirezaa Bakhshi A.; Aghsami, Amirb Aghsami A.; Rabbani, Masoud	2022
E010	Ant colony optimization algorithm implementation for distribution of natural disaster relief logistics in Jombang regency web base	Ali M.;Sucipto H.	2021
E011	The optimization of warehouse location and resources distribution for emergency rescue under uncertainty	Wang, Bo Chena;Qian, Qi Yuanb;Gao, Jia Jingb;Tan, Zhe Yib Tan Z.Y.;Zhou, Yia	2021
E012	A disaster relief commodity supply chain network considering emergency relief volunteers: a case study	Kebriyaii, Omida Kebriyaii O.;Hamzehei, Marzieha .;Khalilzadeh, Mohammad	2021
E013	Introducing a novel multi-objective optimization model for vehicle routing and relief supply distribution in post-disaster phase: combining fuzzy inference systems with NSGA-II and NRGA	Peyman Rabiei; Daniel Arias-Aranda	2021
E014	Genetic Algorithm with Boosting based on Expected Value for Uncertain Routing	Toathom, T., Promsuk, N., & Champrasert, P	2021
E015	A collaborative humanitarian relief chain design for disaster response	Iman Shokr, Fariborz Jolai, Ali Bozorgi-Amiri	2022
E016	Stratified delivery aid plans for humanitarian aid distribution centre selection	Mohammed Nawazish a, Sidhartha S. Padhi b, T.C. Edwin Cheng c	2022
E017	Scenario-based redesigning of a relief supply-chain network by considering humanitarian constraints, triage, and volunteers' help	Zeinab Vosooghi a, S.M.J. Mirzapour Al-e-hashem b c, Behshad Lahijanian	2022
E018	Algoritmo genético multiobjetivo para la	Aduviri Choque, Robert Alonso	2019

	optimización de la distribución de ayuda humanitaria en caso de desastres naturales en el Perú		

3.5 Formulario de extracción de datos

A continuación se presenta el formulario de extracción de datos, el cual ayudará a describir cómo es que cada fuente seleccionada ayudará a contestar las preguntas de investigación planteadas.

Tabla 8. *Formulario de extracción*

Campo	**Descripción**	**Pregunta**
Id	E[número] P.ej: E001	General
Título		General
Autores		General
Año de publicación		General
Fuente	Nombre de la revista, congreso o libro	General
Tipo de fuente	Revista, congreso o capítulo de libro	General
Idioma	Inglés o español	General
Motor de búsqueda	Motor de búsqueda empleado: Scopus, Springer, IEEE, ScienceDirect, Repositorio Tesis PUCP	General
Abstract	Abstract del documento a mencionar	General
Factores se toman en cuenta para manejar la distribución de ayuda humanitaria	Factores que se toman en cuenta para la organización de la distribución de ayuda humanitaria.	P1
Problemas en la distribución de ayuda	Qué problemas se presentan cuando se desea repartir recursos o ayuda luego de un fenómeno natural	P2
Soluciones que actualmente se están empleando para este tema	De qué forma se está abordando actualmente el tema de distribución de ayuda humanitaria	P3
Algoritmos usados en las soluciones actuales	Los algoritmos que usan las soluciones que actualmente se emplean para brindar ayuda	P4

La Tabla 9 presenta un ejemplo del llenado del formulario.

Tabla 9. *Ejemplo de formulario de extracción*

Campo	**Descripción**
Id	E001
Título	Multi-objective optimization considering quality concepts in a green healthcare supply chain for natural disaster response: neural network approaches
Autores	Mohammad Hossein Zavvar Sabegh, Mohammad Mohammadi & Bahman Naderi
Año de publicación	2017
Fuente	International Journal of System Assurance Engineering and Management
Tipo de fuente	Revista
Idioma	Inglés
Motor de búsqueda	Springer
Abstract	This study proposes a new multi-objective mathematical model in pharmaceutical supply chain for natural disaster response considering quality, green concepts. The proposed model includes three objective functions. The first minimizes total manufacturing costs including production costs, purchasing costs, opening manufacturing plant costs, opening distribution centers costs, transportation costs and cost of poor quality (appraisal and prevention costs). The second minimizes environmental effects of products and transportations. The third maximizes humanitarian forces. Before disaster occurrence, to efficiently predict the objective functions values, we apply the back propagation (BP)—neural network, hybrid genetic algorithm (GA)—artificial neural network and particle swarm optimization (PSO). Finally, the effectiveness of the proposed solution shows the proposed multi objective optimization technique and its feasibility to be adopted as suitable methodology. The obtained results illustrate that the BP had high performance, which its R 2 was 0.99. Managerial implications of this research focus on improving the efficiency and effectiveness of the healthcare supply chain for natural disaster response: saving time, minimizing costs, minimizing environmental impact, utilizing resources more effectively (e.g. financial, human, technical, assets, transportation), showing social responsibility for communities affected by the disaster and continuously improving healthcare supply chain management.
Factores se toman en cuenta para manejar la distribución de ayuda humanitaria	El modelo propuesto incluye tres funciones objetivo que abordan factores importantes para la planificación de la distribución de ayuda. El primero es minimizar los costos totales de fabricación, incluidos los costos de producción, costos de compra, costos de apertura de plantas de fabricación, costos de apertura de centros de distribución, costos de transporte y costos de mala calidad (costos de evaluación y prevención). El segundo es minimizar los efectos ambientales de los productos y transportes. Y el tercero es maximizar las fuerzas humanitarias.

Problemas en la distribución de ayuda	Los conceptos ecológicos que impactan en el tema del transporte a los lugares donde la ayuda debe ir es uno de los problemas que se toma en cuenta en este modelo. De igual forma el tema del uso de recursos es un tema que se debe abordar de la mejor manera, ya que se deben usar eficientemente los recursos que se posean para brindar una respuesta rápida.
Soluciones que actualmente se están empleando para este tema	Las implicaciones gerenciales de esta investigación se enfocan en mejorar la eficiencia y eficacia de la cadena de suministro de atención médica para la respuesta a desastres naturales: ahorrar tiempo, minimizar costos, minimizar el impacto ambiental, utilizar los recursos de manera más efectiva (por ejemplo, financieros, humanos, técnicos, activos, transporte), mostrando responsabilidad social para las comunidades afectadas por el desastre y mejorar continuamente la gestión de la cadena de suministro de atención médica.
Algoritmos usados en las soluciones actuales	Antes de que ocurra un desastre, para predecir de manera eficiente los valores de las funciones objetivo, aplicamos la propagación hacia atrás (BP), red neuronal, algoritmo genético híbrido (GA), red neuronal artificial y optimización de enjambre de partículas (PSO).

En el Anexo 1 se encuentra el enlace al documento completo del formulario de extracción de datos.

3.6 Resultados de la revisión

Considerando la numeración de las publicaciones en la Tabla 7, se presenta un resumen de las que responden a las preguntas del documento:

Tabla 10. *Resumen de publicaciones que responden a las preguntas*

Pregunta	**Identificadores de publicaciones que responden a cada pregunta**
¿Qué factores se toman en cuenta para manejar la distribución de ayuda humanitaria?	E001, E003, E005, E006, E007, E011, E016
¿Cuáles son los problemas que se presentan para la distribución de ayuda humanitaria?	E001, E003, E006, E007, E008, E010, E012, E013
¿Cuáles son las soluciones que actualmente se están empleando para este tema?	E005, E007, E009, E013, E017
¿Qué algoritmos están siendo usados para abordar este tema?	E001, E002, E004, E007, E008, E013, E014, E015, E018

Seguidamente se procederá a contestar cada una de las preguntas planteadas:

3.6.1 Pregunta 1 - ¿Qué factores se toman en cuenta para manejar la distribución de ayuda humanitaria?

Durante y después de estos fenómenos naturales, es necesario considerar varios factores para la distribución de ayuda. A continuación se detallarán los principales factores encontrados en los documentos revisados:

1) Costo: Esto se debe a la necesidad de reducir los costos totales, incluyendo producción, compra, apertura de plantas y centros de distribución, transporte y costos por mala calidad (costos de evaluación y prevención) (Zavvar Sabegh et al., 2017). También se debe tener consideración integral de factores como el costo del tiempo, el costo de la penalización por falta de recursos, los orígenes alternativos de los recursos tanto de los proveedores como de los almacenes de emergencia, los diferentes medios de transporte y los múltiples tipos de recursos (B. C. Wang et al., 2021).
2) Tiempo: Otro factor sumamente importante también es el tema del transporte empleado para repartir los recursos. Se considera el tiempo de distribución de los recursos y la equidad para el tema de la asignación de ayuda a otorgar (Chen et al., 2020). Además, varios modelos consideran el tiempo total de viaje, los impactos ambientales totales y la pérdida de demanda total (Mamashli et al., 2021).
3) Efectos ambientales: Los efectos ambientales son importantes, ya que el clima puede provocar algún inconveniente en los recursos o no permitir la movilización de forma natural. También después de aliviar varios incidentes, los rescatistas se cansarán y necesitarán más tiempo para aliviar los incidentes restantes que les fueron asignados; por lo tanto, consideramos este fenómeno como efecto de fatiga en esta investigación (Nayeri et al., 2019).
4) Equidad e incertidumbre: Unos puntos que también se consideran en diversas publicaciones es el tema de la equidad y la incertidumbre. La equidad hace referencia a que se debe buscar poder ayudar de forma equitativa a las comunidades que tienen el mismo nivel de priorización. Con respecto a este punto es importante aclarar que cada punto de ayuda va a tener un nivel de priorización respectivo dependiendo de su nivel de necesidad. Por otro lado, se considera la

incertidumbre de los lugares donde pueden surgir esas demandas y la posibilidad de que el desastre destruya parcialmente algunas de las instalaciones (Bozorgi-Amiri et al., 2012).

5) Centros de distribución: Por último, un factor que se debe considerar es que en estas situaciones existen centros de distribución de ayuda humanitaria (HADC) que son esenciales para cerrar la brecha entre los beneficiarios varados y la ayuda humanitaria durante un desastre (Nawazish et al., 2022).

3.6.2 Pregunta 2 - ¿Cuáles son los problemas que se presentan para la distribución de ayuda humanitaria?

Los resultados muestran que diversos parámetros influyen en el diseño de una red de suministro para productos de socorro durante un fenómeno y también muchos parámetros deben controlarse para que la catástrofe se prevenga en gran medida y se puedan salvar las vidas de muchas personas enviando el producto de socorro a tiempo (Kebriyaii et al., 2021). Se describen los principales problemas identificados en los documentos revisados :

1) Impacto ecológico: Los fenómenos naturales afectan la infraestructura y las rutas de transporte, lo que supone un reto importante para la logística humanitaria. Los desastres pueden bloquear rutas, afectando los tiempos de respuesta y requiriendo ajustes en las rutas de distribución hacia áreas seguras (Zavvar Sabegh et al., 2017; Wang et al., 2023). La optimización de rutas en el problema de enrutamiento de vehículos (VRP) y la programación de distribución es esencial para disminuir los tiempos de respuesta (Rabiei & Arias-Aranda, 2021). En la práctica, también surgen dificultades al encontrar la ruta más corta para satisfacer las necesidades logísticas en el lugar del desastre (Ali & Sucipto, 2021).
2) Falta de información actual: Otro gran problema es que debido a la naturaleza de los fenómenos que ocurren intempestivamente, se tienen muchos datos que son inciertos y esto no permite poder considerar todos los casos para tomar las mejores soluciones posibles. La demanda de los suministros y el costo de adquisición y transporte se consideran parámetros inciertos (Bozorgi-Amiri et al., 2012).
3) Tiempo: Además, el tiempo para finalizar las actividades de ayuda es un problema que se toma

en cuenta, dado que se debe ayudar de forma lo más rápida posible (Nayeri et al., 2019).

Asimismo, en estos casos se tiene varios inconvenientes más como los siguientes: red vial dañada, alta demanda de diversos materiales, escasez de recursos, sitios de múltiples suministros y múltiples demandas, capacidad de transporte limitada, etc (Chen et al., 2020).

3.6.3 Pregunta 3 - ¿Cuáles son las soluciones que actualmente se están empleando para este tema?

Para este caso, hay diversas soluciones que se aplican para resolver este problema y cada una de ellas tienen consideraciones particulares de acuerdo con el modelo que realizan. A continuación, se listan algunas de las soluciones encontradas:

- Un modelo diseñado para reducir el tiempo total de desplazamiento, los impactos ambientales totales y la pérdida de demanda total. El enfoque de optimización estocástica robusta difusa se utiliza para hacer frente a datos inciertos que surgen en condiciones de fenómeno. Ante la complejidad del problema, se adopta un enfoque híbrido que integra programación de objetivos múltiples y un algoritmo heurístico para resolverlo eficientemente (Mamashli et al., 2021).
- Un modelo efectivo de asignación y programación de unidades de rescate puede disminuir pérdidas económicas y humanas en desastres naturales. Este trabajo propone un modelo de programación lineal de enteros mixtos para minimizar los tiempos ponderados de finalización y las demoras en operaciones de socorro (Nayeri et al., 2019).
- El documento presenta un modelo de programación entera mixta no lineal para maximizar la cobertura de demanda y reducir costos operativos y distancias recorridas (Bakhshi et al., 2022).
- Se propone un modelo que aborda dos problemas relacionados: el enrutamiento de vehículos (VRP) y la programación de la distribución de socorro en los puntos de demanda con respecto a tres funciones objetivo: La primera función objetivo es minimizar los costos operativos que incluyen costos fijos y variables de uso. flota heterogénea disponible según la ruta designada. Además, para una modelización más adecuada de la realidad en el complejo espacio de decisión de la fase de respuesta, se introducen en este estudio dos funciones objetivo cualitativas: la importancia de la demanda insatisfecha y la importancia de la demanda satisfecha tardíamente. Para evaluar los

índices cualitativos introducidos, empleamos sistemas de inferencia difusos (FIS) para encapsular el conocimiento y el proceso de razonamiento humano de los tomadores de decisiones (Rabiei & Arias-Aranda, 2021).

- El estudio presenta un modelo de ubicación y asignación, basado en escenarios, multiperíodo y multiobjetivo, para abastecer productos de socorro a puntos de demanda en condiciones de incertidumbre. El modelo no lineal propuesto se linealiza inicialmente y luego se resuelve aplicando la técnica de restricción épsilon. Debido a la propiedad de dureza NP del modelo, también se presentan un NSGA-II y un algoritmo híbrido para resolver instancias de mayor tamaño (Vosooghi et al., 2022).

Cada uno de estos trabajos realizan distintos enfoques en cuanto a los factores que toman para las decisiones para la repartición de ayuda humanitaria y gracias a esto se tienen distintas perspectivas de cómo abordar una misma problemática.

3.6.4 Pregunta 4 - ¿Qué algoritmos están siendo usados para abordar este tema?

Los algoritmos usados en las soluciones encontradas son muy variados. En esta revisión se encontraron dos familias de tipos de algoritmos que se usan con más frecuencia:

1) Algoritmos bioinspirados: Se encontró que varias de las publicaciones analizadas usan un algoritmo genético (puro, híbrido, NSGA-II) (Zavvar Sabegh et al., 2017; Rabiei & Arias-Aranda, 2021; Wang et al., 2023; Aduviri Choque & Robert Alonso, 2019; Toathom et al., 2021).

   Una de las publicaciones menciona que se usan los algoritmos de simulated annealing (SA) y colonia de abejas (BC) para problemas de tamaño grande. Los resultados mostraron que los algoritmos SA y BC proporcionaron respuestas cercanas, pero el algoritmo SA pudo lograr los resultados en menos tiempo que el algoritmo BC (Zargary & Samouei, 2022).

   Asimismo, también se hace uso del enjambre de partículas (PSO) y la colonia de hormigas que también son buenas opciones dado que permiten evaluar la población de datos bajo distintos enfoques. Por ejemplo, en uno de los trabajos se desarrollan tres algoritmos metaheurísticos, a saber, el algoritmo de simulated annealing (SA), el algoritmo de optimización de enjambre de

partículas (PSO) y un método basado en SA y PSO híbrido (SA-PSO), para resolver el problema de investigación (Nayeri et al., 2019).

2) Algoritmos metaheurísticos: De igual manera, algunos algoritmos empleados son algoritmos metaheurísticos los cuales en muchos casos los usan para comparar con otros. Como en uno de los casos donde usa un algoritmo desarrollado y tres algoritmos meta-heurísticos conocidos, los cuales luego los compara (Nayeri et al., 2022). Otra opción interesante que se observó en base a la investigación es una publicación en la que menciona que desarrollaron un algoritmo de relajación lagrangiana para resolver problemas de gran escala (Shokr et al., 2022).

3.7 Conclusiones

sta revisión de literatura brindó una perspectiva más clara sobre los métodos y estudios relacionados con la distribución de ayuda humanitaria en fenómenos naturales. Se pudo notar que los factores que más se toman en cuenta al momento del planteamiento de las soluciones son el tiempo, el costo y la planificación de rutas. Sin embargo, también varias soluciones toman en cuenta los factores de nivel de demanda de recursos, priorización y equidad. En base a todos estos factores mencionados, se va a plantear la solución para el presente trabajo.

De igual forma, pudimos identificar que varias soluciones utilizan algoritmos genéticos y últimamente se está optando por también emplear algoritmos de colmena, los cuales permiten tener buenos resultados y en un menor tiempo posible. De esta manera, se espera que un algoritmo bioinspirado, basado en el comportamiento de una colmena de abejas, genere resultados efectivos en poco tiempo, lo cual es crucial para la toma de decisiones en estos casos.

4. Definición de la variables y función objetivo

4.1 Introducción

Este capítulo aborda el primer objetivo específico: “Identificar y seleccionar las variables y restricciones más relevantes en la distribución de ayuda humanitaria en caso de fenómenos naturales”. Para lograrlo, se presentan dos resultados: la definición detallada de las variables y restricciones clave en la distribución de ayuda, y el desarrollo de la función objetivo para optimizar esta tarea, detallando sus

componentes principales. Estos resultados se describen en la sección de “Resultados alcanzados” y van acompañados de un análisis de los problemas asociados.

4.2 Resultados alcanzados

4.2.1 Definición de las variables y restricciones del problema

4.2.1.1 Introducción

Con el fin de diseñar algoritmos que mejoren la eficiencia en la distribución de asistencia humanitaria, es necesario inicialmente plantear el problema. Esto implica identificar qué aspectos se buscan optimizar, los parámetros que definen una instancia particular del problema, las variables que describen una solución factible y las restricciones que influyen en dichas variables.

4.2.1.2 Variables identificadas

A continuación se hará mención de algunas de las variables que se utilizarán dentro del algoritmo de colonia de abejas aplicado a la distribución de ayuda humanitaria. Dichas variables fueron identificadas en base a la revisión de la bibliografía encontrada en el estado del arte. En este caso se tomaron como referencia principalmente la tesis del ingeniero Aduviri Choque del 2019 y el artículo de Shima Zargary y Parvaneh Samouei del 2022.

- **Puntos de distribución**

Un componente fundamental del problema es la red de distribución, la cual se modela mediante un grafo en el que cada nodo corresponde a un punto de distribución (como almacenes o ubicaciones finales de recursos) y las aristas entre los nodos indican las conexiones o rutas posibles entre ellos. Esta red se describe mediante un conjunto de pares ordenados (j, k) que indican la conexión entre el nodo j y el nodo k.

- **Distancia con respecto a la carretera entre puntos de distribución**

Entre los puntos de distribución se van a presentar distancias o costos asociados con cada conexión entre dichos puntos, lo cual es vital tomar en consideración, ya que el objetivo es minimizar el tiempo total

necesario para distribuir los recursos a las áreas afectadas. En base a esto se buscará los caminos de distribución que sean lo más eficientes posibles.

- **Recursos Disponibles**

Para poder realizar la distribución de los recursos a las poblaciones necesitadas se debe tener un listado de los recursos que se tienen en los almacenes iniciales y la cantidad que se tiene de cada uno de dichos recursos.

- **Demanda de recursos en los puntos de distribución**

Al inicio del planteamiento del problema se necesita tener un listado de los recursos que se necesitan en cada una de las localidades que se busca ayudar y de igual modo se necesita tener registrado la cantidad que se necesita de cada uno de estos recursos para cubrir las necesidades de las comunidades.

- **Nivel de urgencia**

En la red de puntos de distribución cada uno de los puntos va a tener un nivel de urgencia determinado, el cual hace referencia el grado de celeridad con el que debe ser atendido dicho lugar. Un mayor nivel de urgencia hace referencia a que la población necesita ser atendida con mayor celeridad y con la mayor cantidad de recursos solicitados posibles.

4.2.1.3 Variables de decisión

Para la selección de estas variables de decisión se tomó en como referencia el artículo de Ali Bozorgi-Amiri, Mohammad Saeid Jabalameli, Mehdi Alinaghian & Mahdi Heydar del 2012.

- **Asignación de Recursos**

Cada punto de distribución a lo largo de una ruta debe recibir una asignación específica de recursos. Las variables de decisión relacionadas con la asignación de recursos especifican tanto la cantidad como el tipo de bienes que se distribuyen en cada punto. Es esencial optimizar esta asignación para satisfacer las necesidades específicas de cada punto y cumplir con las restricciones de capacidad de los vehículos.

- **Priorización de Puntos de Distribución**

Considerando la urgencia y el nivel de criticidad de envío de cada punto, las variables de decisión pueden determinar el orden de prioridad en el que se atienden los diferentes puntos de distribución. Esto es esencial para garantizar que los recursos se asignen primero a las áreas más críticas y urgentes.

4.2.1.4 Función objetivo

4.2.1.4.1 Maximizar Cobertura de Necesidades

El objetivo es maximizar la cobertura de las necesidades de los puntos de distribución. Busca distribuir los recursos de manera que se atienda a la mayor cantidad de lugares posible, incluso si no se pueden satisfacer todas las demandas por completo. Este enfoque tiene como objetivo principal asegurar que la ayuda humanitaria llegue a la mayor cantidad de áreas afectadas, maximizando así el impacto positivo.. En situaciones de desastres naturales, donde las necesidades son urgentes y diversas, es esencial maximizar la cobertura de las áreas afectadas. Garantizar que la ayuda llegue a la mayor cantidad posible de lugares contribuye a mitigar los efectos del desastre y a satisfacer las demandas críticas de la población afectada (Aduviri Choque, R., & Robert Alonso, 2019)..

La siguiente expresión indica la suma de la cantidad de unidades del recurso k que se llevaron el nodo i al nodo j multiplicado por el porcentaje de cobertura de la cantidad de unidades del recurso k llevados del nodo j al nodo k en base a la demanda de dicho en el nodo j.

$$\sum_{i,j,k} \text{Cantidad\_Recursos}_{i,j} * \text{Ponderación\_Cobertura}_{i,j,k}$$

4.2.1.4.2 Minimizar Costo de Transporte

Este objetivo se enfoca en mejorar la eficiencia logística en la distribución, reduciendo los costos vinculados al transporte de recursos entre los distintos puntos de entrega. Su meta principal es identificar rutas óptimas que disminuyan la distancia total recorrida y aprovechen al máximo los recursos disponibles. En el ámbito de la ayuda humanitaria, optimizar la logística resulta fundamental para gestionar de manera eficiente los recursos limitados. Reducir los costos de transporte no solo conserva fondos, sino que también facilita una distribución más rápida y eficaz de la asistencia a las zonas que lo

requieren. Este enfoque contribuye a maximizar el impacto de la ayuda al garantizar que los recursos lleguen a sus destinos de manera oportuna y eficiente (Y. Wang et al., 2023).

La expresión siguiente representa la suma del costo de transporte por unidad a lo largo del trayecto del nodo i al nodo j llevando el recurso k multiplicado por la cantidad de unidades del recurso k que se llevaron del nodo i al nodo j.

$$\sum_{i,j,k} \text{Costo\_Transporte}_{i,j,k} * \text{Cantidad\_Recursos}_{i,j}$$

4.2.1.4.3 Maximizar Atención Prioritaria a Urgencias con Equidad en la Distribución

La función objetivo busca maximizar la atención a las localidades más urgentes, promoviendo una distribución equitativa de los recursos. La idea es priorizar las áreas más críticas sin concentrar excesivamente los recursos en pocas localidades, logrando así un impacto más amplio y justo en la distribución de ayuda humanitaria (Aduviri Choque & Robert Alonso, 2019). Para esto, se emplea una medida de equidad que pondera la atención a las urgencias y penaliza las distribuciones desequilibradas. La fórmula incluye la raíz cuadrada de la suma de los porcentajes de satisfacción al cuadrado en cada punto, ajustada por la demanda, con el fin de reducir la inequidad.

La siguiente expresión indica la fórmula a emplear:

$$\left(1 - \sqrt{\frac{1}{N} * \sum_k \left(\frac{\sum_{i,j} Cantidad\_Recursos_{i,j} * \text{Ponderación\_Urgencia}_k * Porcentaje\_Satisfacción_{i,j,k}}{Demanda_k}\right)^2}\right) +$$

$$\text{Peso\_Urgencia} * \sum_k \text{Ponderación\_Urgencia}_k * \text{Porcentaje\_Cobertura\_Total}_k$$

Se emplea el promedio de los cuadrados de los promedios de satisfacción de los recursos k que se van a repartir de los nodos i a los nodos j. El término N hace referencia a la cantidad de nodos en la red de distribución de las poblaciones que necesitan ayuda. El término adicional referente el Peso_Urgencia con la Ponderación_Urgencia y Porcentaje_Cobertura_Total introduce una ponderación de urgencia para enfocarse en la atención prioritaria a las áreas más urgentes. Este componente adicional se maximiza, y al ser multiplicado por Peso_Urgencia, controla la importancia relativa de la atención a la urgencia frente a la equidad. El propósito principal de esta función es identificar soluciones que

maximicen la atención a las localidades urgentes mientras minimizan la inequidad en la distribución global de recursos.

4.2.1.4.4 Función objetivo total

La función objetivo que se utilizará para encontrar las soluciones óptimas del problema planteado utilizará las 3 funciones previamente explicadas en el presente informe. Para esto se utilizarán pesos asignados al valor de cada subfunción utilizada, con el objetivo de que se pueda decidir cuál de los criterios tiene mayor o menor relevancia con respecto a los otros al momento de tomar decisiones. La siguiente expresión indica la fórmula que se va a buscar maximizar para obtener los mejores resultados posibles:

$$(P1*Cobertura\_Necesidades) * (P2* 1/Costo\_Transporte) * (P3* Atencion\_Prioritaria)$$

En esta fórmula P1, P2 y P3 hacen referencia a los pesos asociados a cada subfunción objetivo.

4.2.1.5 Restricciones identificadas

Se tomaron como referencia principalmente la tesis del ingeniero Aduviri Choque del 2019 y el artículo de Ali Bozorgi-Amiri, Mohammad Saeid Jabalameli, Mehdi Alinaghian & Mahdi Heydar del 2012.

- **Capacidad de Almacenes**

Esta restricción tiene como objetivo asegurar que la cantidad de recursos almacenados en cada almacén no exceda su capacidad máxima. Evitar el sobrepasamiento de la capacidad de almacenamiento es crucial para mantener la eficiencia logística y garantizar que los almacenes operen dentro de límites físicos y estructurales aceptables. Esta restricción se aplica tanto a los almacenes que actúan como puntos de origen de los recursos como a los almacenes intermedios en la cadena de distribución. Para cada almacén j y cada tipo de recurso i, la cantidad de recursos del tipo i almacenados en j no debe superar la capacidad máxima del almacén j.

$$\sum_i Cantidad\_Recursos_{i,j} \leq Capacidad_j$$

- **Demanda Mínima Atendida**

Esta restricción garantiza que cada punto de distribución, ya sea un almacén final o un punto de demanda, reciba al menos una cantidad mínima de recursos para satisfacer sus necesidades básicas. Evitar que un punto quede desatendido es esencial para cumplir con los requisitos humanitarios y asegurar que todas las ubicaciones designadas reciban algún nivel de asistencia. Para cada punto de distribución k y cada tipo de recurso i, la cantidad de recursos del tipo i recibidos en k no debe ser inferior a una cantidad mínima predeterminada.

$$\sum_j \text{Cantidad\_Recursos}_{i,j} \geq \text{Demanda}_{i,k}$$

- **Disponibilidad de Recursos**

Esta restricción asegura que la cantidad de recursos enviados desde un almacén o punto de origen no exceda la cantidad disponible en ese lugar. Es esencial para evitar asignar más recursos de los que realmente están disponibles, lo que podría llevar a compromisos en la cadena de suministro.. Para cada almacén j y cada tipo de recurso i, la cantidad de recursos del tipo j transportados desde j hacia cualquier otro nodo no debe superar la cantidad disponible de recursos del tipo i en el almacén j.

$$\text{Cantidad\_Recursos}_{i,j} \leq \text{Inventario}_{i,j}$$

4.2.2 Validaciones realizadas

Se validaron las variables, restricciones y función objetivo de este trabajo en relación con la tesis del ingeniero Aduviri, con el fin de asegurar la comparabilidad entre ambas. Esto es esencial para proponer una solución más óptima a la distribución de recursos en desastres naturales, incorporando variables y funciones adicionales que mejoran tanto la programación como la eficiencia en la distribución. En el Anexo 3 se incluye el enlace al documento completo de validación y a las pruebas realizadas sobre los valores de la función objetivo en determinados casos.

4.3 Discusión

En este capítulo se lograron los resultados esperados R1 y R2, el primero se aborda en los ítem 4.2.1.1, 4.2.1.2 y 4.2.1.3, mientras que el segundo se describe en el apartado 4.2.1.4.

Los resultados obtenidos permiten establecer un marco preliminar de parámetros clave y condiciones esenciales que servirán como base para la formulación de estructuras de datos y algoritmos en etapas posteriores. Asimismo, la definición de la función objetivo resulta fundamental, ya que es responsable de analizar y valorar las posibles soluciones al problema.

La consistencia de estos resultados se alinea con investigaciones anteriores y casos de estudio presentados en el Estado del Arte, los cuales detallan la implementación de algoritmos en problemas con una complejidad comparable. Dichos estudios igualmente inician con la identificación de parámetros, restricciones y la formulación de la función objetivo respectiva.

5. Estructuras de datos

5.1 Introducción

En este capítulo se aborda el desarrollo del resultado esperado R3, el cual está vinculado al diseño de las estructuras de datos que respaldan la implementación del algoritmo de colonia de abejas y que también serán usados, con pequeñas adaptaciones según sea el caso, en el algoritmo genético con el que se desea comparar.

5.2 Resultados alcanzados

Este apartado expone en detalle las acciones llevadas a cabo para lograr el resultado esperado R3: "Definición de las estructuras de datos que utilizará el algoritmo de colonia de abejas propuesto" que también servirá para el desarrollo del algoritmo genético contra el que se va a comparar.

5.2.1 Diseño de estructuras de datos que sirven de soporte para el algoritmo de colonia de abejas propuesto

El algoritmo de colonia de abejas adaptado para planificar la distribución de ayuda en situaciones de desastres naturales en el Perú requiere de estructuras de datos que permitan organizar y gestionar la información relevante para el problema. A continuación, se describen las principales estructuras diseñadas para este propósito:

1. Plan de Distribución

El Plan de Distribución representa la distribución de ayuda para un lugar específico afectado por el fenómeno natural. Esta estructura incluye información detallada sobre el lugar, su nivel de urgencia y los recursos demandados en ese lugar. Además, proporciona una visión clara de las necesidades prioritarias en cada área afectada, lo que contribuye a una mejor toma de decisiones durante la planificación y realización de las acciones de ayuda.

2. Distancias entre Lugares

Para optimizar la logística de distribución de ayuda, es crucial tener en cuenta las distancias entre los lugares afectados. La estructura de Distancias entre Lugares permite almacenar esta información de manera organizada. Cada entrada en esta estructura incluye el lugar de origen y un mapa que asocia los lugares de destino con las distancias correspondientes. Esta información es fundamental para determinar las rutas más eficientes y minimizar los tiempos de transporte durante la distribución de ayuda.

3. Solución de Colonia de Abejas

que incluye varios Planes de Distribución específicos para diferentes lugares. Al combinar estos planes, se obtiene una estrategia global de distribución de ayuda, permitiendo evaluar y comparar soluciones para determinar la más adecuada en situaciones de emergencia. Esta estructura de datos permite organizar y gestionar eficientemente la información requerida para planificar la distribución de ayuda en casos de desastres naturales en Perú, mejorando la implementación y ejecución del algoritmo.

5.2.2 Estructuras de datos auxiliares

Junto con las estructuras específicas del algoritmo, se utilizarán estructuras auxiliares destinadas al almacenamiento de información necesaria para la implementación de los algoritmos de abeja y genético. El detalle de estas estructuras se encuentra en el Anexo 4.

5.2.3 Validaciones realizadas

Este capítulo ha sido evaluado por el especialista en algoritmia, quien ha aprobado la información presentada. La evidencia de dicha revisión se encuentra en el Anexo 5.

5.3 Discusión

En este capítulo se ha logrado el resultado esperado R3. Los alcances obtenidos permiten desarrollar una representación adecuada de las soluciones para cada uno de los algoritmos analizados en este proyecto. Este avance resulta crucial, ya que el diseño implementado debe permitir una rápida adaptación de las soluciones potenciales, lo que contribuye significativamente a optimizar el rendimiento de los algoritmos..

En comparación con investigaciones previas y casos de estudio analizados en el Estado del Arte, se ha observado que solo una parte de ellos proporciona una definición específica sobre la representación de una solución para el algoritmo utilizado. En su mayoría, después de exponer las restricciones del problema y la función objetivo, proceden a explicar la adaptación del algoritmo de manera directa.

6. Adaptación del algoritmo de colonia de abejas

6.1 Introducción

En este capítulo se aborda el desarrollo del resultado esperado R4, el cual está vinculado a la adaptación de un algoritmo basado en la colonia de abejas para abordar el problema de distribución de ayuda. Se presenta el detalle de flujo que seguiría la búsqueda de la solución al problema.

6.2 Resultados alcanzados

Este apartado expone en detalle las acciones llevadas a cabo para lograr el resultado esperado R4: "Adaptación de un algoritmo inspirado en la colonia de abejas para resolver el problema de distribución de ayuda".

6.3 Algoritmo de colonia de abejas adaptado al problema de distribución de ayuda

En este capítulo se aborda el desarrollo del resultado esperado R4, enfocado en la adaptación de un algoritmo de colonia de abejas para resolver el problema de distribución de ayuda. Se detalla el flujo que guiaría la búsqueda de la solución al problema. El pseudocódigo se elaboró tomando como referencia el presentado en la tesis de la ingeniera Silvana Yanet García, mencionado en la Figura 1 de este documento.

6.3.1 Pseudocódigo general del algoritmo de colonia de abejas

El esquema general del funcionamiento del algoritmos es el siguiente:

Figura 2. Pseudocódigo general de algoritmo de colonia de abejas

```
1. AlgoritmoColoniaAbejas(N, Cmax, Limite):
2. población = InicializarPoblación(N)
3. mejorSolución=None
4. Para c desde 1 hasta Cmax hacer:
5.     nuevasSolucionesEmpleadas = GenerarNuevasSolucionesEmpleadas(población, red, recursos)
6.     EvaluarAptitud(nuevasSolucionesEmpleadas)
7.     mejorSolución= ObtenerMejorSolución(mejorSolución,nuevasSolucionesEmpleadas)
8.     solucionesObservadoras = SeleccionarSolucionesObservadoras(población)
9.     nuevasSolucionesObservadoras = GenerarNuevasSolucionesObservadoras (solucionesObservadoras, red,
       recursos)
10.    EvaluarAptitud(nuevasSolucionesObservadoras)
11.    mejorSolución = ObtenerMejorSolución(mejorSolución, nuevasSolucionesObservadoras)
12.    población = ReemplazarFuentesAbandonadas(población, Limite)
13. Retornar mejorSolución
```

La función AlgoritmoColoniaAbejas es el núcleo del algoritmo, encargado de buscar soluciones óptimas. Recibe tres parámetros: N (tamaño de la población o número de soluciones candidatas), Cmax (número máximo de ciclos), y Límite (número de ciclos sin mejora antes de detenerse). El procedimiento comienza con la generación de una población de soluciones aleatorias a través de la función InicializarPoblación, representando distintas formas de distribuir recursos en la red de puntos de distribución. Luego, se ejecuta un ciclo principal de Cmax iteraciones en el que se desarrollan varias etapas de optimización.. Dentro de este ciclo, se llevan a cabo varias etapas:

1. **Generación de nuevas soluciones para las abejas empleadas:** Se generan nuevas soluciones mediante la exploración local de las soluciones existentes. Este proceso se realiza para encontrar vecindarios cercanos y mejorar las soluciones actuales.
2. **Evaluación de la aptitud de las soluciones empleadas:** Cada nueva solución generada es evaluada para determinar su calidad en términos de los objetivos de optimización establecidos en las funciones objetivos.
3. **Selección de soluciones para las abejas observadoras:** Se seleccionan soluciones de la población para ser exploradas por las abejas observadoras. Estas soluciones seleccionadas tienen en cuenta la calidad y la diversidad de las soluciones existentes.

4. **Generación de nuevas soluciones para las abejas observadoras:** Las abejas observadoras generan nuevas soluciones explorando vecindarios cercanos a las soluciones seleccionadas. Este proceso busca mejorar aún más las soluciones existentes y diversificar la búsqueda.
5. **Evaluación de la aptitud de las soluciones observadoras:** Se evalúan las nuevas soluciones generadas por las abejas observadoras para determinar su calidad y compararlas con las soluciones existentes en la población.
6. **Verificación y reemplazo de fuentes abandonadas:** Se comprueba si algunas soluciones no han mejorado durante un número de ciclos definido por el parámetro Límite. Si se cumple esta condición, estas soluciones se reemplazan por nuevas soluciones generadas aleatoriamente para mantener la diversidad y la exploración del espacio de búsqueda.

El ciclo se repite Cmax veces o hasta que no se mejore la mejor solución durante Límite iteraciones consecutivas. Finalmente, la función retorna la mejor solución obtenida a lo largo del proceso, la cual representa la distribución óptima de recursos para planificar la entrega de ayuda en situaciones de desastres naturales en el Perú.

6.3.2 Construcción de la población inicial

Figura 3. Pseudocódigo de construcción de población inicial

```
1. InicializarPoblación(N: tamaño población):
2. Población = []
3. Para i = 1 hasta N hacer
4.    Solución = GenerarSoluciónAleatoria()
5.    Población.agregar(Solución)
6. Retornar Población
```

La función InicializarPoblación crea la población inicial de soluciones para el algoritmo de colonia de abejas. A partir del tamaño de la población (N), genera N soluciones candidatas aleatorias usando la función GenerarSoluciónAleatoria, que representa posibles distribuciones de recursos. Cada solución se

agrega a la lista Población, y una vez completada, esta lista es retornada como la población inicial para iniciar la búsqueda de la solución óptima, proporcionando diversidad en las soluciones.

6.3.3 Generación de soluciones empleadas

Figura 4. Pseudocódigo de generación de soluciones empleadas

```
1. GenerarNuevasSolucionesEmpleadas(Población, red, recursos):
2. nuevasSoluciones = []
3. Para cada solución en Población hacer
4.     Para cada abeja empleada en la solución hacer
5.     nuevaSolución = GenerarSoluciónVecina(abeja, red, recursos)
6.     nuevasSoluciones.agregar(nuevaSolución)
7. Retornar nuevasSoluciones
```

La función GenerarNuevasSolucionesEmpleadas genera nuevas soluciones para las abejas empleadas en la población actual del algoritmo de colonia de abejas. Recibe como entrada la población actual, la red de distribución y los recursos disponibles. Para cada solución en la población, cada abeja empleada genera una solución vecina modificada mediante la función GenerarSoluciónVecina, la cual se basa en una estrategia de búsqueda específica. Las nuevas soluciones generadas se almacenan en la lista NuevasSoluciones, que es finalmente retornada. Esta función es crucial para explorar el espacio de búsqueda, diversificar las soluciones y mejorar los resultados del algoritmo.

6.3.4 Selección de soluciones observadoras

```
1. SeleccionarSolucionesObservadoras(Población):
2. SolucionesObservadoras = []
3. Para cada solución en Población hacer
4.     Para cada abeja observadora en la solución hacer
5.         Si la abeja observadora decide explorar basada en la probabilidad de selección entonces
6.             SolucionesObservadoras.agregar(Solución)
7. Retornar SolucionesObservadoras
```

Figura 5. Pseudocódigo de selección de soluciones observadoras

La función SeleccionarSolucionesObservadoras elige un subconjunto de soluciones de la población actual para que las abejas observadoras las exploren y generen nuevas soluciones. Recibe como entrada la población, la red de distribución y los recursos. Las soluciones se eligen basadas en una probabilidad de selección, que puede depender de la aptitud de la solución o el nivel de feromona. La función retorna la lista SolucionesObservadoras, que contiene las soluciones elegidas para ser exploradas.

6.3.5 Generación de soluciones observadoras

```
1. GenerarNuevasSolucionesObservadoras(SolucionesObservadoras, Red, Recursos):
2. NuevasSolucionesObservadoras = []
3. Para cada solución en SolucionesObservadoras hacer
4.     Para cada abeja observadora en la solución hacer
5.         NuevaSolución = GenerarNuevaSoluciónObservadora(abeja observadora, Red, Recursos)
6.         Evaluar la calidad de la nueva solución
7.         Si la calidad de la nueva solución es mejor que la solución original entonces
8.             Reemplazar la solución original por la nueva solución en la población
9.             NuevasSolucionesObservadoras.agregar(nueva solución)
10.        Sino
11.            Mantener la solución original en la población
12.            NuevasSolucionesObservadoras.agregar(solución original)
13. Retornar NuevasSolucionesObservadoras
```

Figura 6. Pseudocódigo de generación de soluciones observadoras

La función GenerarNuevasSolucionesObservadoras genera nuevas soluciones basadas en las soluciones previas de las abejas observadoras en el algoritmo de colonia de abejas. Recibe la lista de soluciones observadoras, la red de distribución y la disponibilidad de recursos. Para cada solución, se crea una nueva solución utilizando la información de la red y los recursos disponibles, y se evalúa su calidad. Si la nueva solución supera la original, esta se reemplaza; de lo contrario, se conserva la solución original. Esto retorna la lista NuevasSolucionesObservadoras con las soluciones generadas.

6.3.6 Evaluación de aptitud

Figura 7. Pseudocódigo de evaluación de aptitud

```
1. EvaluarAptitud(Población)
2. Para cada individuo en la población hacer
3.     Calcular la aptitud del individuo según los objetivos del problema
4.     Almacenar la aptitud calculada para el individuo
```

La función EvaluarAptitud evalúa la calidad de cada solución en la población generada por el algoritmo de colonia de abejas. Itera sobre cada individuo, calculando su aptitud en función de métricas específicas del problema, como minimizar el costo de transporte o maximizar la cobertura de necesidades. Finalmente, la aptitud calculada se asocia a cada individuo.

6.3.7 Obtención de la mejor solución

```
1. ObtenerMejorSolución(mejorSolución, nuevasSolucionesEmpleadas)
2. mejor_solución_actual = mejorSolución
3. Para cada Solución en nuevasSolucionesEmpleadas hacer
4.     Si mejor_solución_actual es un individuo vacío o Aptitud(Solución) > Aptitud(mejor_solución_actual)
       entonces
5.         mejor_solución_actual = Solución
6. Retornar mejor_solución_actual
```

Figura 8. Pseudocódigo de obtención de la mejor solución

La función ObtenerMejorSolución identifica y retorna la mejor solución en términos de aptitud dentro de la población generada por el algoritmo de colonia de abejas. Compara la aptitud de cada individuo con la mejor solución actual y actualiza esta última si se encuentra una mejor opción. Al final, devuelve la solución con la mayor aptitud de toda la población.

6.3.8 Reemplazo de fuentes abandonadas

```
1. ReemplazarFuentesAbandonadas(población, límite)
2. fuentes_abandonadas = 0
3. Para cada Solución en población hacer
4.     Si Solución no ha mejorado en las últimas límite generaciones entonces
5.         Generar una nueva solución aleatoria y reemplazarla por la solución actual
6.         Incrementar fuentes_abandonadas en 1
7. Retornar población
```

Figura 9. Pseudocódigo de reemplazo de fuentes abandonadas

La función ReemplazarFuentesAbandonadas actualiza la población reemplazando soluciones que no han mejorado en un número de generaciones definido por el límite. Itera sobre cada solución, y si no muestra mejora, genera una nueva solución aleatoria en su lugar, aumentando el contador de fuentes abandonadas. Esta función es clave para preservar la diversidad dentro de la población, previniendo el estancamiento y promoviendo una mejor exploración del espacio de búsqueda.

6.3.9 Validaciones

Se llevaron a cabo pruebas de caja blanca para soluciones de tamaño reducido, lo que permitió analizar las funcionalidades del algoritmo y confirmar su efectividad en la creación de un plan de distribución de ayuda ante desastres naturales. Un ejemplo detallado se incluye en el Anexo 6.

Este capítulo fue evaluado y aprobado por el especialista en algoritmia, cuya evidencia de revisión se encuentra en el Anexo 7.

6.4 Discusión

En este capítulo se ha logrado el cumplimiento del resultado esperado R4, referente a la adaptación de un algoritmo de colonia de abejas para resolver el problema de distribución de ayuda.

Los logros obtenidos en este contexto permiten entender el funcionamiento general del algoritmo de colonia de abejas, así como los detalles de cada uno de sus operadores principales. Este avance sienta una base sólida de diseño que será empleada en la futura implementación del algoritmo utilizando herramientas de programación..

En relación con investigaciones previas y casos de estudio revisados en el Estado del Arte, se ha identificado que algunos describen en detalle la adaptación del algoritmo propuesto al caso de estudio, así como los ajustes implementados por diversos investigadores para optimizar su rendimiento durante la ejecución.

7. Adaptación del algoritmo de genético

7.1 Introducción

En este capítulo se aborda, de manera similar al capítulo 6, la adaptación del problema a un algoritmo; sin embargo, en esta sección se adopta un enfoque basado en un algoritmo genético para abordar el problema de distribución de ayuda. Se presenta el detalle de flujo que seguiría la búsqueda de la solución al problema.

7.2 Resultados alcanzados

Este apartado expone en detalle las acciones llevadas a cabo para lograr la adaptación de un algoritmo genético al problema de distribución de ayuda.

7.3 Algoritmo de genético adaptado al problema de distribución de ayuda

En este capítulo se lleva a cabo, de manera similar al capítulo 6 la adaptación del problema a un algoritmo, pero con enfoque de algoritmo genético . Se presenta el detalle de flujo que seguiría la búsqueda de la solución al problema.

7.3.1 Pseudocódigo general del algoritmo genético

El esquema general que describe el funcionamiento del algoritmo es el siguiente:

Figura 10. Pseudocódigo general de algoritmo genético

```
1. AlgoritmoGenético (poblaciónTamaño, numGeneraciones):
2.    población = InicializarPoblación(poblaciónTamaño)
3.    Para generación desde 1 hasta numGeneraciones hacer:
4.       EvaluarAptitud(población)
5.       nuevapoblacion = []
6.       Para i desde 1 hasta poblaciónTamaño/2 hacer:
7.          padres = SeleccionarPadres(población)
8.          padre1, padre2 = padres
9.          hijo1, hijo2 = AplicarCrossover (padre1, padre2)
10.            Mutar(hijo1)
11.            Mutar(hijo2)
12.            EvaluarAptitud(hijo1)
13.            EvaluarAptitud(hijo2)
14.            nuevapoblacion.Agregar(hijo1)
15.            nuevapoblacion.Agregar(hijo2)
16.      Fin Para
17.      población = nuevapoblacion
18.   Fin Para
19.   mejorSolución = ObtenerMejorSolución(población)
20.   Retornar mejorSolución
```

La función AlgoritmoGenético es el núcleo del algoritmo y se encarga de buscar soluciones óptimas. Recibe dos parámetros: poblaciónTamaño (número de soluciones candidatas) y numGeneraciones (máximo de generaciones). Inicia creando una población de soluciones aleatorias con InicializarPoblación, que representa distintas formas de resolver el problema. Luego, entra en un ciclo principal que se repite numGeneraciones veces y abarca varias etapas de optimización.. Dentro de este ciclo, se llevan a cabo varias etapas:

- **Evaluación de la aptitud de la población:** Se evalúa la calidad de cada solución en la población utilizando la función EvaluarAptitud.
- **Generación de una nueva población:** Se seleccionan padres de la población actual utilizando la función SeleccionarPadres, y se generan nuevos individuos mediante la función

AplicarCrossover. Además, los nuevos individuos pueden sufrir mutaciones mediante la función Mutar.

- **Evaluación de la aptitud de los nuevos individuos:** Los nuevos individuos generados son evaluados para determinar su calidad en términos de los objetivos de optimización establecidos.

Al finalizar cada generación, la población se actualiza con los nuevos cromosomas. El proceso se repite hasta que se alcanza el límite establecido de generaciones. Finalmente, la mejor solución de la población final se obtiene mediante la función ObtenerMejorSolución y se retorna como resultado del algoritmo.

7.3.2 Inicialización población del algoritmo genético

Figura 11. Pseudocódigo inicialización población de algoritmo genético

```
1. InicializarPoblación(poblaciónTamaño):
2.     población = []
3.     Para i desde 1 hasta poblaciónTamaño hacer:
4.         cromosoma = GenerarCromosomaAleatorio ()
5.         población.Agregar (cromosoma)
6.     Fin Para
7.     Retornar población
```

La función InicializarPoblación crea la población inicial de cromosomas, tomando como parámetro el tamaño deseado de la población (poblaciónTamaño). Genera cromosomas aleatorios mediante GenerarCromosomaAleatorio, los agrega a la lista población y retorna la población completa, garantizando diversidad en las soluciones iniciales del algoritmo.

7.3.3 Evaluación de aptitud

Figura 12. Pseudocódigo de evaluación de aptitud

```
1. EvaluarAptitud(población):
2.    Para cada cromosoma en población hacer:
3.       cromosoma.aptitud = CalcularAptitud(cromosoma)
4.    Fin Para
```

La función EvaluarAptitud calcula la aptitud de cada cromosoma dentro de la población, usando la función CalcularAptitud y asignando el valor resultante al atributo de cada cromosoma. Esto permite evaluar qué tan bien cada cromosoma resuelve el problema, facilitando la selección de los mejores individuos en futuras generaciones.

7.3.4 Selección de padres

Figura 13. Pseudocódigo de selección de padres

```
1. SeleccionarPadres(población):
2.    padres = []
3.    Para i desde 1 hasta 2 hacer:
4.       padre = SelecciónPorRuleta(población)
5.       padres.Agregar(padre)
6.    Fin Para
7.    Retornar padres
```

La función SeleccionarPadres elige dos padres de la población para el cruzamiento, utilizando la técnica de selección por ruleta, que da mayor probabilidad a los individuos con mejor aptitud. La lista con los dos padres seleccionados se retorna al final.

7.3.5 Cruzamiento

Figura 14. Pseudocódigo de cruzamiento de individuos

```
1. AplicarCrossover (padre1, padre2):
2.     puntoCorte = GenerarPuntoCorteAleatorio ()
3.     hijo1 = padre1[0: puntoCorte] + padre2[puntoCorte:]
4.     hijo2 = padre2[0: puntoCorte] + padre1[puntoCorte:]
5.     Retornar hijo1, hijo2
```

La función AplicarCrossover realiza el cruce entre dos padres (padre1 y padre2) para generar dos hijos. Utiliza un punto de corte aleatorio para combinar las partes de ambos padres en los hijos, permitiendo la mezcla de características. La función retorna los dos hijos generados, lo cual es esencial para aumentar la diversidad genética de la población.

7.3.6 Mutación

Figura 15. Pseudocódigo de mutación de individuos

```
1. Mutar(cromosoma):
2.     Para cada gen en cromosoma hacer:
3.         Si ProbabilidadDeMutación():
4.             gen = GenerarGenAleatorio ()
5.         Fin Si
6.     Fin Para
```

La función Mutar aplica mutación a un cromosoma, iterando sobre cada gen y utilizando una probabilidad para decidir si se debe generar un nuevo valor aleatorio con GenerarGenAleatorio. La mutación introduce variabilidad en la población, ayudando a evitar la convergencia en soluciones subóptimas y permitiendo la exploración de mejores soluciones.

7.3.7 Validaciones

Se llevaron a cabo pruebas de caja blanca aplicadas a soluciones de tamaño reducido, lo que permitió monitorear las funcionalidades del algoritmo y comprobar su efectividad en la elaboración de un plan de distribución de ayuda frente a desastres naturales. Un ejemplo detallado se encuentra en el Anexo 8.

Este capítulo fue evaluado y aprobado por el especialista en algoritmos, cuya evidencia de revisión está incluida en el Anexo 9.

7.4 Discusión

En este capítulo se ha logrado adaptar un algoritmo genético para abordar el problema de distribución de ayuda planteado.

Los resultados obtenidos en este contexto permiten comprender tanto el funcionamiento general del algoritmo genético como los detalles de cada uno de sus operadores principales. Este avance proporciona una base sólida de diseño que será empleada en la implementación futura del algoritmo utilizando herramientas de programación.

En relación con investigaciones previas y casos de estudio revisados en el Estado del Arte, se ha identificado que algunos trabajos describen la adaptación del algoritmo propuesto al caso de estudio, así como las modificaciones realizadas por diversos investigadores para optimizar su rendimiento durante la ejecución.

8. Codificación del algoritmo colonia de abejas

8.1 Introducción

Este capítulo está enfocado en el resultado 5, que consiste en la elaboración del código para el algoritmo de colonia de abejas adaptado al problema de distribución de ayuda. La codificación se ha llevado a cabo siguiendo los diseños previamente definidos, incluyendo el pseudocódigo del algoritmo y las estructuras de datos auxiliares. Se describirán los métodos principales del algoritmo, considerando aspectos técnicos esenciales y observaciones relevantes sobre las situaciones enfrentadas durante su

desarrollo. En el Anexo 10 se incluye el enlace URL para acceder al código del algoritmo de colonia de abejas.

8.2 Resultados alcanzados

Este apartado detalla las acciones realizadas para lograr el resultado esperado R5 : “Codificación del algoritmo de colonia de abejas adaptado al problema de distribución de ayuda”.

8.3 Codificación del algoritmo de colonia de abejas adaptado al problema de distribución de ayuda

De manera similar a la fase de diseño, la exposición del código del algoritmo de colonia de abejas se estructura en torno a los métodos principales que lo conforman. Esto se hace con el propósito de poder detallar y explicar lo que se ha realizado en cada uno de ellos.

8.3.1 Método principal para buscar los planes de distribución

Figura 16. Codificación de método principal para buscar los planes de distribución

```
vector<ResourceDistribution> findBestResourceDistribution(const vector<Resource>& resources,const vector<Stock>& stocks,
    const vector<string>& fromUbigeos,const vector<string>& toUbigeos,const vector<Ubigeo>& ubigeos, const int numbees,
    const int maxtterations,const int maxattempts,const int elitistcount, const double abandonmentthreshold) {

    const int numBees = numbees/2; // Número de abejas
    const int maxIterations = maxtterations/2; // Número máximo de iteraciones
    const int elitistCount = elitistcount; // Número de mejores soluciones para mantener
    const double abandonmentThreshold = abandonmentthreshold; // Umbral de abandono
    // Generador de números aleatorios
    random_device rd;
    mt19937 gen(rd());
    // Inicialización de la población de distribuciones
    vector<vector<ResourceDistribution>> population(numBees);
    vector<double> fitness(numBees);
    vector<vector<ResourceDistribution>> bestSolutions;
    // Mapa para almacenar la cantidad pedida de cada stock por recurso
    map<string, double> requestedAmountMap;
    // Inicialización de la población
    for (int i = 0; i < numBees; ++i) {
        cout<<"En la abeja "<<i+1<<" de findBestResourceDistribution"<<endl;
        population[i] = generateRandomDistribution(resources,stocks, fromUbigeos, toUbigeos, ubigeos, gen);
        fitness[i] = evaluateDistributionObjetive(population[i],stocks,resources,fromUbigeos,toUbigeos,ubigeos,requestedAmountMap);
    }
    // Iterar sobre las generaciones
    for (int iteration = 0; iteration < maxIterations; ++iteration) {
        cout<<"En la iteracion "<<iteration+1<<" de findBestResourceDistribution"<<endl;
        // Modificación de distribuciones
        for (int i = 0; i < numBees; ++i) {
            if (fitness[i] > abandonmentThreshold) {
                modifyDistribution(population[i], gen);
                fitness[i] = evaluateDistributionObjetive(population[i],stocks,resources,fromUbigeos,toUbigeos,ubigeos,
                requestedAmountMap);
            }
        }
        // Evaluar y mantener las mejores soluciones
        for (int i = 0; i < numBees; ++i) {
            if (bestSolutions.size() < elitistCount || fitness[i] < evaluateDistributionObjetive(bestSolutions.back(),stocks,resources,
                fromUbigeos,toUbigeos,ubigeos,requestedAmountMap)) {
                bestSolutions.push_back(population[i]);
                sort(bestSolutions.begin(), bestSolutions.end(), [&](const vector<ResourceDistribution>& a, const vector<ResourceDistribution>& b){
                    return evaluateDistributionObjetive(a,stocks,resources,fromUbigeos,toUbigeos,
                        ubigeos,requestedAmountMap) < evaluateDistributionObjetive(b,stocks,resources,fromUbigeos,
                        toUbigeos,ubigeos,requestedAmountMap);
                });
                if (bestSolutions.size() > elitistCount) {
                    bestSolutions.pop_back();
                }
            }
        }
    }
    return bestSolutions.front();
}
```

La Figura 16 muestra la codificación del método principal utilizado para buscar planes de distribución empleado el algoritmo de colonia de abejas, en el cual se consideró el número de abejas, el número máximo de iteraciones, el número de mejores soluciones para mantener y el umbral de abandono, variables establecidas en la etapa de diseño. Vemos que en primera instancia lo que se hace es inicializar la población con la función generateRandomDistribution que se explicará a detalle más adelante. De igual forma cada una de estas poblaciones inicializadas pasan por la función evaluateDistributionObjetive, la cual calcula el valor correspondiente a su valor objetivo definido para este problema. Seguidamente se pasa a iterar e ir modificando las soluciones iniciales en búsqueda de mejores opciones. Una vez terminadas estas iteraciones se devuelve la solución con el mayor valor de función objetivo.

8.3.2 Generación de planes de distribución aleatorios

Figura 17. Codificación de método para la generación de planes de distribución aleatorios

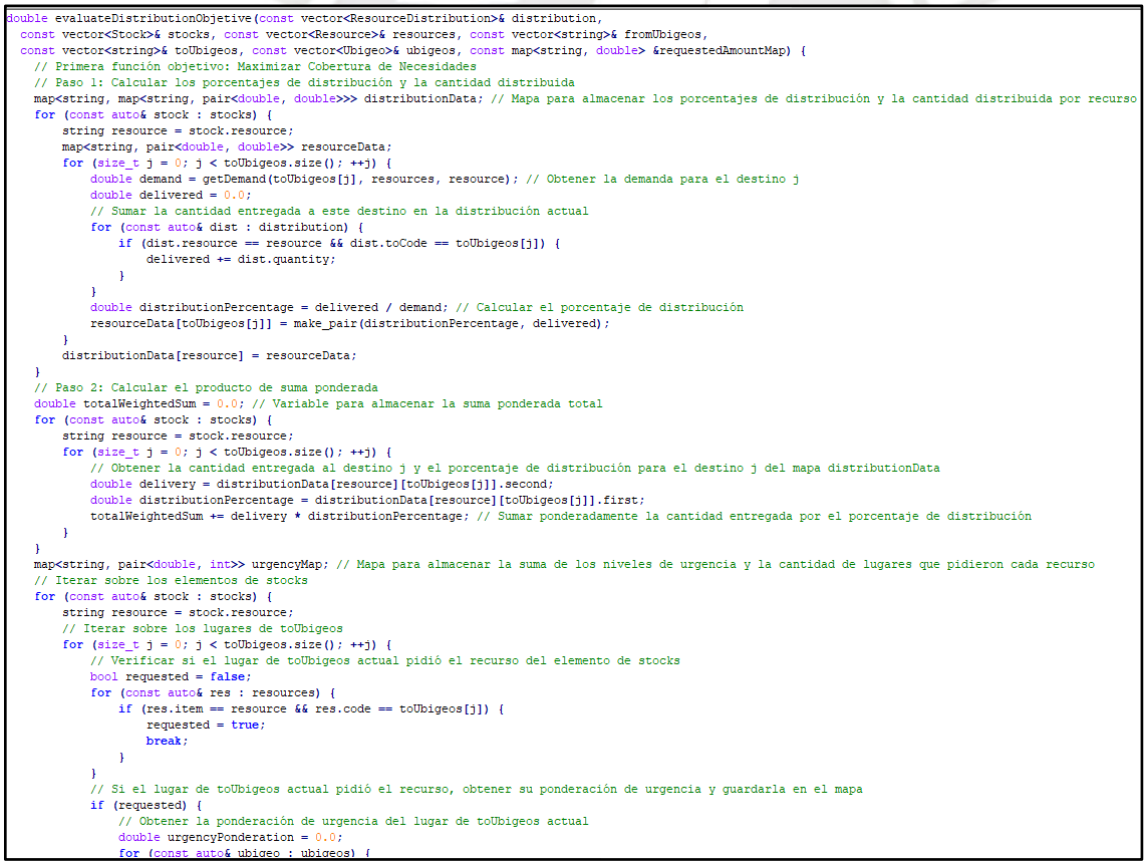
```
vector<ResourceDistribution> generateRandomDistribution(const vector<Resource>& resources,
   const vector<Stock>& stocks, const vector<string>& fromUbigeos, const vector<string>& toUbigeos,
   const vector<Ubigeo>& ubigeos, mt19937& gen) {

    vector<ResourceDistribution> distribution;
    // Calcular la demanda total de cada ubicación de destino para cada recurso
    unordered_map<string, double> totalDemandsByStock; // Stock -> Quantity
    unordered_map<string, unordered_map<string, double>> totalDemandsByStockToUbigeo; // Stock -> (Ubigeo -> Quantity)

    for (const auto& stock : stocks) {
        // Inicializar el total por elemento de stock
        totalDemandsByStock[stock.resource] = 0.0;

        for (const auto& resource : resources) {
            for (const auto& toCode : toUbigeos) {
                if (stock.resource == resource.item && resource.code == toCode){
                    // Sumar la cantidad de este recurso al total por elemento de stock
                    totalDemandsByStock[stock.resource] += resource.quantity;

                    // Sumar la cantidad de este recurso al total por código de toUbigeos por elemento de stock
                    totalDemandsByStockToUbigeo[stock.resource][toCode] += resource.quantity;
                    }
            }
        }
    }

    // Generar asignaciones aleatorias para cada ubicación de destino
    for (const auto& stock : stocks) {
        //primero verificamos si se puede cubrir toda la demanda:
        double total=0.0;
        // Iterar sobre los códigos de destino
        for (const auto& toCode : toUbigeos) {
            for (const auto& resource : resources) {
                if (resource.code == toCode && stock.resource==resource.item) {
                    total+=resource.quantity;
                }
            }
        }
```

En la Figura 17 se puede observar parte de la codificación de la función encargada de la generación de planes de distribución aleatorios. En esta función lo primero que se realiza es almacenar la cantidad de

recursos se pidieron en cada uno de los lugares destinos por cada uno de los elementos que se tiene en el stock. Se almacena tanto a nivel de lugar distinto como en general de la cantidad pedida por cada elemento del stock, esto nos servirá para luego de la generación de cantidad aleatorias poder verificar que no se sobrepasa del stock de los recursos en los almacenes y tampoco se exceda de lo solicitado de cada elemento en los lugares. Luego se procede a realizar una asignación para cada tipo de producto en el stock general y en este proceso se evalúa si se puede cubrir o no con la demanda total de todos los lugares destinos que solicitaron este producto. En base a si se puede o no cubrir la demanda total se procede con asignaciones aleatorias controladas para cada uno de los casos, en los cuales se verifica que no se sobrepase los stock permitidos al momento de asignar la cantidad a entregar en cada lugar.

8.3.3 Función de evaluación de las soluciones generadas

Figura 18. Codificación de función de evaluación de las soluciones generadas



```
double evaluateDistributionObjetive(const vector<ResourceDistribution>& distribution,
  const vector<Stock>& stocks, const vector<Resource>& resources, const vector<string>& fromUbigeos,
  const vector<string>& toUbigeos, const vector<Ubigeo>& ubigeos, const map<string, double> &requestedAmountMap) {
    // Primera función objetivo: Maximizar Cobertura de Necesidades
    // Paso 1: Calcular los porcentajes de distribución y la cantidad distribuida
    map<string, map<string, pair<double, double>>> distributionData; // Mapa para almacenar los porcentajes de distribución y la cantidad distribuida por recurso
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        map<string, pair<double, double>> resourceData;
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            double demand = getDemand(toUbigeos[j], resources, resource); // Obtener la demanda para el destino j
            double delivered = 0.0;
            // Sumar la cantidad entregada a este destino en la distribución actual
            for (const auto& dist : distribution) {
                if (dist.resource == resource && dist.toCode == toUbigeos[j]) {
                    delivered += dist.quantity;
                }
            }
            double distributionPercentage = delivered / demand; // Calcular el porcentaje de distribución
            resourceData[toUbigeos[j]] = make_pair(distributionPercentage, delivered);
        }
        distributionData[resource] = resourceData;
    }
    // Paso 2: Calcular el producto de suma ponderada
    double totalWeightedSum = 0.0; // Variable para almacenar la suma ponderada total
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            // Obtener la cantidad entregada al destino j y el porcentaje de distribución para el destino j del mapa distributionData
            double delivery = distributionData[resource][toUbigeos[j]].second;
            double distributionPercentage = distributionData[resource][toUbigeos[j]].first;
            totalWeightedSum += delivery * distributionPercentage; // Sumar ponderadamente la cantidad entregada por el porcentaje de distribución
        }
    }
    map<string, pair<double, int>> urgencyMap; // Mapa para almacenar la suma de los niveles de urgencia y la cantidad de lugares que pidieron cada recurso
    // Iterar sobre los elementos de stocks
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        // Iterar sobre los lugares de toUbigeos
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            // Verificar si el lugar de toUbigeos actual pidió el recurso del elemento de stocks
            bool requested = false;
            for (const auto& res : resources) {
                if (res.item == resource && res.code == toUbigeos[j]) {
                    requested = true;
                    break;
                }
            }
            // Si el lugar de toUbigeos actual pidió el recurso, obtener su ponderación de urgencia y guardarla en el mapa
            if (requested) {
                // Obtener la ponderación de urgencia del lugar de toUbigeos actual
                double urgencyPonderation = 0.0;
                for (const auto& ubigeo : ubigeos) {
```

En la Figura 18 se puede observar parte de la codificación de la función de evaluación de las soluciones generadas. Con respecto a la función objetivo, en el desarrollo del programa se decidió partir en dos el

algoritmo de colonia de abejas. La búsqueda de los planes de distribución y de las rutas para las entregas se realizan de forma separado, debido a que si se busca la combinación óptima de plan de distribución y de plan de rutas al mismo tiempo esto conllevaría a que se necesitarían de muchas iteraciones más para poder encontrar una solución correcta. Por ende en esta parte del código se realizan los cálculos respectivos para obtener los valores de la primera y tercera función objetivo indicados en la formulación general de la función objetivo para el presente trabajo.

8.3.4 Modificación de planes de distribución generados

Figura 19. Codificación de función de modificación de planes de distribución generados

```
void modifyDistribution(vector<ResourceDistribution>& distribution, mt19937& gen) {
    // Esta función puede implementar diferentes operadores de modificación para explorar nuevos vecinos
    // Por ejemplo, se puede intercambiar las cantidades de recursos entre ubicaciones
    uniform_int_distribution<size_t> distIndex(0, distribution.size() - 1);
    size_t index1 = distIndex(gen);
    size_t index2 = distIndex(gen);

    // Intercambiar las cantidades de recursos entre las dos ubicaciones
    swap(distribution[index1].quantity, distribution[index2].quantity);
}
```

En la Figura 19 se puede observar la función encargada de modificación de planes de distribución generados. Esta función nos permite explorar nuevas versiones posibles generadas en base a la modificación de soluciones previas.

8.3.5 Método general para buscar los planes de distribución

```
vector<DistributionPlanDistance> processUbigeoCombinations(const vector<Ubigeo>& ubigeos, const vector<string>& fromUbigeos,
  const vector<string>& toUbigeos, const int numbees, const int maxtterations,
  const int maxattempts, const int elitistcount, const double abandonmentthreshold) {

   vector<DistributionPlanDistance> result;
   ofstream outFile("Detalle de rutas.txt");
   if (!outFile.is_open()) {
       cerr << "Error al abrir el archivo Plan de distribucion.txt" << endl;
       exit(1);
   }
   int i=1;
   // Iterar sobre cada código en fromUbigeos y toUbigeos para calcular distancias
   for (const string& startCode : fromUbigeos) {
       // Encontrar el Ubigeo de inicio
       Ubigeo fromUbigeo = findUbigeoByCode(ubigeos, startCode);
       string fromName = fromUbigeo.name;
       for (const string& endCode : toUbigeos) {
           // Encontrar el Ubigeo de fin
           Ubigeo toUbigeo = findUbigeoByCode(ubigeos, endCode);
           string toName = toUbigeo.name;
           // Calcular la ruta más corta usando el algoritmo de colonia de abejas
           cout<<"Antes del artificialBeeColony"<<endl;
           vector<UbigeoResult> shortestPath = artificialBeeColony(ubigeos, startCode, endCode,numbees,maxtterations,
           maxattempts,elitistcount,abandonmentthreshold);
           cout<<"LUEGO del artificialBeeColony"<<endl;
           double distance = shortestPath.empty() ? 0.0 : shortestPath.back().distance;
           // Crear un objeto DistributionPlanDistance y agregarlo al resultado
           DistributionPlanDistance planDistance = {"inicio", startCode, fromName, endCode, toName, distance};
           result.push_back(planDistance);
           outFile <<i<<") "<< "Ruta de " << fromName << " a " << toName << ":" << endl;
           for (size_t i = 0; i < shortestPath.size() - 1; i++) {
               const UbigeoResult& from = shortestPath[i];
               const UbigeoResult& to = shortestPath[i + 1];
               double dist = calculateDistance(findUbigeoByCode(ubigeos, from.code), findUbigeoByCode(ubigeos, to.code));
               outFile << "De " << from.code << " (" << from.name << ") a " << to.code << " (" << to.name << ") hay "
               << dist << " km." << endl;
           }
           outFile << endl;
           // Imprimir 100 veces el carácter "-" seguido de un salto de línea
           for (int j = 0; j < 100; ++j) {
               outFile << "-";
           }
           outFile << endl;
           i++;
       }
   }
   return result;
```

Figura 20. Codificación de método general para buscar los planes de distribución

La Figura 20 muestra el código del método general para buscar planes de distribución. Este método itera entre cada par de almacén y destino, llamando a la función artificialBeeColony para encontrar la ruta entre ambos puntos. Luego, imprime el detalle de la ruta en un archivo de salida antes de continuar con el siguiente par.

8.3.6 Método principal para buscar los planes de rutas

Figura 21. Codificación de método principal para buscar los planes de rutas

```
vector<UbigeoResult> artificialBeeColony(const vector<Ubigeo>& ubigeos, const string& startCode, const string& endCode,
  const int numbees, const int maxtterations, const int maxattempts,
  const int elitistcount, const double abandonmentthreshold) {

    // Parámetros del algoritmo
    const int numBees = numbees; // Número de abejas
    const int maxIterations = maxtterations; // Número máximo de iteraciones
    const int maxAttempts = maxattempts; // Límite máximo de intentos para generar una ruta válida
    const int elitistCount = elitistcount; // Número de mejores soluciones para mantener
    const double abandonmentThreshold = abandonmentthreshold; // Umbral de abandono
    // Generador de números aleatorios
    random_device rd;
    mt19937 gen(rd());
    // Inicialización de la población de rutas
    vector<vector<string>> population(numBees);
    vector<double> fitness(numBees);
    vector<bool> validRoute(numBees, false); // Marcar si una abeja ha generado una ruta válida
    // Memoria de mejores soluciones
    vector<pair<vector<string>, double>> bestSolutions;
    // Inicialización de la población
    for (int i = 0; i < numBees; ++i) {
        cout<<"En la abeja "<<i+1<<endl;
        population[i] = generateRandomRoute(ubigeos, startCode, endCode, gen);
        if (population[i].empty()) {
            cerr << "Advertencia: No se pudo generar una ruta válida para la abeja " << i + 1
            << " después de " << maxAttempts << " intentos."<< " desde " << startCode << " a " << endCode << "." << endl;
            // Maneja el error asignando una ruta de otra abeja exitosa si está disponible
            if (!bestSolutions.empty()) {
                population[i] = bestSolutions.front().first; // Asigna la ruta de la mejor abeja
                fitness[i] = bestSolutions.front().second;
            } else {
                // Si no hay otra ruta exitosa, puedes decidir si reintentar o finalizar la ejecución
            }
        } else {
            // Si se genera una ruta válida, calcula su fitness
            fitness[i] = evaluateRoute(population[i], ubigeos);
        }
    }

    cout<<"Luego del for de generateRandomRoute"<<endl;
    // Iterar sobre las generaciones
    for (int iteration = 0; iteration < maxIterations; ++iteration) {
        cout<<"En la iteracion "<<iteration+1<<endl;
        // Modificación de rutas
        for (int i = 0; i < numBees; ++i) {
            // Modificar la ruta si supera el umbral de abandono
            if (fitness[i] > abandonmentThreshold) {
                modifyRoute(population[i], gen);
                fitness[i] = evaluateRoute(population[i], ubigeos);
            }
        }
```

La Figura 21 muestra el método principal del algoritmo de colonia de abejas para buscar planes de rutas, considerando parámetros como el número de abejas, iteraciones máximas, intentos para rutas válidas, mejores soluciones y umbral de abandono. Inicia con generateRandomRoute para crear la población y evalúa cada ruta con evaluateRoute. Luego, iterativamente busca mejorar las soluciones y retorna la de mayor valor objetivo.

8.3.7 Generación de rutas aleatorias

Figura 22. Codificación de generación de rutas aleatorias

```
vector<string> generateRandomRoute(const vector<Ubigeo>& ubigeos, const string& startCode, const string& endCode, mt19937& gen) {
    vector<string> route = {startCode};

    // Estructura de datos para el seguimiento de las rutas
    unordered_set<string> visitedUbigeos;
    visitedUbigeos.insert(startCode);
    // Parámetros de rango de búsqueda
    double initialDistance = maxDistanceBetweenPoints / 2; // Comienza con la mitad del rango máximo
    const size_t maxIterations = 2000;
    size_t iterationCount = 0;
    while (iterationCount < maxIterations) {
        iterationCount++;
        vector<size_t> nearbyIndices;
        // Obtiene el último punto en la ruta
        string lastCode = route.back();
        Ubigeo currentUbigeo = findUbigeoByCode(ubigeos, lastCode);
        // Filtra los puntos cercanos al último punto en la ruta
        for (size_t i = 0; i < ubigeos.size(); ++i) {
            Ubigeo ubigeo = ubigeos[i];

            if (visitedUbigeos.find(ubigeo.code) == visitedUbigeos.end()) {
                double distance = calculateDistance(currentUbigeo, ubigeo);

                if (distance <= initialDistance) {
                    nearbyIndices.push_back(i);
                }
            }
        }
        // Si no se encuentran índices cercanos, aumenta el rango de búsqueda
        if (nearbyIndices.empty()) {
            initialDistance += maxDistanceBetweenPoints / 4; // Aumenta el rango de búsqueda de forma gradual
            continue;
        }
        // Selecciona un índice aleatorio de los puntos cercanos
        uniform_int_distribution<size_t> distIndex(0, nearbyIndices.size() - 1);
        size_t index = nearbyIndices[distIndex(gen)];
        string ubigeoCode = ubigeos[index].code;
        // Si el punto seleccionado es el punto final, agrega el punto final a la ruta y termina
        if (ubigeoCode == endCode) {
            route.push_back(ubigeoCode);
            return route;
        }
        // Agrega el punto seleccionado a la ruta
        route.push_back(ubigeoCode);
        visitedUbigeos.insert(ubigeoCode);
    }
    // Si no se pudo encontrar una ruta válida, devuelve una ruta vacía
    return {};
}
```

La Figura 22 muestra la codificación de la función encargada de la generación de rutas aleatorias. En esta función lo que se realiza es en primer lugar una búsqueda de los puntos cercanos al último punto en la ruta que se va formando, para lo cual en primera instancia se toma que la máxima distancia entre los puntos que forman un plan de rutas no debe exceder de 80 kilómetros entre ellos. Luego se selecciona un punto aleatorio de los puntos filtrados como posibles siguientes puntos en la ruta. Si este punto es el punto destino se deja de iterar, caso contrario se sigue iterando para buscar nuevos puntos en la ruta.

8.3.8 Función de evaluación de rutas generadas

Figura 23. Codificación de función de evaluación de rutas generadas

```
double evaluateRoute(const vector<string>& route, const vector<Ubigeo>& ubigeos) {
    double totalDistance = 0.0;
    //Esto es para que se obtenga rutas de minimo 4 nodos
    if(route.size() < 4) return 999999999;

    for (size_t i = 0; i < route.size() - 1; ++i) {
        Ubigeo currentUbigeo = findUbigeoByCode(ubigeos, route[i]);
        Ubigeo nextUbigeo = findUbigeoByCode(ubigeos, route[i + 1]);
        totalDistance += calculateDistance(currentUbigeo, nextUbigeo);
    }
    return totalDistance;
}
```

La Figura 23 muestra la función de evaluación de rutas generadas, enfocada en calcular la cantidad total de kilómetros recorridos en una ruta. Esta evaluación ayuda a seleccionar soluciones con menor distancia, alineándose con la segunda función objetivo del problema.

8.3.9 Modificación de planes de rutas generados

Figura 24. Codificación de función de modificación de planes de rutas generados

```
void modifyRoute(vector<string>& route, mt19937& gen) {
    if (route.size() < 3) return;

    vector<size_t> indices(route.size() - 2);
    iota(indices.begin(), indices.end(), 1);
    shuffle(indices.begin(), indices.end(), gen);

    int index1;
    int index2;
    // Si solo hay un elemento en indices
    if (indices.size() == 1) {
        index1 = indices[0];
        index2 = index1;

        // Genera un índice aleatorio dentro del rango permitido hasta obtener uno distinto de index1
        while (index2 == index1) {
            index2 = rand() % route.size();
        }

    } else {
        // Si indices tiene más de un elemento, utiliza los dos primeros elementos
        index1 = indices[0];
        index2 = indices[1];
    }

    // Verificar que los índices seleccionados son distintos
    if (index1 != index2 && index1 < route.size() && index2 < route.size()) {
        swap(route[index1], route[index2]);
    }
}
```

En la Figura 24 se puede observar la función encargada de modificación de planes de rutas generados. Esta función nos permite explorar nuevas versiones posibles generadas en base a la modificación de soluciones previas.

8.3.10 Validaciones realizadas

Se llevaron a cabo múltiples pruebas en poblaciones pequeñas para analizar el desempeño de cada uno de los operadores principales del algoritmo, así como la correcta evolución y validación de las soluciones. En el Anexo 11 se incluye una prueba detallada de la codificación del algoritmo.

8.4 Discusión

En este capítulo se logró el resultado esperado R5, correspondiente a la codificación del algoritmo de colonia de abejas adaptado para resolver el problema de distribución de ayuda, tomando como base el diseño previo del pseudocódigo y las estructuras definidas.

El logro de estos resultados implica disponer de un algoritmo de colonia de abejas desarrollado y funcional. Asimismo, al tener el código implementado y al observar las situaciones surgidas durante su ejecución, se puede comprender con mayor profundidad el funcionamiento de este algoritmo bioinspirado y cómo lleva a cabo la búsqueda de posibles soluciones.

Este progreso representa un paso significativo en la obtención de los resultados del proyecto y destaca la relevancia de contar con un diseño previo, ya que simplifica notablemente la implementación del código.

9. Codificación del algoritmo genético

9.1 Introducción

Este capítulo presenta el resultado 6, enfocado en el desarrollo del código del algoritmo genético para la distribución de ayuda, siguiendo el pseudocódigo y las estructuras de datos diseñadas previamente. Se explicarán los métodos principales y aspectos técnicos relevantes. La URL para acceder al código se encuentra en el Anexo 10.

9.2 Resultados alcanzados

Este apartado describe en detalle las acciones llevadas a cabo para alcanzar el resultado esperado R6: "Codificación del algoritmo genético adaptado al problema de distribución de ayuda.".

9.3 Codificación del algoritmo de genético adaptado al problema de distribución de ayuda

Al igual que en la etapa de diseño, la presentación del código del algoritmo genético se organiza en función de sus métodos principales. Esto se hace con el propósito de poder detallar y explicar lo que se ha realizado en cada uno de ellos.

9.3.1 Método principal para búsqueda genética de planes de distribución

Figura 25. Codificación de método principal para búsqueda genética de planes de distribución

```
vector<ResourceDistribution> findBestResourceDistributionGenetic(int populationSize, int geneLength, int generations,const vector<Stock>& stocks,
const vector<Resource>& resources, const vector<string>& fromUbigeos,const vector<string>& toUbigeos,const vector<Ubigeo>& ubigeos) {

    srand(time(0));
    map<string, double> requestedAmountMap;
    for (const auto& stock : stocks) {
        string resource = stock.resource; double requestedAmount = 0.0;
        for (const auto& resourceData : resources) {
            if (resourceData.item == resource) requestedAmount += resourceData.quantity;
        }
        requestedAmountMap[resource] = requestedAmount;
    }
    vector<Chromosome> population;
    for (int i = 0; i < populationSize; ++i) population.push_back(generateRandomChromosome(resources, stocks, fromUbigeos, toUbigeos, ubigeos));
    for (auto& chromosome : population) {
        chromosome.fitness = evaluateFitness(chromosome, stocks, resources, fromUbigeos, toUbigeos, ubigeos, requestedAmountMap);
    }
    for (int generation = 0; generation < generations; ++generation) {
        vector<Chromosome> newPopulation;
        for (int i = 0; i < populationSize / 2; ++i) {
            Chromosome parent1 = tournamentSelection(population); Chromosome parent2 = tournamentSelection(population);
            pair<Chromosome, Chromosome> offspringPair = crossover(parent1, parent2);
            Chromosome& offspring1 = offspringPair.first; Chromosome& offspring2 = offspringPair.second;
            mutate(offspring1); mutate(offspring2);
            // Verificar y corregir los cromosomas después del cruce y la mutación
            verifyAndCorrectChromosome(offspring1, stocks); verifyAndCorrectChromosome(offspring2, stocks);
            offspring1.fitness = evaluateFitness(offspring1, stocks, resources, fromUbigeos, toUbigeos, ubigeos, requestedAmountMap);
            offspring2.fitness = evaluateFitness(offspring2, stocks, resources, fromUbigeos, toUbigeos, ubigeos, requestedAmountMap);
            newPopulation.push_back(offspring1);newPopulation.push_back(offspring2);
        }
        population = newPopulation;
    }
    Chromosome best = *max_element(population.begin(), population.end(), [](const Chromosome& a, const Chromosome& b) {
        return a.fitness < b.fitness;
    });
    return best.genes;
}
```

La función findBestResourceDistributionGenetic implementa un algoritmo genético para optimizar la distribución de recursos. Recibe parámetros como el tamaño de la población, la longitud de los genes, el número de generaciones, y listas de stocks y ubicaciones. Primero, calcula las cantidades requeridas de cada recurso y crea una población inicial de cromosomas aleatorios, evaluando su aptitud con evaluateFitness. Luego, durante el número especificado de generaciones, genera una nueva población mediante selección, cruce (crossover), y mutación. La selección se realiza con tournamentSelection, y la validez de los cromosomas mutados se asegura con verifyAndCorrectChromosome. Al final, se elige el mejor cromosoma de la última población, que representa la distribución óptima de recursos, y se retorna su configuración.

9.3.2 Generación de cromosomas aleatorios

Figura 26. Codificación de generación de cromosomas aleatorios

```
Chromosome generateRandomChromosome(const vector<Resource>& resources, const vector<Stock>& stocks,const vector<string>& fromUbigeos,
        const vector<string>& toUbigeos, const vector<Ubigeo>& ubigeos) {
    Chromosome chromosome(0); // Inicializa el cromosoma con tamaño 0
    // Calcular la demanda total de cada ubicación de destino para cada recurso
    unordered_map<string, double> totalDemandsByStock; // Stock -> Quantity
    unordered_map<string, unordered_map<string, double>> totalDemandsByStockToUbigeo; // Stock -> (Ubigeo -> Quantity)
    for (const auto& stock : stocks) {
        totalDemandsByStock[stock.resource] = 0.0;
        for (const auto& resource : resources) {
            for (const auto& toCode : toUbigeos) {
                if (stock.resource == resource.item && resource.code == toCode) {
                    totalDemandsByStock[stock.resource] += resource.quantity;
                    totalDemandsByStockToUbigeo[stock.resource][toCode] += resource.quantity;
                }
            }
        }
    }
    for (const auto& stock : stocks) {
        double total = 0.0;
        for (const auto& toCode : toUbigeos) {
            for (const auto& resource : resources) {
                if (resource.code == toCode && stock.resource == resource.item) total += resource.quantity;
            }
        }
        if (total <= stock.quantityTotal) {
            for (const auto& toCode : toUbigeos) {
                Ubigeo toUbigeo = findUbigeoByCode(ubigeos, toCode);
                if (!toUbigeo.code.empty()) {
                    double pedido = 0.0;
                    for (const auto& resource : resources) {
                        if (resource.code == toCode && stock.resource == resource.item) pedido += resource.quantity;
                    }
                    for (auto it = fromUbigeos.begin(); it != fromUbigeos.end(); ++it) {
                        const auto& fromCode = *it;
                        Ubigeo fromUbigeo = findUbigeoByCode(ubigeos, fromCode);
                        if (!fromUbigeo.code.empty()) {
                            bool isLastElement = next(it) == fromUbigeos.end();
                            double assigned;
```

La función generateRandomChromosome genera un cromosoma aleatorio que representa una posible distribución de recursos basada en las demandas y stocks disponibles. Esta función toma como parámetros las listas de recursos, stocks, ubicaciones de origen y destino, y ubigeos. Primero, calcula las demandas totales de cada recurso por ubicación de destino. Luego, distribuye el stock según la demanda: si el stock es suficiente, asigna la cantidad solicitada aleatoriamente; si es insuficiente, realiza una distribución proporcional y ajustada. Cada asignación se guarda en una instancia de ResourceDistribution y se añade al cromosoma, que finalmente es retornado como una lista balanceada de asignaciones de recursos.

9.3.3 Función de evaluación de aptitud del algoritmo genético

Figura 27. Codificación de función de evaluación de aptitud del algoritmo genético

```
double evaluateFitness(const Chromosome& chromosome, const vector<Stock>& stocks,const vector<Resource>& resources,
const vector<string>& fromUbigeos,const vector<string>& toUbigeos, const vector<Ubigeo>& ubigeos,const map<string, double>& requestedAmountMap) {
    map<string, map<string, pair<double, double>>> distributionData;
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        map<string, pair<double, double>> resourceData;
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            double demand = getDemand(toUbigeos[j], resources, resource);
            double delivered = 0.0;
            for (const auto& dist : chromosome.genes) {
                if (dist.resource == resource && dist.toCode == toUbigeos[j]) delivered += dist.quantity;
            }
            double distributionPercentage = delivered / demand;
            resourceData[toUbigeos[j]] = make_pair(distributionPercentage, delivered);
        }
        distributionData[resource] = resourceData;
    }
    double totalWeightedSum = 0.0;
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            double delivery = distributionData[resource][toUbigeos[j]].second;
            double distributionPercentage = distributionData[resource][toUbigeos[j]].first;
            totalWeightedSum += delivery * distributionPercentage;
        }
    }
    map<string, pair<double, int>> urgencyMap;
    for (const auto& stock : stocks) {
        string resource = stock.resource;
        for (size_t j = 0; j < toUbigeos.size(); ++j) {
            bool requested = false;
            for (const auto& res : resources) {
                if (res.item == resource && res.code == toUbigeos[j]) {
                    requested = true;
                    break;
                }
            }
```

La función evaluateFitness calcula la aptitud de un cromosoma que representa la distribución de recursos, midiendo la satisfacción de la demanda en cada destino. Usa una suma ponderada de las cantidades distribuidas y prioridades, junto con métricas de cobertura de stock y urgencia, generando un valor que refleja la efectividad de la distribución.

9.3.4 Selección por torneo de planes de distribución

Figura 28. Codificación de selección por torneo de planes de distribución

```
Chromosome tournamentSelection(const vector<Chromosome>& population) {
    int tournamentSize = 3;
    Chromosome best = population[rand() % population.size()];
    for (int i = 1; i < tournamentSize; ++i) {
        Chromosome contender = population[rand() % population.size()];
        if (contender.fitness > best.fitness) {
            best = contender;
        }
    }
    return best;
}
```

La función tournamentSelection selecciona un cromosoma de una población mediante el método de torneo, eligiendo el de mayor aptitud entre tres cromosomas seleccionados aleatoriamente. Esto aumenta

la probabilidad de seleccionar soluciones de alta calidad. La función retorna el cromosoma con mejor aptitud.

9.3.5 Cruzamiento de cromosomas de planes de distribución

Figura 29. Codificación de cruzamiento de cromosomas de planes de distribución

```
pair<Chromosome, Chromosome> crossover(const Chromosome& parent1, const Chromosome& parent2) {
    int geneLength = parent1.genes.size();
    Chromosome offspring1(geneLength), offspring2(geneLength);

    int crossoverPoint = rand() % geneLength;
    for (int i = 0; i < geneLength; ++i) {
        if (i < crossoverPoint) {
            offspring1.genes[i] = parent1.genes[i];
            offspring2.genes[i] = parent2.genes[i];
        } else {
            offspring1.genes[i] = parent2.genes[i];
            offspring2.genes[i] = parent1.genes[i];
        }
    }
    return {offspring1, offspring2};
}
```

La función crossover combina dos cromosomas parentales para generar dos descendientes. Utiliza un punto de cruce aleatorio, copiando genes de los padres y permitiendo que los hijos hereden características de ambos. La función devuelve el par de hijos generados.

9.3.6 Mutación de cromosomas de distribución

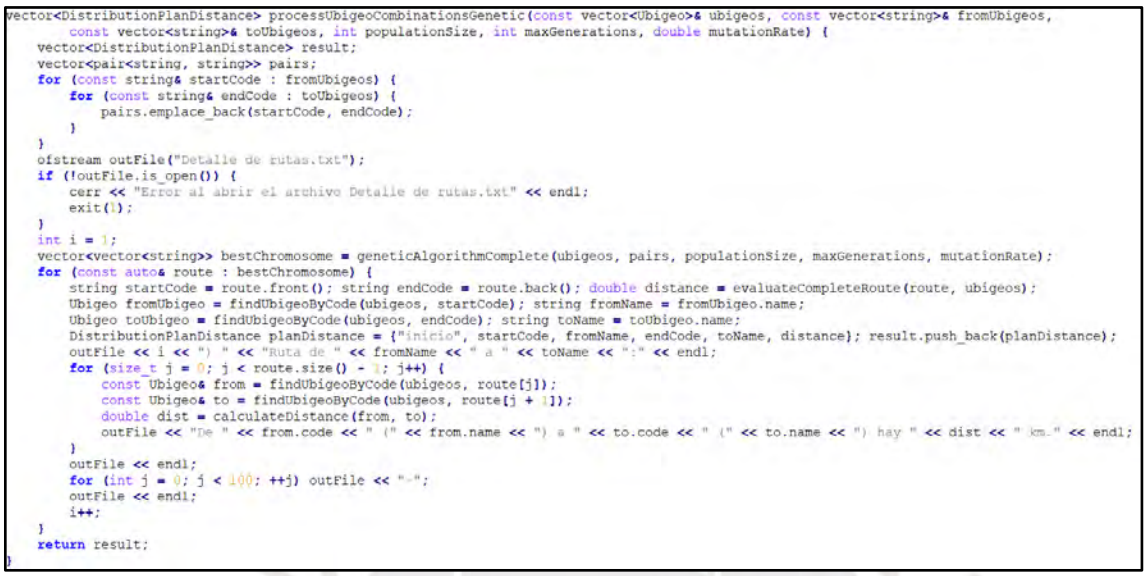
Figura 30. Codificación de mutación de cromosomas de distribución

```
void mutate(Chromosome& chromosome) {
    double mutationRate = 0.01;
    for (auto& gene : chromosome.genes) {
        if ((double)rand() / RAND_MAX < mutationRate) {
            gene.quantity = static_cast<double>(rand() % 100 + 1);
        }
    }
}
```

La función mutate introduce variaciones aleatorias en los cromosomas de un algoritmo genético, aplicando una mutación con un 1% de probabilidad por gen. Esto ayuda a evitar el estancamiento en óptimos locales y facilita una búsqueda más exhaustiva de soluciones óptimas.

9.3.7 Método general para búsqueda genética de rutas

Figura 31. Codificación de método general para búsqueda genética de rutas



```
vector<DistributionPlanDistance> processUbigeoCombinationsGenetic(const vector<Ubigeo>& ubigeos, const vector<string>& fromUbigeos,
        const vector<string>& toUbigeos, int populationSize, int maxGenerations, double mutationRate) {
    vector<DistributionPlanDistance> result;
    vector<pair<string, string>> pairs;
    for (const string& startCode : fromUbigeos) {
        for (const string& endCode : toUbigeos) {
            pairs.emplace_back(startCode, endCode);
        }
    }
    ofstream outFile("Detalle de rutas.txt");
    if (!outFile.is_open()) {
        cerr << "Error al abrir el archivo Detalle de rutas.txt" << endl;
        exit(1);
    }
    int i = 1;
    vector<vector<string>> bestChromosome = geneticAlgorithmComplete(ubigeos, pairs, populationSize, maxGenerations, mutationRate);
    for (const auto& route : bestChromosome) {
        string startCode = route.front(); string endCode = route.back(); double distance = evaluateCompleteRoute(route, ubigeos);
        Ubigeo fromUbigeo = findUbigeoByCode(ubigeos, startCode); string fromName = fromUbigeo.name;
        Ubigeo toUbigeo = findUbigeoByCode(ubigeos, endCode); string toName = toUbigeo.name;
        DistributionPlanDistance planDistance = {"inicio", startCode, fromName, endCode, toName, distance}; result.push_back(planDistance);
        outFile << i << ") " << "Ruta de " << fromName << " a " << toName << ":" << endl;
        for (size_t j = 0; j < route.size() - 1; j++) {
            const Ubigeo& from = findUbigeoByCode(ubigeos, route[j]);
            const Ubigeo& to = findUbigeoByCode(ubigeos, route[j + 1]);
            double dist = calculateDistance(from, to);
            outFile << "De " << from.code << " (" << from.name << ") a " << to.code << " (" << to.name << ") hay " << dist << " km." << endl;
        }
        outFile << endl;
        for (int j = 0; j < 100; ++j) outFile << "-";
        outFile << endl;
        i++;
    }
    return result;
}
```

Este código busca las mejores rutas de distribución de recursos entre varios puntos de inicio y destino mediante un algoritmo genético. Primero, genera todas las combinaciones posibles de pares inicio-destino y las almacena en pairs. Luego, ejecuta el algoritmo genético con geneticAlgorithmComplete para encontrar el mejor cromosoma, que representa las rutas óptimas. Cada ruta en el mejor cromosoma se evalúa usando evaluateCompleteRoute, calculando la distancia total, y se crea un objeto DistributionPlanDistance con los detalles, almacenándolo en el vector result. Finalmente, se registra en un archivo la información de cada ruta, incluyendo distancias, y se devuelve result con las rutas óptimas encontradas.

9.3.8 Método principal para búsqueda genética de rutas

Figura 32. Codificación de método principal para búsqueda genética de rutas

```
vector<vector<string>> geneticAlgorithmComplete(const vector<Ubigeo>& ubigeos, const vector<pair<string, string>>& pairs,
                                                int populationSize, int maxGenerations, double mutationRate) {
    random_device rd; mt19937 gen(rd());
    vector<vector<vector<string>>> population = initializePopulation(ubigeos, pairs, populationSize, gen);
    vector<double> fitness(populationSize);
    for (int generation = 0; generation < maxGenerations; ++generation) {
        for (int i = 0; i < populationSize; ++i) {
            fitness[i] = 0.0;
            for (const auto& route : population[i]) {
                fitness[i] += evaluateCompleteRoute(route, ubigeos);
            }
        }
        vector<size_t> indices(populationSize);
        iota(indices.begin(), indices.end(), 0);
        sort(indices.begin(), indices.end(), [&fitness](size_t a, size_t b) { return fitness[a] < fitness[b]; });
        vector<vector<vector<string>>> newPopulation;
        uniform_real_distribution<double> mutationDist(0.0, 1.0);
        for (size_t i = 0; i < populationSize; i += 2) {
            const auto& parent1 = population[indices[i]];
            const auto& parent2 = population[indices[i + 1]];
            pair<vector<vector<string>>, vector<vector<string>>> children = crossoverChromosomes(parent1, parent2, gen);
            vector<vector<string>>& child1 = children.first;
            vector<vector<string>>& child2 = children.second;
            for (auto& route : child1) {
                if (mutationDist(gen) < mutationRate) mutateCompleteRoute(route, gen);
            }
            for (auto& route : child2) {
                if (mutationDist(gen) < mutationRate) mutateCompleteRoute(route, gen);
            }
            newPopulation.push_back(child1);
            newPopulation.push_back(child2);
        }
        population = newPopulation;
    }
    return population.front();  // Retorna el mejor cromosoma encontrado
}
```

La función geneticAlgorithmComplete aplica un algoritmo genético para encontrar las mejores rutas de distribución entre múltiples pares de ubicaciones. Primero, inicializa una población de cromosomas, donde cada cromosoma representa un conjunto de rutas. Cada cromosoma es evaluado por su aptitud, calculando la distancia total de todas las rutas que contiene. Luego, se seleccionan los mejores cromosomas para reproducirse, aplicando un cruce para generar descendientes que heredan rutas de ambos padres. Además, se introduce la mutación en las rutas de los descendientes con cierta probabilidad para mantener la diversidad genética. Este proceso de selección, cruce y mutación se repite durante un número especificado de generaciones. Al final del proceso, la función devuelve el cromosoma con la mejor aptitud, que contiene las rutas óptimas encontradas.

9.3.9 Inicialización de población de rutas

Figura 33. Codificación de inicialización de población de rutas

```
vector<vector<vector<string>>> initializePopulation(const vector<Ubigeo>& ubigeos, const vector<pair<string, string>>& pairs,
                    int populationSize, mt19937& gen) {
    vector<vector<vector<string>>> population(populationSize, vector<vector<string>>(pairs.size()));
    for (int i = 0; i < populationSize; ++i) {
        for (size_t j = 0; j < pairs.size(); ++j) {
            const auto& pair = pairs[j];
            vector<string> route = generateCompleteRoute(ubigeos, pair.first, pair.second, gen);
            if (!route.empty()) {
                population[i][j] = route;
            } else {
                population[i][j] = {pair.first, pair.second}; // Ruta mínima si no se puede generar una válida
            }
        }
    }
    return population;
}
```

La función initializePopulation crea la población inicial de cromosomas para el algoritmo genético, generando combinaciones de rutas entre ubicaciones. Si no hay una ruta válida, asigna una ruta directa mínima, asegurando diversidad en la población inicial.

9.3.10 Generación de posibles rutas

Figura 34. Codificación de generación de posibles rutas

```
vector<string> generateCompleteRoute(const vector<Ubigeo>& ubigeos, const string& startCode, const string& endCode, mt19937& gen) {
    vector<string> route = {startCode};
    unordered_set<string> visitedUbigeos;
    visitedUbigeos.insert(startCode);
    double initialDistance = maxDistanceBetweenPoints / 2;  const size_t maxIterations = 2000; size_t iterationCount = 0;
    while (iterationCount < maxIterations) {
        iterationCount++;
        vector<size_t> nearbyIndices;
        string lastCode = route.back();
        Ubigeo currentUbigeo = findUbigeoByCode(ubigeos, lastCode);
        for (size_t i = 0; i < ubigeos.size(); ++i) {
            Ubigeo ubigeo = ubigeos[i];
            if (visitedUbigeos.find(ubigeo.code) == visitedUbigeos.end()) {
                double distance = calculateDistance(currentUbigeo, ubigeo);
                if (distance <= initialDistance) {
                    nearbyIndices.push_back(i);
                }
            }
        }
        if (nearbyIndices.empty()) {
            initialDistance += maxDistanceBetweenPoints / 4;
            continue;
        }
        uniform_int_distribution<size_t> distIndex(0, nearbyIndices.size() - 1);
        size_t index = nearbyIndices[distIndex(gen)];
        string ubigeoCode = ubigeos[index].code;
        if (ubigeoCode == endCode) {
            route.push_back(ubigeoCode);
            return route;
        }
        route.push_back(ubigeoCode);
        visitedUbigeos.insert(ubigeoCode);
    }
    cout << "No se pudo generar una ruta válida de " << startCode << " a " << endCode << " en " << maxIterations << " iteraciones." << endl;
    return {};
}
```

La función generateCompleteRoute crea una ruta completa entre dos ubicaciones, usando un generador aleatorio para seleccionar ubicaciones intermedias. Si no se encuentra una ruta adecuada tras un número máximo de iteraciones (maxIterations), retorna una ruta vacía. Esta función es clave para generar rutas iniciales en el algoritmo genético.

9.3.11 Evaluación de aptitud de rutas en algoritmo genético

Figura 35. Codificación de evaluación de aptitud de rutas en algoritmo genético

```
double evaluateCompleteRoute(const vector<string>& route, const vector<Ubigeo>& ubigeos) {
    double totalDistance = 0.0;
    if (route.size() < 2) return numeric_limits<double>::max();

    for (size_t i = 0; i < route.size() - 1; ++i) {
        Ubigeo currentUbigeo = findUbigeoByCode(ubigeos, route[i]);
        Ubigeo nextUbigeo = findUbigeoByCode(ubigeos, route[i + 1]);
        totalDistance += calculateDistance(currentUbigeo, nextUbigeo);
    }
    return totalDistance;
}
```

La función evaluateCompleteRoute calcula la distancia total de una ruta, acumulando las distancias entre ubicaciones consecutivas. Devuelve una distancia máxima si la ruta es inválida y usa esta distancia total para evaluar la eficiencia en el algoritmo genético.

9.3.12 Cruzamiento de rutas del algoritmo genético

Figura 36. Codificación de cruzamiento de rutas del algoritmo genético

```
pair<vector<vector<string>>, vector<vector<string>>> crossoverChromosomes(const vector<vector<string>>& parent1,
            const vector<vector<string>>& parent2, mt19937& gen) {
    size_t numRoutes = parent1.size();
    vector<vector<string>> child1 = parent1;
    vector<vector<string>> child2 = parent2;

    uniform_int_distribution<size_t> distRoute(0, numRoutes - 1);
    size_t crossoverRouteIndex = distRoute(gen);

    swap(child1[crossoverRouteIndex], child2[crossoverRouteIndex]);

    return {child1, child2};
}
```

La función crossoverChromosomes realiza un cruce entre dos cromosomas, cada uno representado por un conjunto de rutas. Copia los cromosomas de los padres a los hijos y selecciona aleatoriamente una ruta para intercambiar entre ambos cromosomas hijos. Esto introduce variabilidad genética en la población del algoritmo genético. Finalmente, devuelve el par de cromosomas hijos generados mediante el cruce.

9.3.13 Mutación de rutas del algoritmo genético

Figura 37. Codificación de mutación de rutas del algoritmo genético

```
void mutateCompleteRoute(vector<string>& route, mt19937& gen) {
    if (route.size() < 3) return;
    uniform_int_distribution<size_t> distIndex(1, route.size() - 2);
    size_t index1 = distIndex(gen);
    size_t index2 = distIndex(gen);
    if (index1 != index2) {
        swap(route[index1], route[index2]);
    }
}
```

La función mutateCompleteRoute introduce una mutación en una ruta representada por un vector de cadenas, para mantener la diversidad en el algoritmo genético. Si la ruta tiene al menos tres elementos, selecciona dos índices al azar (excluyendo inicio y fin) y los intercambia. Este cambio aleatorio permite explorar nuevas configuraciones en busca de mejores soluciones.

9.3.14 Validaciones realizadas

Se llevaron a cabo varias pruebas en poblaciones reducidas para analizar el desempeño de los operadores principales del algoritmo, además de verificar la correcta evolución y validación de las soluciones. En el Anexo 12 se detalla una de las pruebas realizadas a la codificación del algoritmo.

9.4 Discusión

En este capítulo se logró el resultado esperado R6, correspondiente a la codificación del algoritmo genético adaptado para abordar el problema de distribución de ayuda, siguiendo el diseño previamente establecido que incluye tanto el pseudocódigo como las estructuras de datos.

Al alcanzar este resultado, se dispone de un algoritmo genético completamente desarrollado y funcional. Además, contar con el código implementado y analizar las situaciones que surgen durante su ejecución proporciona una comprensión más profunda sobre el funcionamiento de este algoritmo inspirado en la naturaleza y su proceso de búsqueda de soluciones óptimas.

Este avance constituye un paso fundamental para obtener los resultados del proyecto y subraya la importancia de un diseño previo, ya que facilita de manera significativa la implementación del código

10. Interfaz gráfica

10.1 Introducción

En este capítulo se desarrolla el resultado 7, centrado en el diseño y creación de una interfaz gráfica destinada a la ejecución del algoritmo de colonia de abejas para la distribución de ayuda. La interfaz permite ingresar parámetros de ejecución y genera archivos de texto con los planes de distribución y rutas. En el anexo 10 se incluye la URL del código de la interfaz.

10.2 Resultados alcanzados

Este apartado describe en detalle las acciones llevadas a cabo para lograr el resultado esperado R7: "Interfaz gráfica diseñada para ejecutar el algoritmo basado en la colonia de abeja aplicado al problema de distribución de ayuda".

10.3 Interfaz gráfica para la ejecución de los algoritmos

El contenido de esta sección explicará cada sección de la pantalla del programa para la carga de datos, la selección de puntos de destino a entregar recursos y el ingreso de los parámetros.

10.3.1 Pantalla general de la interfaz

Figura 38. Interfaz para la ejecución de los algoritmos

Se muestra la pantalla general de la interfaz, en esta se tienen una serie de funcionalidades que a continuación se explicarán de forma más detallada por sección.

10.3.2 Selección de algoritmo

Figura 39. Selección del algoritmo

Se muestra la sección de la interfaz correspondiente a la selección del algoritmo que se desea ejecutar. El algoritmo por defecto es el algoritmo de abejas, en caso se escoja el algoritmo genético los parámetros que se le solicitará al usuario serán distintos, dado que cada tipo de algoritmo requiere de parámetros distintos.

10.3.3 Carga de datos

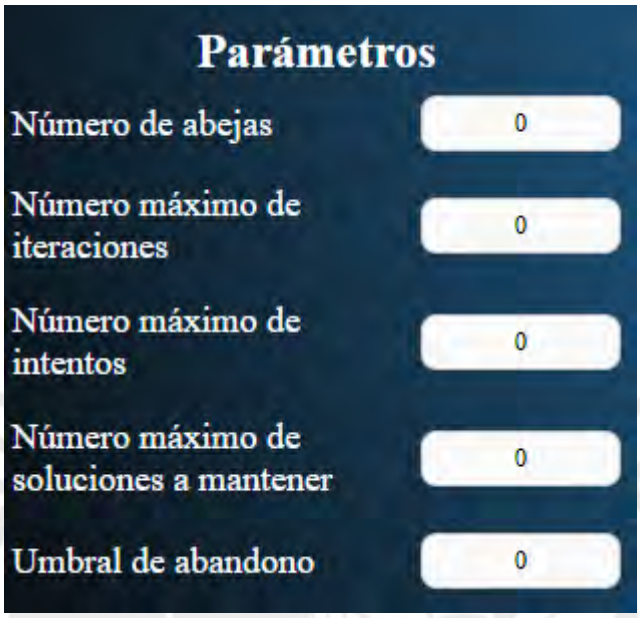
Figura 40. Carga de datos para algoritmos

La interfaz incluye una sección para cargar datos necesarios para los algoritmos. El primer archivo contiene los stocks generales y por almacén; el segundo, las demandas de recursos en las áreas que requieren ayuda; y el tercero, información general de lugares mapeados en Perú, como distrito,

provincia, coordenadas, nivel de urgencia y población. Los archivos deben cargarse en formato CSV, obtenido mediante una conversión sencilla desde Excel.

10.3.4 Ingreso de variables del algoritmo de colonia de abejas

Figura 41. Ingreso de variables del algoritmo de colonia de abejas



Se muestra la sección de la interfaz correspondiente al ingreso de variables del algoritmo de colonia de abejas. En este se ingresa la cantidad de abejas, el número máximo de iteraciones, el máximo de intentos, el máximo de soluciones a mantener y el umbral de abandono.

10.3.5 Ingreso de variables del algoritmo genético

Figura 42. Ingreso de variables del algoritmo genético

Se muestra la sección de la interfaz correspondiente al ingreso de variables del algoritmo genético. En este se ingresa el tamaño de la población, la longitud del gen, el número de iteraciones y la tasa de mutación que se emplearán para la búsqueda de soluciones..

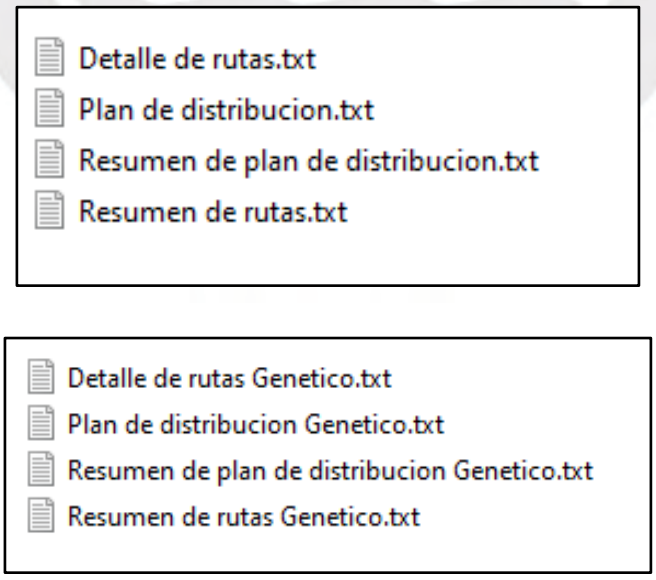
10.3.6 Información de almacenes y selección de destinos

Figura 43. Información de almacenes y selección de destinos

Se muestra la sección de la interfaz correspondiente a la información de almacenes y selección de destinos. Los datos de almacenes iniciales se podrán observar luego de ingresar el archivo de almacenes y de información general. Para escoger los lugares a los que se van a distribuir los recursos se tiene un combo box donde se observan los datos de los lugares ingresados en el archivo demanda. Luego de seleccionar algunos puntos de destino en la parte derecha podremos observar lugares seleccionados.

10.3.7 Resultados ejecución

Figura 44. Resultados ejecución



Al ejecutar la interfaz después de cargar los archivos CSV y seleccionar los destinos, el algoritmo genera cuatro archivos .txt: dos con el plan de distribución y dos con el plan de rutas para la entrega de recursos. Si se usó el algoritmo genético, los archivos incluyen "Genético" en el nombre; de lo contrario, este texto no aparece.

10.3.8 Validaciones realizadas

Se llevaron a cabo pruebas de integración para asegurar que los datos ingresados a través de la interfaz se utilizan correctamente en la ejecución de los algoritmos, y que los resultados presentados sean coherentes con la mejor solución proporcionada por el algoritmo específico. El Anexo 13 detalla las pruebas de integración realizadas para cada caso.

10.4 Discusión

En este capítulo se logró el resultado esperado R7, enfocado en el desarrollo de una interfaz gráfica diseñada para ejecutar el algoritmo basado en la colonia de abejas aplicado al problema de distribución de ayuda. Asimismo, esta interfaz permite la ejecución del algoritmo genético, facilitando la comparación de sus resultados con los del algoritmo de colonia de abejas.

La interfaz gráfica ofrece a los usuarios la posibilidad de configurar los algoritmos y completar el proceso de planificación para la distribución de recursos, obteniendo un resultado final basado en los datos proporcionados.

11. Calibración de variables

11.1 Introducción

Este capítulo se centra en el resultado 8, orientado a la calibración de variables de los algoritmos de colonia de abejas y genético. Su propósito es determinar los valores óptimos de los parámetros que permitan maximizar tanto el rendimiento como la calidad de las soluciones.

11.2 Resultados alcanzados

Este apartado describe en detalle las acciones realizadas para lograr el resultado esperado R8: "Realizar la calibración de las variables que se vayan a utilizar en los algoritmos de colonia de abejas y genético".

11.3 Calibración de variables para el algoritmo de abejas

A continuación, se presentan las pruebas llevadas a cabo para ajustar las variables de entrada requeridas por el algoritmo de colonia de abejas durante su ejecución.

11.3.1 Número de abejas

La calibración de este parámetro comenzó con un valor inicial de 50. Posteriormente, se ajustó este valor incrementándolo y disminuyéndolo, obteniendo los resultados que se muestran en la tabla siguiente.:

Tabla 11. Calibración de número de abejas

numBees	Mejor Fitness obtenido de repartición de recursos	Mejor Fitness obtenido de rutas para entregas	Tiempo de ejecución
10	2,008,257	82,708	2 minutos y 25 segundos
25	2,008,170	81,435	5 minutos y 58 segundos
50	2,008,170	54,573	11 minutos y 34 segundos

75	2,008,165	54,860	16 minutos y 2 segundos
100	2,008,124	51,110	22 minutos y 30 segundos

Para la repartición de recursos, los valores óptimos de fitness se obtienen con un numBees de 10, 50, y 100. En términos de rutas, los valores menores de fitness, que indican menos kilómetros recorridos, se logran con numBees de 50 y 75. La mejor opción es numBees igual a 50, ya que ofrece buenos resultados en un tiempo razonable en comparación con el valor de 100, que toma casi el doble de tiempo.

11.3.2 Número de iteraciones de búsqueda

Para el número de iteraciones, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 100, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 12. Calibración de número de iteraciones de búsqueda

maxIterations	**Mejor Fitness obtenido de repartición de recursos**	**Mejor Fitness obtenido de rutas para entregas**	**Tiempo de ejecución**
25	2,008,130	54,615	4 minutos y 56 segundos
50	2,008,140	54,583	7 minutos y 0 segundos
100	2,008,170	54,573	11 minutos y 58 segundos
125	2,008,170	54,573	13 minutos y 24 segundos
150	2,008,170	54,573	16 minutos y 21 segundos

Se observa que en los últimos 3 casos hay un empate en ambos valores de fitness, tanto de repartición de recursos como de rutas, así que se optará por el que toma un menor tiempo de ejecución que es la opción con un número de iteraciones de 100.

11.3.3 Límite máximo de intentos para generar una ruta válida

Para el límite máximo de intentos, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 10, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 13. Calibración de límite máximo de intentos

maxIterations	Mejor Fitness obtenido de repartición de recursos	Mejor Fitness obtenido de rutas para entregas	Tiempo de ejecución
4	2,008,070	54,573	11 minutos y 45 segundos
7	2,008,090	54,563	11 minutos y 25 segundos
10	2,008,170	54,554	10 minutos y 45 segundos
13	2,008,140	54,555	11 minutos y 40 segundos
15	2,008,170	54,554	11 minutos y 47 segundos

Los mejores valores de fitness para la repartición de recursos y rutas se obtienen con maxIterations de 10 y 15, ya que resultan en menores distancias recorridas. Dado que el valor de 10 reduce el tiempo de ejecución en comparación con 15, la opción recomendada es usar maxIterations igual a 10.

11.3.4 Número de mejores soluciones para mantener

Para el número de mejores soluciones a mantener cuando se realiza la búsqueda en la población, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 5, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 14. Calibración de número de mejores soluciones para mantener

elitistCount	Mejor Fitness obtenido de	Mejor Fitness obtenido de rutas	Tiempo de ejecución

	repartición de recursos	para entregas	
2	2,008,150	54,573	11 minutos y 22 segundos
4	2,008,150	54,573	10 minutos y 52 segundos
5	2,008,160	54,570	10 minutos y 51 segundos
7	2,008,170	54,560	10 minutos y 41 segundos
10	2,008,170	54,560	10 minutos y 48 segundos

Podemos observar que se obtienen los mejores fitness para la repartición de recursos en los casos de elitistCount con valores de 7 y 10 y por el lado del fitness para rutas encontramos las soluciones con menor distancia recorrida también cuando el elitistCount tiene valor de 7 y 10. Como en este caso se demora menos en la ejecución con valor de 7, se escogerá esta solución.

11.3.5 Umbral de abandono

Para el umbral de abandono, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 5, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 15. Calibración de número de mejores soluciones para mantener

abandonmentthreshold	Mejor Fitness obtenido de repartición de recursos	Mejor Fitness obtenido de rutas para entregas	Tiempo de ejecución
2	2,008,120	54,704	11 minutos y 29 segundos
4	2,008,140	54,650	11 minutos y 26 segundos
5	2,008,170	54,586	11 minutos y 29 segundos

7	2,008,160	54,580	11 minutos y 27 segundos
10	2,008,170	54,573	11 minutos y 25 segundos

Se observa que se obtienen los mejores fitness para la repartición de recursos en los caso en los que el abandonmentthreshold toma los valores de 5 y 10. En cuanto al fitness para la obtención de rutas, el mejor resultado es con el valor de 10. Por lo tanto, la mejor opción es poner la variable abandonmentthreshold con valor de 10.

11.4 Calibración de variables para el algoritmo genético

A continuación, se mostrarán las pruebas realizadas para calibrar las variables de entrada que necesita el algoritmo genético para su ejecución.

11.4.1 Tamaño de población

Tal como se indicó previamente, la calibración de este parámetro comenzó con un valor inicial de 50. Posteriormente, este valor se ajustó tanto hacia valores más altos como más bajos, obteniendo los resultados que se detallan en la tabla siguiente:

Tabla 16. Calibración de tamaño de población

populationSize	**Mejor Fitness obtenido de repartición de recursos**	**Mejor Fitness obtenido de rutas para entregas**	**Tiempo de ejecución**
15	1,065,064	115,038	2 minutos y 14 segundos
25	1,344,070	119,158	3 minutos y 44 segundos
50	1,510,185	88,557	7 minutos y 20 segundos
100	1,761,426	78,275	15 minutos y 55 segundos

150	1,604,344	67,159	24 minutos y 19 segundos

Se observa que se obtienen los mejores fitness para la repartición de recursos en los casos en los que el populationSize toma los valores de 100 y 150. En cuanto al fitness para la obtención de rutas, el mejor resultado es con los mismos valores, pero vemos que la diferencia de tiempo es bastante entre las ejecuciones y en este tipo de soluciones el tiempo de respuesta también es crucial. Por lo tanto, la mejor opción es poner la variable populationSize con un valor de 100.

11.4.2 Tamaño de gen

Para el tamaño de gen, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 10, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 17. Calibración de tamaño de gen

geneLength	**Mejor Fitness obtenido de repartición de recursos**	**Mejor Fitness obtenido de rutas para entregas**	**Tiempo de ejecución**
2	1,528,782	72,455	15 minutos y 48 segundos
5	1,642,143	78,275	15 minutos y 50 segundos
10	1,761,426	78,275	15 minutos y 55 segundos
12	1,648,341	78,275	15 minutos y 32 segundos
15	1,622,530	78,275	15 minutos y 29 segundos

Se observa que se obtienen los mejores fitness en la repartición de recursos son en los valores del geneLength de 10 y 12. También se puede observar que solo en el valor de 2 el fitness para la obtención de rutas es distinto a los otros casos, pero que su valor de fitness para la repartición de recursos es bajo, por lo que esta opción se descarta. Por ende, la mejor opción es colocar esta variable con valor de 10.

11.4.3 Número de generaciones

Para el número de generaciones o bucles que se realizarán durante la búsqueda de las mejores soluciones, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 100, por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 18. Calibración de número de generaciones

generations	Mejor Fitness obtenido de repartición de recursos	Mejor Fitness obtenido de rutas para entregas	Tiempo de ejecución
25	1,780,847	67,156	6 minutos y 44 segundos
50	1,625,685	66,652	10 minutos y 10 segundos
100	1,761,426	78,275	15 minutos y 55 segundos
125	1,658,246	81,148	18 minutos y 42 segundos
150	1,559,576	87,128	21 minutos y 30 segundos

Se observa que se obtienen los mejores fitness en la repartición de recursos son en los valores del generations de 25 y 100. En cuanto al fitness de la planeación de rutas el mejor fitness es para el valor de 25, ya que en este se recorre menos distancia. Por ende, se va a tomar el valor de 25 para la variable de generations.

11.4.4 Tasa de mutación

Para la variable de tasa de mutación en el algoritmo, como se mencionó anteriormente, se iniciaron las pruebas con un valor de 0.1 que hace referencia a un 10% , por lo que a continuación se evaluarán valores mayores y menores a este.

Tabla 19. Calibración de tasa de mutación

mutationRate	Mejor Fitness obtenido de repartición de recursos	Mejor Fitness obtenido de rutas para entregas	Tiempo de ejecución
0.01	1,615,617	78,512	7 minutos y 9 segundos
0.05	1,788,026	64,297	7 minutos y 8 segundos
0.1	1,640,943	67,156	7 minutos y 5 segundos
0.15	1,762,451	72,451	7 minutos y 6 segundos
0.20	1,816,826	78,566	7 minutos y 6 segundos

Se observa que se obtienen los mejores fitness en la repartición de recursos son en los valores del mutationRate de 0.05 y 0.20, pero con mejor valor para el caso de 0.20 por poca diferencia. En cuanto al fitness de la planeación de rutas los mejores fitness son para los valores de 0.05 y 0.1 , ya que en estos se recorre menos distancia, pero el mejor valor se obtiene con el valor de 0.05. Por ende, se va a tomar el valor de 0.05 para la variable de mutationRate.

11.5 Validaciones realizadas

Este capítulo fue evaluado por el especialista en algoritmia, quien aprobó la información presentada. La evidencia de esta revisión se encuentra en el Anexo 14.

11.6 Discusión

En este capítulo se logró el resultado esperado R8, enfocado en la calibración de las variables requeridas para la implementación de los algoritmos de colonia de abejas y genético.

Este resultado permite determinar los valores óptimos que mejoran la calidad de las soluciones generadas por el algoritmo para el problema planteado. Para ello, fue indispensable ajustar los valores iniciales probados, buscando alcanzar el mayor valor de fitness posible.

12. Experimentación numérica

12.1 Introducción

En este capítulo se aborda el resultado 9, enfocado en la elaboración del informe de experimentación numérica para evaluar el desempeño de los algoritmos de colonia de abejas y genético. El objetivo es determinar cuál de estos algoritmos representa una solución más eficiente para el problema de distribución de recursos en situaciones de desastres naturales.

12.2 Resultados alcanzados

Este apartado describe en detalle las acciones realizadas para lograr el resultado esperado R9: “Informe de análisis numérico destinado a evaluar la eficiencia de los algoritmos de colonia de abejas y genético”.

12.2.1 Informe de experimentación numérica

En esta sección analiza los datos obtenidos de los algoritmos de colonia de abeja y genético mediante las pruebas de Shapiro-Wilk, F de Fisher y Prueba Z. Se utilizaron las variables calibradas en el capítulo anterior para obtener los mejores resultados en ambos algoritmos. Las pruebas se realizaron en RStudio y se detallan a continuación.

12.2.1.1 Recolección de datos

Para la recopilación de datos, se utilizaron los mejores resultados de la calibración de variables de ambos algoritmos. Luego, se resumieron los resultados agrupando la cantidad de recursos distribuidos desde los almacenes a los puntos de destino, facilitando el análisis organizado. A continuación, se presenta la tabla de resultados que se empleará en las pruebas estadísticas.

Tabla 20. Resultados agrupados del algoritmo de colonia de abejas

Recurso	Cusco	Los Olivos	Porcentaje atención
Analgesics	129102	342498	100.00%
Antihistamines	205178	544322	100.00%
Blankets	187576	497624	100.00%
Canned Olives	484800	399980	90.28%
Canned Peas	569981	400000	76.04%
Cookies	499980	469960	86.60%
Diapers	192558	510842	100.00%
Drinking Water	480022	399979	81.69%
Helmets	183798	487602	100.00%
Instant Noodles	261324	693276	100.00%
Medicinal Alcohol	349941	375836	87.36%
Nuts	448978	479965	94.57%
Sterile Gauze	208162	552238	100.00%
Toilet Paper	499984	400000	67.67%
Towels	189629	503071	100.00%
Tuna	449996	449946	69.02%
Wet Wipes	200442	531758	100.00%

Tabla 21. Resultados agrupados del algoritmo genético

Recurso	Cusco	Los Olivos	Porcentaje atención
Analgesics	98	91679	19.46%
Antihistamines	62163	164721	30.27%
Blankets	65046	311677	54.98%
Canned Olives	207130	172795	38.77%
Canned Peas	189340	282684	37.00%
Cookies	342687	187794	47.36%
Diapers	67939	170286	33.87%
Drinking Water	498528	286551	72.88%
Helmets	248	138754	20.70%
Instant Noodles	83570	486399	59.71%
Medicinal Alcohol	206	270678	32.61%
Nuts	110	196748	20.04%
Sterile Gauze	95	346110	45.53%
Toilet Paper	228	286105	21.53%
Towels	115	149887	21.65%
Tuna	141501	198676	26.09%
Wet Wipes	138182	127147	36.24%

En las Figuras 45 y 46 podemos observar una representación gráfica de los resultados generados por ambos algoritmos.

Figura 45. Gráfica de resultados de algoritmo de colonia de abejas para pruebas estadísticas

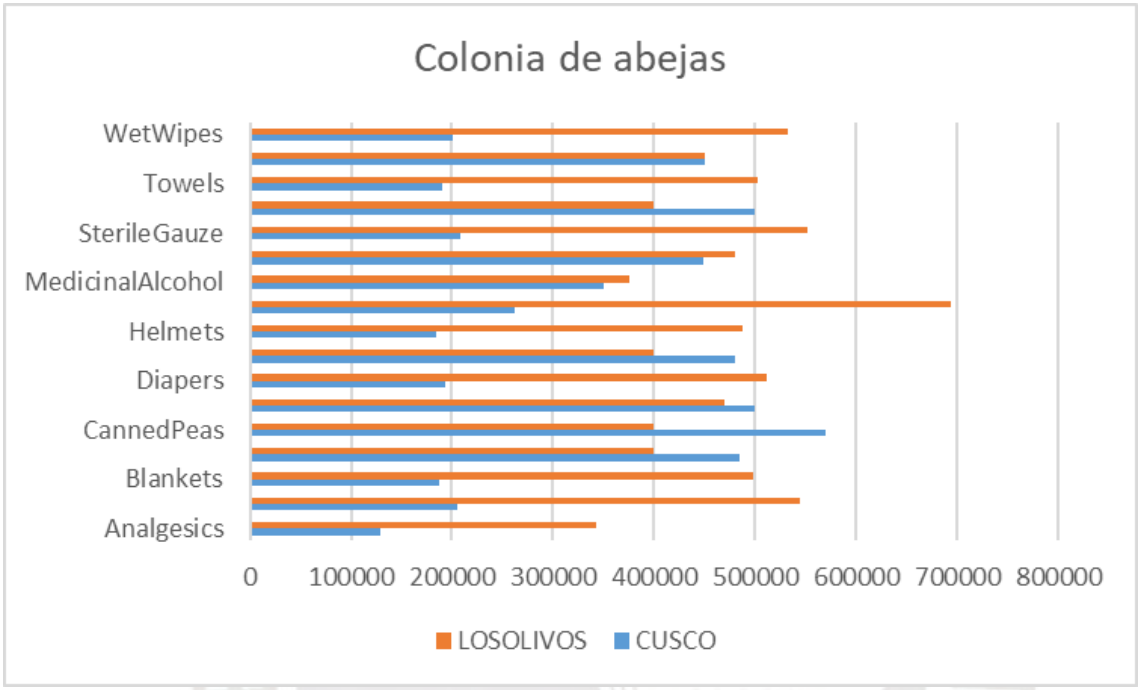
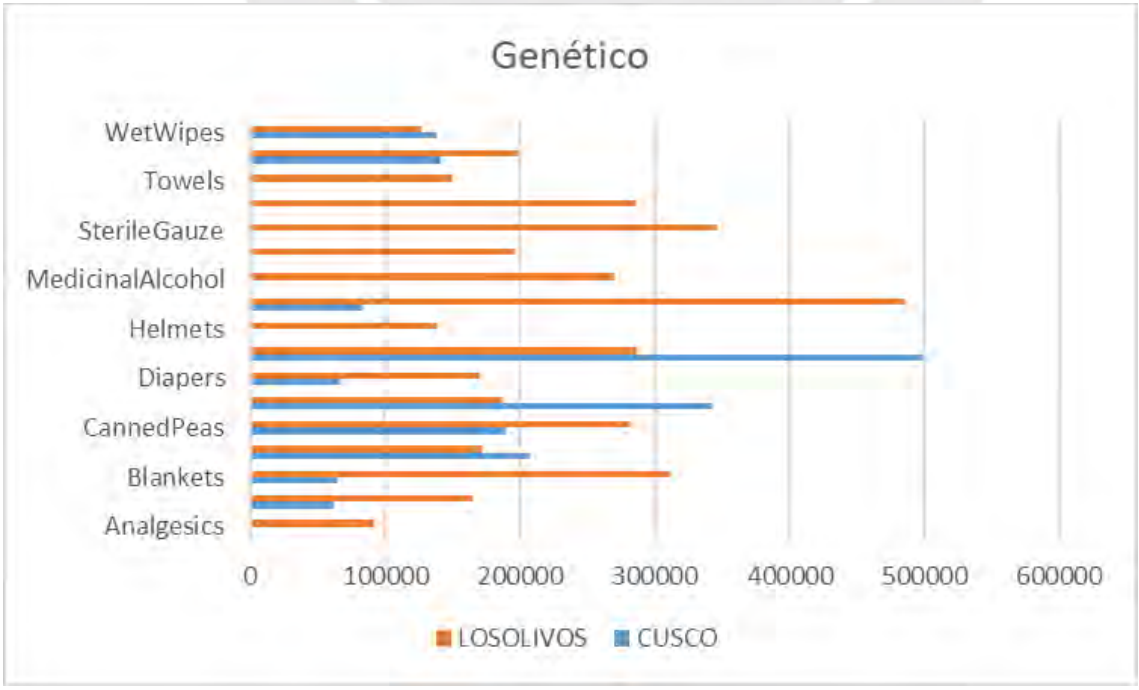


Figura 46. Gráfica de resultados de algoritmo genético para pruebas estadísticas



Con esto podemos observar a simple vista que parece que el algoritmo de colonia de abejas tiene mejores resultados, ya que logra repartir mayor cantidad de recursos. Esta premisa la podremos verificar por medio de las siguientes pruebas estadísticas.

12.2.1.2 Prueba de Shapiro-Wilk

Se utilizó la prueba de Shapiro-Wilk para verificar si los datos obtenidos siguen una distribución normal. Esta prueba es necesaria, ya que la normalidad en los datos es un requisito previo para la prueba Z.

Para la prueba de Shapiro-Wilk se realiza la validación de las muestras para cada algoritmo por separado, pero la hipótesis se mantiene dado que se tiene el mismo objetivo en ambos casos:

H0: La muestra sigue una distribución normal

H1: La muestra no sigue una distribución normal

Para el algoritmo de colonia de abejas:

Prueba de Shapiro-Wilk para el algoritmo de colonia de abejas

```
> plan_abejas <- as.numeric(resumen_plan_distribucion$LOSOLIVOS)
> shapiro_test_plan <- shapiro.test(plan_abejas)
> print(shapiro_test_plan)

        Shapiro-Wilk normality test

data:  plan_abejas
W = 0.93234, p-value = 0.2381
```

En la Figura 47 podemos observar que el valor del p-value es 0.2381 que es mayor que 0.05. Por lo tanto, no se rechaza la hipótesis nula, lo que indica que no hay suficiente evidencia para decir que los datos del algoritmo de abejas no siguen una distribución normal. En otras palabras, se concluye que los datos del algoritmo de abejas siguen una distribución normal.

Para el algoritmo de genético:

Prueba de Shapiro-Wilk para el algoritmo genético

```
> plan_genetico <- as.numeric(resumen_plan_distribucion_genetico$LOSOLIVOS)
> shapiro_test_plan_genetico <- shapiro.test(plan_genetico)
> print(shapiro_test_plan_genetico)

        Shapiro-Wilk normality test

data:  plan_genetico
W = 0.91261, p-value = 0.1107
```

En la Figura 48 podemos observar que el valor del p-value es 0.1107 que es mayor que 0.05. Por lo tanto, no se rechaza la hipótesis nula, lo que indica que no hay suficiente evidencia para decir que los datos del algoritmo genético no siguen una distribución normal. En otras palabras, se concluye que los datos del algoritmo genético siguen una distribución normal.

12.2.1.3 Prueba F de Fisher

Otro requisito para realizar la prueba Z es verificar si la varianza entre las muestras es significativamente diferente u homogénea. Para dar respuesta a este planteamiento se utiliza la prueba F de Fisher, en la cual las hipótesis son las siguientes:

H0: Las varianzas entre las muestras son homogéneas

H1: Las varianzas entre las muestras son diferentes

Figura 47. Prueba F de Fisher para los algoritmos

```
> fisher_test <- var.test(plan_abejas, plan_genetico)
> print(fisher_test)

        F test to compare two variances

data:  plan_abejas and plan_genetico
F = 0.73644, num df = 16, denom df = 16, p-value = 0.5477
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.266693 2.033562
sample estimates:
ratio of variances
          0.736435
```

En la Figura 47, a un nivel de significancia de 0.05, no hay evidencia suficiente para rechazar la hipótesis nula, ya que el p-valor obtenido es de 0.5477. Por lo tanto, se concluye que la varianza entre ambas muestras es homogénea.

12.2.1.4 Prueba Z

Finalmente se procede a realizar la prueba Z, luego de garantizar la normalidad y las varianzas homogéneas entre las muestras.

En este caso las hipótesis son las siguientes:

H0: La media del algoritmo de colonia de abejas es igual a la del algoritmo genético

H1: La media del algoritmo de colonia de abejas es diferente a la del algoritmo genético

Figura 48. Prueba Z para los algoritmos

```
> n_plan_abejas <- length(plan_abejas)
> n_plan_genetico <- length(plan_genetico)
> se_diff <- sqrt((sd_plan_abejas^2 / n_plan_abejas) + (sd_plan_genetico^2 / n_plan_genetico))
> z <- (mean_plan_abejas - mean_plan_genetico) / se_diff
> p_value <- 2 * (1 - pnorm(abs(z)))
> cat("Prueba Z\n")
Prueba Z
> cat("Valor Z:", z, "\n")
Valor Z: 7.77343
> cat("p-valor:", p_value, "\n")
p-valor: 7.549517e-15
> cat("mean_plan_abejas:", mean_plan_abejas, "\n")
mean_plan_abejas: 472876.3
> cat("mean_plan_genetico:", mean_plan_genetico, "\n")
mean_plan_genetico: 227570.1
```

En la Figura 50, el p-valor muy pequeño (7.549517e-15) indica una probabilidad extremadamente baja de que la diferencia entre las medias de los algoritmos ocurra por azar. Esto permite rechazar la hipótesis nula (H0) y aceptar la hipótesis alternativa (H1), concluyendo que la media del algoritmo de colonia de abejas es distinta y superior a la del algoritmo genético, lo que sugiere que el primer algoritmo ofrece mejores soluciones.

12.2.2 Validaciones realizadas

Este capítulo fue revisado y aprobado por el especialista en algoritmia respecto a la información presentada. La evidencia de esta revisión se encuentra en el Anexo 15.

12.3 Discusión de resultados

En este capítulo se detalla el logro del resultado esperado R9: "Realización del informe de análisis numérico destinado a evaluar la eficiencia de los algoritmos de colonia de abejas y genético". Se realizaron pruebas estadísticas para comparar ambos algoritmos: la prueba de Shapiro-Wilk confirmó la normalidad de los datos, la prueba F de Fisher verificó la homogeneidad de las varianzas, y la prueba Z mostró que la media del algoritmo de colonia de abejas es significativamente mayor que la del algoritmo genético (p-valor < 0.05). Estos análisis demuestran la superioridad del algoritmo de colonia de abejas en la distribución de recursos, ya que asigna eficientemente los recursos y cubre un mayor porcentaje de la demanda.

En conclusión, los resultados indican que el algoritmo de colonia de abejas es una herramienta efectiva y óptima para la distribución de ayuda en desastres naturales, cumpliendo con los objetivos planteados y superando al algoritmo genético.

13. Conclusiones y trabajos futuros

Este capítulo presenta las conclusiones obtenidas tras cumplir los objetivos establecidos en el proyecto de tesis. Además, incluye una sección dedicada a trabajos futuros, donde se sugieren posibles extensiones basadas en los hallazgos de la investigación realizada.

13.1 Conclusiones

El presente proyecto de tesis aborda el problema central de la "ineficiente distribución de ayuda humanitaria en caso de fenómenos naturales en el Perú", para lo cual se plantea el objetivo general de "implementar un algoritmo de colonia de abejas para optimizar la planificación de la distribución de ayuda humanitaria en casos de fenómenos naturales en el Perú". A continuación, se exponen las conclusiones obtenidas al finalizar el proyecto de tesis:

- Definición de Variables y Restricciones: Se logró determinar y establecer las variables y restricciones clave para la distribución de ayuda humanitaria. Estas variables se basaron en estudios previos y entrevistas con especialistas en logística humanitaria, lo que permitió una comprensión profunda del problema. Entre las variables clave se incluyeron la accesibilidad de las rutas, la urgencia de las necesidades, la capacidad de los medios de transporte, y la cantidad de recursos

disponibles. La función objetivo diseñada integró estas variables, permitiendo que el algoritmo optimice la distribución de ayuda de manera efectiva y eficiente.

- Implementación del Algoritmo de Colonia de Abejas: El algoritmo de colonia de abejas fue adaptado y codificado específicamente para el problema de distribución de ayuda humanitaria. Este proceso incluyó la definición de las estructuras de datos adecuadas, la implementación de las funciones auxiliares necesarias, y la integración de la función objetivo. La fase de codificación se realizó utilizando lenguajes de programación como Python y C++, asegurando una implementación robusta y eficiente. Además, se llevaron a cabo pruebas exhaustivas para validar el correcto funcionamiento del algoritmo, lo que incluyó pruebas de unidad y validaciones empíricas con datos simulados.

- Desarrollo de la Herramienta de Software: Se diseñó y desarrolló una interfaz gráfica que permite ejecutar el algoritmo de colonia de abejas, facilitando la interacción del usuario con el sistema. Esta herramienta ofrece funcionalidades para ingresar los datos del problema, configurar los parámetros del algoritmo y visualizar los resultados obtenidos. La interfaz fue creada para ser intuitiva y de fácil uso, permitiendo a los usuarios realizar simulaciones y análisis de manera eficiente. Su desarrollo se llevó a cabo siguiendo principios de diseño ágil de software, basándose en el marco de trabajo Extreme Programming y empleando metodologías como Kanban para garantizar la entrega continua de un software de alta calidad.

- Comparación de Algoritmos: Se llevó a cabo una comparación detallada entre el algoritmo de colonia de abejas y un algoritmo genético previamente empleado para la distribución de ayuda humanitaria. Este análisis incluyó pruebas estadísticas exhaustivas, como la prueba de normalidad Shapiro-Wilk, la prueba F para evaluar varianzas y la prueba Z para contrastar las medias de los resultados generados por ambos algoritmos. Los resultados indicaron que el algoritmo de colonia de abejas presenta un mejor desempeño en términos de eficiencia y cobertura de necesidades, especialmente en problemas de gran magnitud. En particular, el algoritmo de colonia de abejas demostró una mayor capacidad para manejar la complejidad y la escala de los problemas de distribución, asegurando una asignación más equitativa y rápida de los recursos disponibles.

- Validación de Resultados: Los resultados obtenidos confirman la hipótesis de que el algoritmo de colonia de abejas es una herramienta eficaz y eficiente para optimizar la distribución de ayuda humanitaria en escenarios de desastres naturales. Las pruebas empíricas y los análisis estadísticos respaldan la efectividad del algoritmo, mostrando mejoras significativas en comparación con soluciones anteriores. Además, se realizaron estudios de caso con datos reales y simulados que confirmaron la robustez y adaptabilidad del algoritmo a diferentes escenarios y condiciones. Esto no solo refuerza la viabilidad del algoritmo en contextos prácticos, sino que también demuestra su potencial para ser utilizado en operaciones de emergencia y planificación de recursos a gran escala.

En resumen, se concluye que la implementación del algoritmo de colonia de abejas para el problema de distribución de ayuda humanitaria es factible y proporciona una solución óptima y practicable en escenarios reales. Además, este trabajo contribuye al campo de la logística humanitaria al introducir un algoritmo que no había sido aplicado anteriormente en este contexto. Este enfoque innovador no solo incrementa la eficiencia y eficacia en la distribución de recursos durante emergencias, sino que también establece una base robusta para investigaciones y desarrollos futuros en el campo de la optimización de recursos.

13.2 Trabajos futuros

A continuación, se proponen algunas extensiones o trabajos futuros asociados al presente proyecto de tesis:

- Creación de una Población Inicial Óptima: Una posible mejora es desarrollar métodos para crear una población inicial óptima para el algoritmo de colonia de abejas, lo que podría mejorar aún más la eficiencia y los resultados del algoritmo. Esta mejora podría incluir técnicas de preprocesamiento de datos y heurísticas avanzadas para generar soluciones iniciales de alta calidad.

- Extensión a Otros Tipos de Recursos: El algoritmo de colonia de abejas podría adaptarse para la optimización de la distribución de otros recursos en contextos diferentes, como la asignación de recursos médicos en hospitales o la distribución de suministros en operaciones militares. Esto ampliaría el alcance y la aplicabilidad del algoritmo a diversas áreas.

- Integración con Sistemas de Información: El algoritmo y la herramienta desarrollada pueden ser integrados en sistemas de información empresariales o en plataformas de gestión de emergencias para facilitar su uso directo por parte de los administradores de ayuda humanitaria. Esta integración permitiría una implementación más amplia y práctica de la solución propuesta.

- Mejora de la Interfaz Gráfica: Desarrollar una interfaz gráfica más intuitiva y accesible, que incluya características avanzadas de visualización y análisis de datos, podría mejorar la experiencia del usuario y facilitar la toma de decisiones en tiempo real durante situaciones de emergencia.

- Investigación de Otros Algoritmos Bioinspirados: Explorar y comparar otros algoritmos bioinspirados, como los algoritmos de enjambre de partículas o los algoritmos de optimización basada en cuántica, podría proporcionar nuevas perspectivas y mejoras en la eficiencia de la distribución de ayuda humanitaria.

  Estos trabajos futuros tienen el potencial de mejorar significativamente la efectividad y eficiencia del algoritmo de colonia de abejas y expandir su aplicabilidad a una variedad de problemas de optimización en diferentes contextos.

Referencias

Baumeister y Leary (1997): Writing Narrative Literature Reviews. *Review of General Psychology,* 311 - 320

Tranfield, Denyer y Smart (2003): Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic Review. *British Journal of Management*, 14(3), 207–222

Webster & Watson (2002): Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly Vol. 26 No. 2.* 13- 23.

Hannah Snyder., (2019): Literature review as a research methodology: An overview and guidelines. *Journal of Business Research Volume 104.* 333-339

Denyer, D., & Tranfield, D. (2009): Producing a systematic review. *The Sage handbook of organizational research methods*. 671–689

Kitchenham, B. (2004): Procedures for Performing Systematic Reviews. *Keele University, Keele.* 33.

Mark Petticrew, Helen Roberts (2006): Systematic Reviews in the Social Sciences: A Practical Guide.

Aduviri Choque, & Robert Alonso. (2019). *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.

Ali, M., & Sucipto, H. (2021). Ant Colony Optimization Algorithm Implementation for Distribution

of Natural Disaster Relief Logistics in Jombang Regency Web Base. *IOP Conference Series: Earth and Environmental Science*, *704*(1), 012008. https://doi.org/10.1088/1755-1315/704/1/012008

Bakhshi, A., Aghsami, A., & Rabbani, M. (2022). A scenario-based collaborative problem for a relief supply chain during post-disaster under uncertain parameters: a real case study in Dorud. *Journal of Modelling in Management*. https://doi.org/10.1108/JM2-06-2021-0138

Bozorgi-Amiri, A., Jabalameli, M. S., Alinaghian, M., & Heydari, M. (2012). A modified particle swarm optimization for disaster relief logistics under uncertain environment. *The International Journal of Advanced Manufacturing Technology*, *60*(1–4), 357–371. https://doi.org/10.1007/s00170-011-3596-8

Chen, Y., Tadikamalla, P. R., Shang, J., & Song, Y. (2020). Supply allocation: bi-level programming and differential evolution algorithm for Natural Disaster Relief. *Cluster Computing*, *23*(1), 203–217. https://doi.org/10.1007/s10586-017-1366-6

Kebriyaii, O., Hamzehei, M., & Khalilzadeh, M. (2021). A disaster relief commodity supply chain network considering emergency relief volunteers: a case study. *Journal of Humanitarian Logistics and Supply Chain Management*, *11*(3), 493–521. https://doi.org/10.1108/JHLSCM-08-2020-0073

Mamashli, Z., Bozorgi-Amiri, A., Dadashpour, I., Nayeri, S., & Heydari, J. (2021). A heuristic-based multi-choice goal programming for the stochastic sustainable-resilient routing-allocation problem in relief logistics. *Neural Computing and Applications*, *33*(21), 14283–14309. https://doi.org/10.1007/s00521-021-06074-8

Nawazish, M., Padhi, S. S., & Edwin Cheng, T. C. (2022). Stratified delivery aid plans for humanitarian aid distribution centre selection. *Computers & Industrial Engineering*, *171*, 108451. https://doi.org/10.1016/j.cie.2022.108451

Nayeri, S., Asadi-Gangraj, E., & Emami, S. (2019). Metaheuristic algorithms to allocate and schedule of the rescue units in the natural disaster with fatigue effect. *Neural Computing and Applications*, *31*(11), 7517–7537. https://doi.org/10.1007/s00521-018-3599-6

Nayeri, S., Tavakkoli-Moghaddam, R., Sazvar, Z., & Heydari, J. (2022). A heuristic-based simulated annealing algorithm for the scheduling of relief teams in natural disasters. *Soft Computing*, *26*(4), 1825–1843. https://doi.org/10.1007/s00500-021-06425-6

Rabiei, P., & Arias-Aranda, D. (2021). Introducing a novel multi-objective optimization model for vehicle routing and relief supply distribution in post-disaster phase: combining fuzzy inference systems with NSGA-II and NRGA. *2021 6th International Conference on Transportation Information and Safety (ICTIS)*, 1226–1243. https://doi.org/10.1109/ICTIS54573.2021.9798556

Shokr, I., Jolai, F., & Bozorgi-Amiri, A. (2022). A collaborative humanitarian relief chain design for disaster response. *Computers & Industrial Engineering*, *172*, 108643. https://doi.org/10.1016/j.cie.2022.108643

Toathom, T., Promsuk, N., & Champrasert, P. (2021). Genetic Algorithm with Boosting based on Expected Value for Uncertain Routing. *2021 International Conference on Science & Contemporary Technologies (ICSCT)*, 1–6. https://doi.org/10.1109/ICSCT53883.2021.9642552

Vosooghi, Z., Mirzapour Al-e-hashem, S. M. J., & Lahijanian, B. (2022). Scenario-based redesigning of a relief supply-chain network by considering humanitarian constraints, triage, and volunteers' help. *Socio-Economic Planning Sciences*, *84*, 101399. https://doi.org/10.1016/j.seps.2022.101399

Wang, B. C., Qian, Q. Y., Gao, J. J., Tan, Z. Y., & Zhou, Y. (2021). The optimization of warehouse location and resources distribution for emergency rescue under uncertainty. *Advanced Engineering*

Informatics, *48*, 101278. https://doi.org/10.1016/j.aei.2021.101278

Wang, Y., Wang, X., Fan, J., Wang, Z., & Zhen, L. (2023). Emergency logistics network optimization with time window assignment. *Expert Systems with Applications*, *214*, 119145. https://doi.org/10.1016/j.eswa.2022.119145

Zargary, S., & Samouei, P. (2022). Production-Routing-Inventory in Post-Disaster Conditions: a Multi-Objective Mathematical Model and Two Algorithms. *Process Integration and Optimization for Sustainability*, *6*(4), 1163–1183. https://doi.org/10.1007/s41660-022-00274-y

Zavvar Sabegh, M. H., Mohammadi, M., & Naderi, B. (2017). Multi-objective optimization considering quality concepts in a green healthcare supply chain for natural disaster response: neural network approaches. *International Journal of System Assurance Engineering and Management*, *8*(S2), 1689–1703. https://doi.org/10.1007/s13198-017-0645-1

Organización Meteorológica Mundial (31 de agosto de 2021). Los desastres de índole meteorológica han aumentado en los últimos 50 años y han causado más daños, pero menos muertes. https://public.wmo.int/es/media/comunicados-de-prensa/los-desastres-de-%C3%ADndole-meteorol%C3%B3gica-han-aumentado-en-los-%C3%BAltimos-50#:~:text=Seg%C3%BAn%20el%20Atlas%20de%20la,3%2C64%20billones%20de%20d%C3%B3lares

Espinosa Bordón, Odalis (2008). Los desastres naturales y la sociedad / Natural disasters and society. Biblioteca Médica Nacional/Centro Nacional de Información de Ciencias Médicas.

Kumar, S.; Havey, T. (2013). Before and after disaster strikes: A relief supply chain decision support framework. International Journal of Production Economics , 145 (2), pp. 613-629.

Cozzolino, A. (2012). Humanitarian Logistics: Cross-Sector Cooperation in Disaster Relief Management. New York: Springer.

Balcik, B.; Beamon, B. M.; Krejci, C. C.; Muramatsu, K. M.; Ramirez, M. (2010). Coordination in humanitarian relief chains: Practices, challenges and opportunities. International Journal of Production Economics, 126 (1), pp. 22-34.

Centre for Research on the Epidemiology of Disasters - CRED. (2020). Disaster Year in Review 2019.

Organización Panamericana de la Salud. La Seguridad Alimentaria y Nutricional en Situaciones de Emergencia. https://www.paho.org/es/emergencias-salud/seguridad-alimentaria-nutricional-situaciones-emergencia

Pedro Ignacio Arcos González, Rafael Castro Delgado y Francisco del Busto Prado (2002). Desastres y salud pública: un abordaje desde el marco teórico de la epidemiología. Rev. Esp. Salud Publica vol.76 no.2

Organización de las Naciones Unidas (abril del 2021) . Cada vez más migrantes climáticos en Perú. https://es.unesco.org/courier/2021-4/cada-vez-mas-migrantes-climaticos-peru#:~:text=Seg%C3%BAn%20datos%20del%20IDMC%2C%20en,causa%20de%20las%20cat%C3%A1strofes%20naturales.

Daniel Bitrán Bitrán (1995). Impacto económico de los desastres naturales en la infraestructura de salud. Comisión Económica para América Latina y el Caribe.

Norma Carrillo Hidalgo, Enrique Guadalupe Gómez (2001). Desastres naturales y su influencia en el medio ambiente. Revista Del Instituto De investigación De La Facultad De Minas, Metalurgia Y Ciencias geográficas.

Julio Mario Daza, Jairo R. Montoya, Francesco Narducci (2009). Resolución del problema de enrutamiento de vehículos con limitaciones de capacidad utilizando un procedimiento metaheurístico de dos fases. Revista EIA (Escuela de Ingeniería de Antioquia, Medellín) p. 23-38

Instituto de ingeniería del conocimiento (21 de marzo de 2021). Algoritmos bioinspirados: optimización basada en la evolución natural. https://www.iic.uam.es/noticias/algoritmos-bioinspirados-optimizacion-basada-evolucion-natural/#:~:text=Los%20algoritmos%20bioinspirados%20abarcan%2C%20a,inmunol%C3%B3gicos%20y%20las%20redes%20neuronales.

Toth, P., & Vigo, D. (2002). The vehicle routing problem. Society for Industrial and Applied Mathematics. Retrieved from https://dl.acm.org/citation.cfm?id=505847

NU. CEPAL (2014). Panorama Social de América Latina 2014. Editorial CEPAL.

PuntoEdu (marzo, 2023). No basta un fenómeno natural para que ocurra un desastre, se requieren condiciones de vulnerabilidad. https://puntoedu.pucp.edu.pe/voces-pucp/no-basta-un-fenomeno-natural-para-que-ocurra-un-desastre-se-requieren-condiciones-de-vulnerabilidad/

ACNUR (mayo, 2019). Ayuda humanitaria, la única forma de sobrevivir para millones de personas. https://eacnur.org/es/actualidad/noticias/eventos/ayuda-humanitaria-la-unica-forma-de-sobrevivir-para-millones-de-personas

Microsoft. (2018). Visual Studio Code - Code Editing. Redefined. Retrieved April 14, 2018, from https://code.visualstudio.com/

Bjarne Stroustrup (2013). The C++ Programming Language, 4th Edition.

Foundation, P. S. (2018). Welcome to Python.org. Retrieved April 14, 2018, from https://www.python.org/

Jupyter, P. (2018). Project Jupyter. Retrieved April 14, 2018, from http://jupyter.org/

RStudio. (2020). Take control of your R code. https://rstudio.com/products/rstudio/#rstudio-desktop

Troncoso, C. E., & Daniele, E. G. (2004). Las entrevistas semiestructuradas como instrumentos de recolección de datos: una aplicación en el campo de las ciencias naturales. Universidad Nacional Del Comahue - Consejo Provincial de Educación de Neuquen, 4(2), 12.

Extreme Programming. (2013). Extreme Programming: A gentle introduction. http://www.extremeprogramming.org/

Razali, N. M., & Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk , Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. Journal of Statistical Modeling and Analytics, 2(1), 21–33. https://doi.org/doi:10.1515/bile-2015-0008

Fallas, J. (2012). Análisis de Varianza. Universidad para la Cooperación Internacional. http://www.ucipfg.com/Repositorio/MGAP/MGAP-05/BLOQUE-ACADEMICO/Unidad-2/complementarias/analisis_de_varianza_2012.pdf

Massey, A., & Miller, S. J. (2016). Tests of Hypotheses Using Statistics. Mathematics Department, Brown University, Providence, RI 2912, 1–32.

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

INDECI. (2008). Lecciones Aprendidas del Sur. Retrieved from http://bvpad.indeci.gob.pe/doc/pdf/esp/doc1259/doc1259-contenido.pdf

García, Silvana Yanet (2018). Optimización mediante el algoritmo de colonia de abejas artificial. Universidad NAcional de La Pampa - Facultad de Ingeniería.

Isla López, D. J. (2018). Implementación de Búsqueda Local en el Algoritmo de Colonia Artificial de Abejas para Resolver Problemas de Optimización con Restricciones. Tesis de Ingeniería en Sistemas Computacionales. Instituto Tecnológico de Veracruz, Tecnológico Nacional de México.

Anexos

Anexo 1: Formulario de extracción

Para observar el formulario de extracción de datos completo dirigirse al siguiente enlace: "Formulario de extracción".

Anexo 2: Plan de proyecto

1) Justificación

El Perú, ubicado en una región geográficamente compleja y expuesta a diversos fenómenos naturales, se enfrenta regularmente a desastres como terremotos, inundaciones, deslizamientos de tierra y eventos climáticos extremos (INDECI, 2008). Estos eventos tienen un impacto significativo en la infraestructura, la economía y, lo más importante, en la vida de los ciudadanos. La gestión eficiente y efectiva de la distribución de ayuda durante y después de los desastres naturales es fundamental para minimizar las pérdidas humanas y materiales, y para acelerar la recuperación y reconstrucción de las comunidades afectadas.

Uno de los mayores desafíos en la planificación logística de la distribución de ayuda es la asignación óptima de recursos limitados, como alimentos, medicinas y refugio, a las áreas afectadas (Bozorgi-Amiri et al., 2012). Se sabe que el planeamiento logístico involucra la solución a diversos tipos de problemas de optimización, incluyendo problemas de ubicación de almacenes, diseño de la cadena de suministro, distribución a gran escala, distribución de última milla, evacuación y planeamiento de inventario (Ortuño et al., 2013). La falta de una estrategia adecuada puede resultar en una distribución ineficiente, retrasos en la entrega de suministros esenciales y una respuesta inadecuada a las necesidades de la población afectada. En este contexto, se requiere una solución innovadora que pueda abordar eficientemente este problema logístico complejo.

Algunos métodos tradicionales pueden basarse en enfoques heurísticos que no garantizan la solución más eficiente y pueden conducir a una distribución inadecuada de los recursos disponibles. Otros sistemas pueden depender en gran medida de la experiencia y el juicio humano, lo que puede llevar a decisiones subjetivas y falta de coordinación. Hasta el momento se han realizado diversas soluciones con distintos enfoques y planteamientos de sistemas de distribución; sin embargo, muchos de ellos no son óptimos y presentan limitaciones significativas (Aduviri Choque, R., & Robert Alonso, 2019). Por lo tanto, se plantea la implementación de un algoritmo de colonia de abejas para la planificación de la distribución de ayuda en caso de fenómenos naturales en el Perú, ya que es una propuesta necesaria y relevante en el contexto actual.

En contraste, la implementación de un algoritmo de colonia de abejas ofrece una solución optimizada y basada en principios naturales. Este enfoque utiliza la inteligencia colectiva de las abejas para buscar la mejor solución en términos de tiempo, recursos y eficiencia (Zargary & Samouei, 2022). Aprovecha la capacidad de las abejas para adaptarse a diferentes entornos y encontrar las rutas más eficientes, y aplica esos principios a la distribución de ayuda en caso de desastres naturales.

Al adoptar este enfoque, se superan las limitaciones de los sistemas tradicionales, mejorando la precisión y la eficiencia de la distribución de ayuda. El algoritmo de colonia de abejas puede considerar múltiples variables y restricciones, como la ubicación de los centros de suministro, las demandas de las áreas afectadas, las condiciones de transporte y las capacidades logísticas disponibles. Al hacerlo, se asegura que los recursos sean asignados de manera óptima, evitando duplicaciones innecesarias y garantizando que lleguen a las áreas más necesitadas en el momento adecuado.

Los principales beneficios de la aplicación de este algoritmo en la optimización de la asignación de tablas a unidades de almacenamiento son:

- Optimización de recursos: El algoritmo de colonia de abejas permite asignar de manera óptima los recursos limitados, como alimentos, medicinas y refugio, a las áreas afectadas por desastres naturales. Esto garantiza una distribución eficiente y equitativa, evitando la escasez en algunas áreas y el desperdicio en otras.
- Eficiencia en la planificación: El algoritmo de colonia de abejas encuentra soluciones óptimas en términos de tiempo y recursos, considerando múltiples factores como la geografía, las capacidades de transporte y las necesidades de la población. Esto permite una planificación más eficiente y efectiva, acelerando la entrega de ayuda y minimizando los tiempos de respuesta.
- Toma de decisiones informada: El algoritmo de colonia de abejas se basa en datos y algoritmos de optimización, lo que ayuda a tomar decisiones informadas y basadas en evidencia. La información en tiempo real sobre las necesidades de las áreas afectadas, los recursos disponibles y las capacidades logísticas permite una toma de decisiones más precisa y estratégica.
- Reducción de pérdidas humanas y materiales: Al optimizar la distribución de ayuda, el algoritmo contribuye a minimizar las pérdidas humanas y materiales durante y después de los desastres naturales. Al acelerar la entrega de suministros esenciales, se brinda asistencia oportuna a las comunidades afectadas, lo que ayuda a salvar vidas y a reducir el sufrimiento en situaciones de emergencia.

2) Viabilidad

La implementación de un algoritmo de colonia de abejas para la planificación de la distribución de ayuda en caso de fenómenos naturales en el Perú es altamente viable considerando las herramientas y metodologías disponibles. Las herramientas de desarrollo como C++, Visual

Code, Python, Jupyter Lab y RStudio ofrecen un conjunto completo de opciones para la implementación y análisis requeridos. Estos lenguajes son ampliamente utilizados y cuentan con una gran cantidad de recursos en línea, lo que facilitará el proceso de desarrollo y solución de posibles dificultades.

Además, la aplicación de metodologías como Extreme Programming, así como las pruebas estadísticas como la prueba de Shapiro-Wilk, prueba F, prueba Z y pruebas unitarias, proporcionan un marco sólido para garantizar la calidad y eficacia del algoritmo implementado. Estas metodologías y pruebas asegurarán que el algoritmo esté debidamente probado y optimizado, cumpliendo con los objetivos de la tesis.

En cuanto a la viabilidad económica, el proyecto no requerirá una inversión significativa, ya que se utilizarán herramientas de código abierto y recursos ya disponibles. Esto reduce los costos asociados y facilita el acceso a las herramientas necesarias para llevar a cabo la implementación del algoritmo.

Por lo tanto, el proyecto de implementación de un algoritmo de colonia de abejas para la planificación de la distribución de ayuda en caso de fenómenos naturales en el Perú es viable debido a las herramientas de desarrollo disponibles, las metodologías adecuadas y la baja inversión económica requerida. Estos factores contribuyen a la realización exitosa de la tesis y garantizan un enfoque sólido en la resolución de la problemática planteada.

3) Alcance

El presente proyecto de tesis pertenece al área de ciencias de la computación, más específicamente a la rama de algoritmos de optimización y tiene como objetivo principal implementar por completo un algoritmo de colonia de abejas para la planificación de la distribución de ayuda humanitaria en caso de fenómenos naturales en el Perú. Este algoritmo permitirá optimizar la

tarea de la planificación de la distribución de ayuda en caso de desastres, para así maximizar la optimización de recursos y la eficiencia de la planificación en estos casos.

En ese sentido, en primer lugar, se procederá a realizar un análisis y planteamiento de las variables, factores, y sus respectivas relaciones más importantes a considerar al momento de realizar esta tarea. Posteriormente, se procederá a definir las estructuras de datos necesarias para implementar el algoritmo de colonia de abejas, luego se diseñará el algoritmo en base a la información recopilada.

A continuación, se implementará el algoritmo de colonia de abejas y adicionalmente, se realizará el diseño e implementación de un algoritmo genético encontrado en la literatura, esto con el objetivo de ser comparado con el algoritmo de colonia de abejas planteado y corroborar la validez de su aplicación en la tarea de planificación de la distribución de ayuda humanitaria . La comparación de algoritmos se realizará mediante un proceso de experimentación numérica sobre los resultados obtenidos, esta experimentación se llevará a cabo mediante el uso de diversas pruebas estadísticas.

Finalmente, se implementará una interfaz que facilite la ejecución de los algoritmos planteados, esta recibirá la información necesaria para correr los algoritmos y también mostrará los resultados de la ejecución, sin embargo, cabe mencionar que el alcance de este proyecto no comprende el desarrollo de un sistema de información que contenga como funcionalidad a los algoritmos implementados.

4) Limitaciones

La formulación y los resultados del algoritmo estarán condicionados por la disponibilidad de información recolectada. En dicha información se puede no obtener todos los datos que se

necesitan para el planteamiento del algoritmo, por lo que algunos se tendrán que generar de forma aleatoria. Del mismo modo, un factor importante es la capacidad de procesamiento de la computadora en la que se ejecutará, pero dado que no se ha trabajado previamente con este tipo de algoritmos no se puede medir el impacto de este punto.

5) Riesgos

En la Tabla 22 se muestran los riesgos identificados junto con la probabilidad de ocurrencia, el impacto sobre el proyecto, la severidad del riesgo y el plan de contingencia en caso de que este se concrete.

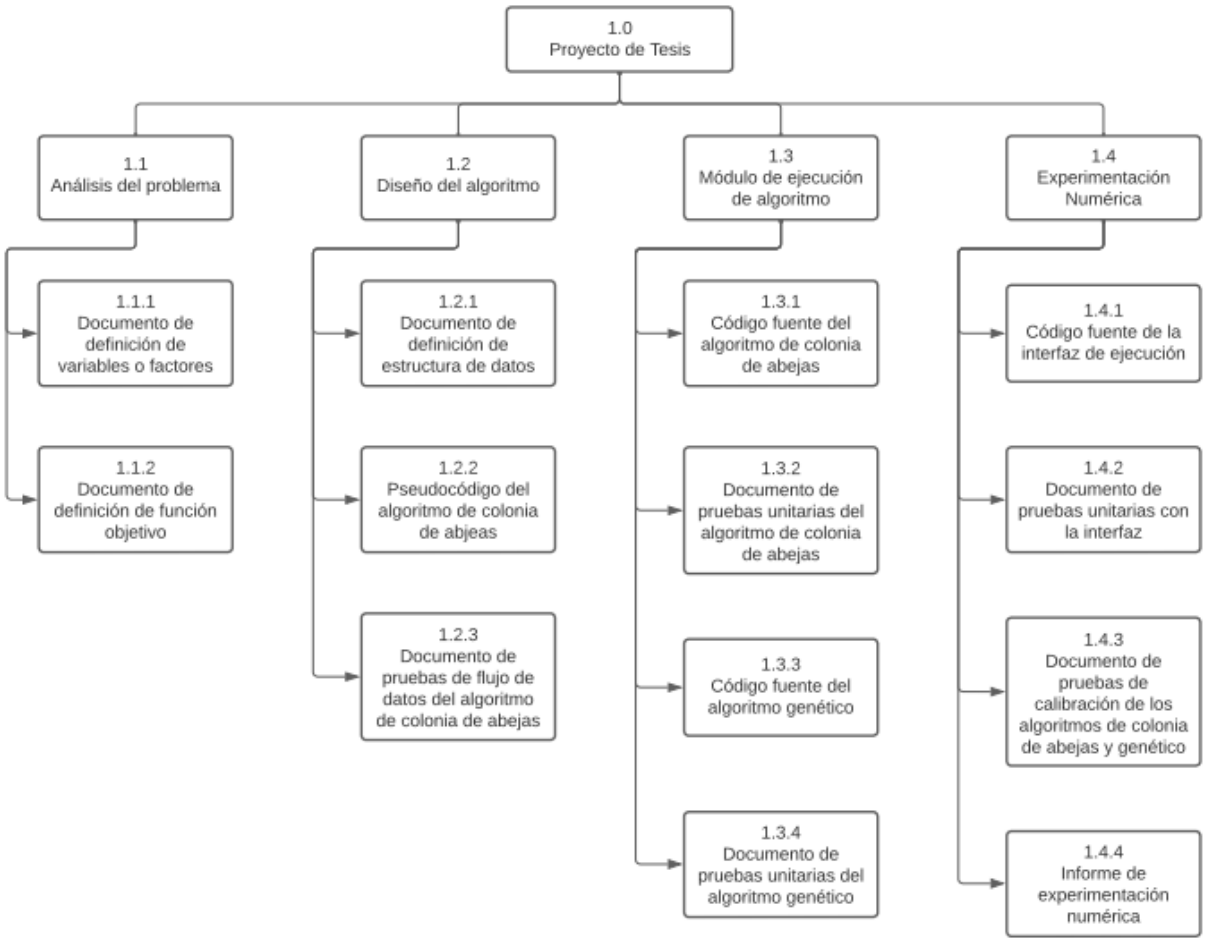
Tabla 22. *Riesgos del proyecto*

Descripción	**Síntomas**	**Mitigación**	**Contingencia**
Poco o nulo acceso a información de terceros (información acerca de bases de datos reales)	No se obtiene respuestas de las personas que brindan la información de terceros	Gestionar el acceso a la información de manera oportuna	En caso de que no se consiga, generar información propia de manera aleatoria
Problemas de conectividad al momento de realizar actividades programadas	Fallas constantes en la conectividad en los días previos a las actividades programadas	Contratar un plan celular para acceder a la reunión	Activar el plan celular y acceder a la reunión programada
Especialistas no realizan la validación de los entregables en los tiempos establecidos	No se obtiene respuesta de los especialistas en las comunicaciones previas a la fecha de entrega	Planificar las reuniones de revisión de manera oportuna	Solicitar ayuda de un especialista alterno para la revisión de entregables
Mala planificación del proyecto	Retraso en las entregas de las actividades planificadas	Planificar un horario a la semana para avanzar las actividades planteadas	Replantear el calendario de actividades

6) Estructura de descomposición del trabajo

La estructura de descomposición del trabajo puede ser visualizada en la Figura 51.

Figura 49. Estructura de descomposición del trabajo



7) Lista de tareas

A continuación, en la Tabla 23 se presenta la lista de tareas en base a la estructura de descomposición planteada.

Tabla 23. *Lista de tareas*

Nro.	Tarea	Duración estimada (días)	Esfuerzo asociado (horas/persona)	Costo estimado (Soles)
1	Entregable 4 de Tesis 1	0	0	0
2	Exposición Final Tesis 1	0	0	0
3	Definir las variables o factores más relevantes	1	6	120
4	Definir la función objetivo	1	6	120
5	Reunión de validación con el asesor y realizar ajustes	2	4	80
6	Validar la definición de las variables o factores más relevantes con un especialista	1	2	40
7	Validar la definición de la función objetivo con un especialista	1	2	40
	Subtotal	**6**	**20**	**400**
8	Definir la estructura de datos	2	12	240
9	Reunión de validación con el asesor y realizar ajustes	1	4	80
10	Validar estructura de datos con un especialista en algoritmia	1	2	40
11	Exposición 1	1	4	80
12	Diseñar el pseudocódigo del algoritmo de colonia de abejas	4	24	480
13	Reunión de validación con el asesor y realizar ajustes	1	4	80
14	Realizar pruebas de flujo de datos sobre el diseño del algoritmo de colonia de abejas	2	12	240
15	Validar pseudocódigo del algoritmo por un especialista en algoritmia	1	4	80
16	Exposición 2	1	2	40
	Subtotal	**14**	**68**	**1360**
17	Codificar el algoritmo de colonia de abejas	5	40	800
18	Realizar pruebas unitarias del algoritmo de colonia de abejas	1	6	120
19	Reunión de validación con el asesor y realizar ajustes	1	4	80
20	Realizar correcciones	1	5	100

21	Exposición 3	1	4	80
22	Codificar la interfaz de ejecución para los algoritmos	2	16	320
23	Realizar pruebas unitarias de la interfaz de ejecución	1	6	120
24	Reunión de validación con el asesor y realizar ajustes	1	4	80
25	Exposición 4	1	2	40
	Subtotal	**14**	**87**	**1740**
26	Avance parcial y correcciones finales	5	30	600
27	Exposición parcial	1	4	80
	Subtotal	**6**	**34**	**680**
28	Codificar el algoritmo genético	5	40	800
29	Realizar pruebas unitarias del algoritmo genético	1	6	120
30	Reunión de validación con el asesor y realizar ajustes	1	4	80
31	Exposición 5	1	2	40
32	Realizar correcciones	1	5	100
33	Reunión de validación con el asesor y realizar ajustes	1	4	80
34	Realizar pruebas unitarias del algoritmo genético	1	6	120
	Subtotal	**11**	**67**	**1340**
35	Realizar experimentación numérica	8	40	800
36	Exposición 6	1	4	80
	Subtotal	**9**	**44**	**880**
37	Correcciones finales	6	24	480
38	Entregable final	1	4	80
	Subtotal	**7**	**28**	**560**
	TOTAL	**67**	**348**	**6960**

8) Cronograma del proyecto

A continuación, en la Tabla 24 se presenta el cronograma del proyecto en base a la estructura de

descomposición de trabajo y la lista de tareas.

Tabla 24. *Cronograma del proyecto*

Nro.	Tarea	Duración planificada (días)	Esfuerzo planificado (horas/persona)	Dependencia	Fecha inicio	Fecha fin
1	Entregable 4 de Tesis 1	0	0	-	08/06/2023	19/06/2023
2	Exposición Final Tesis 1	0	0	-	20/06/2023	30/06/2023
3	Definir las variables o factores más relevantes	1	6	1	18/03/2024	19/03/2024
4	Definir la función objetivo	1	6	1-2	20/03/2024	21/03/2024
5	Reunión de validación con el asesor y realizar ajustes	2	4	1-3	22/03/2024	24/03/2024
6	Validación de definición de las variables o factores más relevantes con respecto a tesis a mejorar	1	2	1-4	25/03/2024	26/03/2024
7	Validación de la definición de la función objetivo con respecto a tesis a mejorar	1	2	1-5	27/03/2024	28/03/2024
8	Definir la estructura de datos	2	12	1-5	29/03/2024	31/03/2024
9	Reunión de validación con el asesor y realizar ajustes	1	4	1-7	01/04/2024	02/04/2024
10	Validar estructura de datos con un especialista en algoritmia	1	2	1-8	03/04/2024	04/04/2024
11	Exposición 1	1	4	1-9	05/04/2024	05/04/2024
12	Diseñar el pseudocódigo del	3	24	1-10	06/04/2024	08/04/2024

	algoritmo de colonia de abejas					
13	Reunión de validación con el asesor y realizar ajustes	1	4	1-11	09/04/2024	09/04/2024
14	Realizar pruebas de flujo de datos sobre el diseño del algoritmo de colonia de abejas	1	12	1-12	10/04/2024	10/04/2024
15	Validar pseudocódigo del algoritmo por un especialista en algoritmia	1	4	1-13	11/04/2024	11/04/2024
16	Exposición 2	1	2	1-14	12/04/2024	12/04/2024
17	Codificar el algoritmo de colonia de abejas	3	40	1-15	13/04/2024	19/04/2024
18	Exposición 3	1	4	1-19	19/04/2024	19/04/2024
19	Codificar el algoritmo de colonia de abejas	3	40	1-15	20/04/2024	23/04/2024
20	Realizar pruebas unitarias del algoritmo de colonia de abejas	1	6	1-16	24/04/2024	25/04/2024
21	Reunión de validación con el asesor y realizar ajustes	1	4	1-17	25/04/2024	25/04/2024
22	Exposición 4	1	2	1-23	26/04/2024	26/04/2024
23	Codificar la interfaz de ejecución para los algoritmos	3	16	1-20	27/04/2024	30/04/2024
24	Realizar pruebas unitarias de la interfaz de ejecución	2	6	1-21	01/05/2024	02/05/2024
25	Exposición parcial	1	4	1-25	03/05/2024	10/05/2024
26	Codificar el algoritmo genético	5	40	1-26	13/05/2024	17/05/2024

27	Realizar pruebas unitarias del algoritmo genético	1	6	1-27	18/05/2024	19/05/2024
28	Reunión de validación con el asesor y realizar ajustes	1	4	1-28	20/05/2024	21/05/2024
29	Exposición 5	1	2	1-29	22/05/2024	23/05/2024
30	Realizar correcciones	1	5	1-30	24/05/2024	25/05/2024
31	Reunión de validación con el asesor y realizar ajustes	1	4	1-31	26/05/2024	27/05/2024
32	Realizar pruebas unitarias del algoritmo genético	1	6	1-32	28/05/2024	29/05/2024
33	Realizar experimentación numérica	8	40	1-33	30/05/2024	06/06/2024
34	Exposición 6	1	4	1-34	06/06/2024	07/06/2024
35	Correcciones finales	6	24	1-35	08/06/2024	13/06/2024
36	Entregable final	1	4	1-36	14/06/2024	15/06/2024
	Total	**67**	**348**	**-**	08/06/2023	15/06/2024

9) Lista de recursos

A continuación, se describen los recursos necesarios para el desarrollo del presente proyecto de tesis.

- **Personas involucradas y necesidades de capacitación**

  En cuanto a las personas involucradas en el desarrollo del proyecto se tienen principalmente dos:

  - **El tesista**

    Es el alumno que elabora el proyecto de tesis con el objetivo de obtener el título universitario.

- **El asesor**

  Es el especialista encargado de guiar al alumno tesista en el desarrollo del proyecto de tesis .

- **Materiales requeridos para el proyecto**

  No aplica.

- **Estándares utilizados en el proyecto**

En relación a los estándares utilizados en el proyecto se tiene lo siguiente:

- **PMBOK parte 2: El estándar para la dirección de proyectos**

  Uso de algunas buenas prácticas para la dirección de proyectos propuestas por el instituto de administración de proyectos PMI en la guía PMBOK 6ta edición.

- **Equipamiento**

Con respecto al equipamiento necesario para el desarrollo del proyecto se tiene lo siguiente:

- **Computadora**

  Este equipamiento está disponible para el tesista del proyecto y es necesario para la realización de todos los componentes del proyecto de tesis.

- **Herramientas requeridas**

Con respecto a las herramientas requeridas se tiene las siguientes:

- **IDE para el lenguaje C++**

  Esta herramienta de construcción de software es necesaria para la codificación de los algoritmos propuestos en el presente proyecto de tesis.

- **IDE para el lenguaje Python**

  Esta herramienta de construcción de software es necesaria para la codificación de las interfaces a usar en el presente proyecto de tesis.

- **IDE para el lenguaje R**

  Esta herramienta es necesaria para realizar la experimentación numérica sobre los

resultados obtenidos por los algoritmos planteados.

- **Herramienta en la nube para control de versiones de software**

  Es necesaria una herramienta de control de versiones para mantener seguros los avances del desarrollo del proyecto de tesis y gestionar sus cambios de manera óptima.

- **Herramienta de comunicación (Zoom, Google meets, WhatsApp, entre otros)**

  Se requiere de herramientas de comunicación para poder realizar las reuniones de revisión con el asesor del proyecto y las reuniones con el especialista en algoritmia.

- **Herramienta de ofimática (Word y Excel)**

  Finalmente se requiere de herramientas de ofimática para la redacción de los documentos necesarios en el presente proyecto de tesis.

10) Costeo del proyecto

A continuación, en la Tabla 25 se plantea un análisis de costos para el desarrollo del presente proyecto de tesis.

Tabla 25. *Análisis de costos*

Ítem	**Descripción**	**Unidad**	**Cantidad**	**Valor unitario (S/.)**	**Monto parcial (S/.)**	**Monto total (S/.)**
1.	**Personas involucradas**	---	---	---	---	**10,060**
1.1	Estudiante 1	Horas	348	20	6960	--
1.2	Asesor 1	Horas	62	50	3100	
2.	**Bienes y equipos**	---	---	---	---	**826**
2.1	Laptop	Horas	413	2	826	--
3.	**Servicios**	---	---	---	---	**480**
3.1	Internet	Mes	4	120	480	---
	Total	---	---	---	---	**11,366**

Anexo 3: Validación de variables, restricciones y función objetivo

Para observar el documento de validación de las variables, restricciones y función objetivo planteadas en el presente trabajo con respecto a la tesis a mejorar del ingeniero Aduviri dirigirse al siguiente enlace:

https://docs.google.com/document/d/1WAvejbdtcwcNZiPg8x37OEkIZC1Qs1Xd/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

También se adjunta el enlace al documento donde se realizaron unas pruebas de los valores de la función objetivo para unos casos.

https://docs.google.com/spreadsheets/d/1LzKdiLkHfW2Ha5ysDkHj_HlNPGOCCmo1/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 4: Estructuras auxiliares

- Lugares destinos: Contiene la información general de los lugares que requieren de la entrega de recursos. También almacena la información general de las demás localidades que no están solicitando recursos, dado que esto puede ser cambiante en el tiempo.

Atributo	Ejemplo
Descripción	Lima, Ancón
Código Ubigeo	150102
Latitud	-11.702568503
Longitud	-77.0958901385
Cantidad población	69243
Nivel de urgencia	1

- Recurso: Esta estructura servirá para almacenar la información de la cantidad de los recursos en stock que se tienen en los almacenes para repartirlos en las localidades que requieren de ayuda.

Atributo	Ejemplo
Nombre	Atún
Unidad de medida	Lata
Cantidad	1500000

- Demanda por lugar: Tiene la información de los recursos que necesita cada uno de los lugares que requieren de ayuda.

Atributo	Ejemplo
Lugar	Ayacucho, Pacapausa
Código Ubigeo	050704
Nivel de urgencia	4
Demanda de recursos	Agua potable -> 4200 Atún -> 4800 Gasas estériles -> 1500 Algodón -> 4000

Anexo 5: Validación de estructuras de datos

Para observar el documento de validación de las estructuras de datos planteadas en el presente trabajo dirigirse al siguiente enlace:

https://docs.google.com/document/d/1wUv-0aPwCKyOMHIVVZu18eJ3L2u9WPiV/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 6: Pruebas de caja blanca al pseudocódigo del algoritmo de colonia de abejas.

Tabla 26. Datos iniciales para prueba de caja blanca del algoritmo de colonia de abejas.

Puntos del grafo

Puntos normales	Iquitos	Tacna	Chiclayo
Almacenes	Estadio Nacional (Lima)	Estadio Huancayo (Huancayo)	

Recursos Disponibles

	Estadio Nacional (Lima)	Estadio Huancayo (Huancayo)
Agua	140	60
Atún	50	40
Arroz	70	70
Papa	120	80

Costo de envio por km a los puntos destino

	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	2.1	1.65	1.35	1.5
Estadio Huancayo (Huancayo)	1.65	1.2	1.5	1.65

Demanda de Recursos

	Iquitos	Tacna	Chiclayo	Puno
Agua	50	70	80	60
Atún	40	20	15	10
Arroz	35	25	40	50
Papa	40	30	40	60

Nivel de Urgencia

Iquitos	2
Tacna	3
Chiclayo	3
Puno	4

Para este caso se está tomando el valor de N (número de abejas) como 2 y el valor de Cmax es 1. Luego de inicializar los datos generales, se procede a la generación de nuevas soluciones por parte de las abejas empleadas. En este caso se mostrará los datos correspondientes a un solución (plan de distribución) que generaría una abeja empleada.

Tabla 27. Plan de distribución generado por abeja empleadora

Distribución de recursos

Envio de agua (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	40	0	0	20
Estadio Huancayo (Huancay	10	10	40	0

Envio de atún (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	0	0	0	15
Estadio Huancayo (Huancay	0	0	15	10

Envio de arroz (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	0	30	10	10
Estadio Huancayo (Huancay	20	0	20	0

Envio de papa (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	10	0	10	0
Estadio Huancayo (Huancay	5	20	5	0

Distancias de rutas de entrega

Km de las rutas				
	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	50	42	65	80
Estadio Huancayo (Huancay	75	40	65	48

Seguidamente se calcula la aptitud de dicho plan de distribución. Esto se realizaría por medio de los cálculos planteados para la función objetivo y lo que se obtendría sería similar a lo siguiente:

Tabla 28. Evaluación de aptitud

Evaluación de aptitud					
Cobertura de necesidades	252.107			Funcion objetivo=	280.6252528
Costo de tranporte	730.5	0.00137			
Atencion prioritaria a urgencias:	28.5167				

Luego se procedería con la obtención de la mejor solución hasta el momento, pero dado que para el ejemplo se ha tomado que solo se tiene un plan de distribución generado, este vendría a ser el más óptimo hasta el momento. Después de esto, las abejas observadoras procederán a generar una solución en base al universo de soluciones que se tiene hasta el momento.

Tabla 29. Plan de distribución generado por abeja observadora

Distribución de recursos

Envio de agua (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	40	0	40	20
Estadio Huancayo (Huancayo)	10	10	40	0

Envio de atún (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	10	5	0	15
Estadio Huancayo (Huancayo)	15	0	15	10

Envio de arroz (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	5	20	10	10
Estadio Huancayo (Huancayo)	15	15	20	10

Envio de papa (cantidad)

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	10	10	10	0
Estadio Huancayo (Huancayo)	10	20	10	0

Distancias de rutas de entrega

Km de las rutas

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	50	38	65	62
Estadio Huancayo (Huancayo)	60	36	50	48

De esta nueva solución generada también se debe calcular su aptitud

Tabla 30. Evaluación de aptitud

Evaluación de aptitud

Cobertura de necesidade:	**373.3988095**			Funcion objetivo=	408.5997166
Costo de tranporte	644.85	**0.00155**			
Atencion prioritaria a urge	**35.19935631**				

En este caso podemos observar que el valor de la función objetivo de la última solución es mejor

que la anterior que se tenía almacenada como la mejor solución hasta el momento, por lo que esta solución nueva pasaría a ser la nueva mejor solución.

Dado que se colocó la variable Cmax con valor 1 para este ejemplo, ya no se haría otro bucle en búsqueda de nuevas soluciones, así que la solución que devolvería el algoritmo vendría a ser la última solución generada por la abeja observadora.

Este fue un ejemplo corto para el flujo de caja del algoritmo de colonia de abejas planteado para la presente tesis, si se desea ver un ejemplo a más detalle se puede visualizar el siguiente link: https://docs.google.com/spreadsheets/d/1xhfO7CqFI9ghmYuvhzA7lEUNyP_gWh9K/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 7: Validación de pseudocódigo del algoritmo de colonia de abejas

Para observar el documento de validación de pseudocódigo planteado para el algoritmo de colonia de abejas en el presente trabajo dirigirse al siguiente enlace: https://docs.google.com/document/d/1SepGZV4fCx88EPb9cIezY0bdKq_Y21sA/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 8: Pruebas de caja blanca al pseudocódigo del algoritmo genético.

Tabla 31. Datos iniciales para prueba de caja blanca del algoritmo genético.

Puntos del grafo			
Puntos normales	Iquitos	Tacna	Chiclayo
Almacenes	Estadio Nacional (Lima)	Estadio Huancayo (Huancayo)	
Recursos Disponibles			

	Estadio Nacional (Lima)	Estadio Huancayo (Huancayo)
Agua	140	60
Atún	50	40
Arroz	70	70
Papa	120	80

Costo de envio por km a los puntos destino

	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	2.1	1.65	1.35	1.5
Estadio Huancayo (Huancayo)	1.65	1.2	1.5	1.65

Demanda de Recursos

	Iquitos	Tacna	Chiclayo	Puno
Agua	50	70	80	60
Atún	40	20	15	10
Arroz	35	25	40	50
Papa	40	30	40	60

Nivel de Urgencia

Iquitos	2
Tacna	3
Chiclayo	3
Puno	4

Para este caso se está tomando el valor de tamaño de población como 3 y el valor de número de generaciones es 1.

Luego de inicializar los datos generales, se procede a la generación de soluciones iniciales aleatorias. En este caso se mostrará los datos correspondientes a una solución (plan de distribución) que se generaría en esta etapa.

Tabla 32. Plan de distribución generado en inicialización de población

Distribución de recursos

Envio de agua (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	50	0	20	0	70	140
Estadio Huancayo (Huancayo)	0	60	0	0	60	60

Envio de atún (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	0	0	0	5	5	50
Estadio Huancayo (Huancayo)	0	10	0	0	10	40

Envio de arroz (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	0	0	12	10	22	70
Estadio Huancayo (Huancayo)	0	10	5	0	15	70

Envio de papa (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	10	10	0	0	20	120
Estadio Huancayo (Huancayo)	0	20	15	0	35	80

Distancias de rutas de entrega

Km de las rutas

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	50	40	65	80
Estadio Huancayo (Huancayo)	80	45	65	48

En realidad en para los parámetros iniciales ingresados, se generan tres soluciones como el indicado previamente, pero por motivo de practicidad se mostró solo uno de estos. Seguidamente se calcula la aptitud de dicho plan de distribución. Esto se realizaría por medio de los cálculos planteados para la función objetivo y lo que se obtendría sería similar a lo siguiente:

Tabla 33. Evaluación de aptitud

Evaluación de aptitud

Cobertura de necesidade:	165.2785714			Funcion objetivo=	182.9596394
Costo de tranporte	741.45	0.00135			
Atencion prioritaria a urge	17.67971927				

Luego se procede con la selección de dos padres del universo de soluciones y en base a estos se procede con la acción de cruzamiento de genes del cromosoma entre ellos. Con esto se obtendrían 2 hijos resultantes de dicho cruzamiento.

Tabla 34. Hijo generado del proceso de cruzamiento

Distribución de recursos

Envio de agua (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	50	0	20	0	70	140
Estadio Huancayo (Huancayo)	0	60	0	0	60	60

Envio de atún (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	0	0	0	40	40	50
Estadio Huancayo (Huancayo)	0	40	0	0	40	40

Envio de arroz (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	0	30	10	10	50	70
Estadio Huancayo (Huancayo)	20	0	20	0	40	70

Envio de papa (cantidad)

	Puntos de entrega					
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	10	0	10	0	20	120
Estadio Huancayo (Huancayo)	5	20	5	0	30	80

Distancias de rutas de entrega

Km de las rutas

	Puntos de entrega			
Puntos Iniciales	Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	50	42	65	80
Estadio Huancayo (Huancayo)	75	40	65	48

Después de esto, se procede con la mutación de los hijos resultantes para seguir con la búsqueda

de mejores soluciones.

Tabla 35. Proceso de mutación de hijos

Distribución de recursos

Envio de agua (cantidad)

Puntos Iniciales	Puntos de entrega: Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	50	0	20	0	70	140
Estadio Huancayo (Huancayo)	0	60	0	0	60	60

Envio de atún (cantidad)

Puntos Iniciales	Puntos de entrega: Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	15	0	10	25	50	50
Estadio Huancayo (Huancayo)	5	20	10	5	40	40

Envio de arroz (cantidad)

Puntos Iniciales	Puntos de entrega: Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	0	30	10	10	50	70
Estadio Huancayo (Huancayo)	20	0	20	0	40	70

Envio de papa (cantidad)

Puntos Iniciales	Puntos de entrega: Iquitos	Tacna	Chiclayo	Puno	Cantidad distribuida	Stock
Estadio Nacional (Lima)	10	0	10	0	20	120
Estadio Huancayo (Huancayo)	5	20	5	0	30	80

Distancias de rutas de entrega

Km de las rutas

Puntos Iniciales	Puntos de entrega: Iquitos	Tacna	Chiclayo	Puno
Estadio Nacional (Lima)	50	42	65	80
Estadio Huancayo (Huancayo)	75	40	65	48

Y de igual modo se procede con el cálculo de la aptitud de estas nuevas soluciones generadas.

Tabla 36. Evaluación de aptitud

Evaluación de aptitud

Cobertura de necesidade:	**349.6071429**		Funcion objetivo=	384.5873264
Costo de tranporte	730.5	**0.00137**		
Atencion prioritaria a urge	**34.97881463**			

En este caso podemos observar que el valor de la función objetivo de la última solución es mejor que la anterior que se tenía almacenada como la mejor solución hasta el momento, por lo que esta solución nueva pasaría a ser la nueva mejor solución.

De esta forma se volvería a buscar padres, realizarles el cruzamiento y mutar a los hijos resultantes. En los ejemplos anteriores, podemos observar que la mejor solución hasta el momento es la segunda solución mostrada.

Este fue un ejemplo corto para el flujo de caja del algoritmo genético planteado para la presente tesis, si se desea ver un ejemplo a más detalle se puede visualizar el siguiente link:

https://docs.google.com/spreadsheets/d/1TfaoMclOV2Sl9lDgQLBl8z2aAQ0HCRZS/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 9: Validación de pseudocódigo del algoritmo genético

Para observar el documento de validación de pseudocódigo planteado para el algoritmo genético en el presente trabajo dirigirse al siguiente enlace:

https://docs.google.com/document/d/1teO6eAvkCf3H2fPnqToKuMF7z5es2OlG/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 10: Direcciones URL para los algoritmos y la interfaz

- Algoritmo colonia de abejas

https://drive.google.com/file/d/1jFejgkkDEPyLCyJEw06MnNz-C2sa0y4x/view?usp=sharing

- Algoritmo genético

https://drive.google.com/file/d/120anI69KUKliRc-koJEuVEUwjL22p7q7/view?usp=sharing

- Interfaz gráfica

https://drive.google.com/file/d/1zjD82jUaMZym4AS5V9nO6aKzy0nK0DO5/view?usp=sharing

- Back en Django

https://drive.google.com/file/d/1t2IBox3LfAQ30XnY2kX6XseNyHp3e8hH/view?usp=sharing

Anexo 11: Caso de prueba para la ejecución de código colonia de abejas

Tabla 37. Parámetros utilizados para la ejecución de prueba del algoritmo colonia de abejas

Parámetros

`int numBees = 50;`
`int maxIterations = 100;`
`int maxAttempts = 10;`
`int elitistCount = 5;`
`double abandonmentThreshold = 5;`

• numBees: Número de abejas • maxIterations: Número máximo de iteraciones • maxAttempts: Límite máximo de intentos para generar una ruta válida • elitistCount: Número de mejores soluciones para mantener • abandonmentThreshold: Umbral de abandono

Tabla 38. Salida de plan de distribución de recursos

Plan de distribución de recursos
Envio de: [Cookies] (BC001) Puntos Iniciales HUANCAYO SAN MIGUEL CUSCO 148440 156640 LOS OLIVOS 121540 143340
En esta imagen podemos observar el plan de distribución de los recursos que se deberían enviar del recurso “Cookies” a Huancayo y San Miguel tomando como puntos de partidas los almacenes en Cusco y Los Olivos

Tabla 39. Salida de resumen plan de distribución de recursos

Resumen plan de distribución de recursos

```
TOTAL ENTREGADO de: [Cookies] (BC001)
CUSCO                          499960
LOS OLIVOS                     469960
Total entregado de Cookies: 969920
```

Recurso	Stock Inicial	Stock Final
Analgesics	600000	595284
Antihistamines	600000	592505
Blankets	800000	793148
CannedBeans	1300000	1287769
CannedOlives	1200000	0
CannedPeas	1800000	1785243
Cookies	970000	80
Cotton	1600000	1589660
Diapers	900000	892966
DrinkingWater	1200000	1189228
Helmets	650000	646286
InstantNoodles	980000	970454
MedicinalAlcohol	800000	791692
Milk	1200000	1188244
Nuts	980000	970177
Rice	1400000	1386854
Soap	1000000	988614
SterileGauze	790000	784396
ToiletPaper	1700000	1686700
Towels	900000	893073
Tuna	1500000	1487232
WetWipes	700000	693678

Podemos observar la cantidad en total de “Cookies” que se mandaron desde los almacenes

iniciales (Cusco y Los Olivos) a los puntos de destino, de igual forma se puede visualizar al final el stock inicial de los recursos y también el stock final luego de la repartición de recursos.

Tabla 40. Salida de plan de rutas

Plan de rutas

```
1) Ruta de LOS OLIVOS a HUANCAYO:
De 150117 (LOS OLIVOS) a 150717 (SAN BARTOLOME) hay 62.6697 km.
De 150717 (SAN BARTOLOME) a 150714 (RICARDO PALMA) hay 13.4801 km.
De 150714 (RICARDO PALMA) a 150118 (LURIGANCHO) hay 18.7406 km.
De 150118 (LURIGANCHO) a 150704 (CARAMPOMA) hay 56.9023 km.
De 150704 (CARAMPOMA) a 120802 (CHACAPALPA) hay 61.8999 km.
De 120802 (CHACAPALPA) a 120433 (YAULI) hay 38.4363 km.
De 120433 (YAULI) a 120407 (EL MANTARO) hay 17.7275 km.
De 120407 (EL MANTARO) a 120805 (MOROCOCHA) hay 85.2801 km.
De 120805 (MOROCOCHA) a 120212 (NUEVE DE JULIO) hay 96.059 km.
De 120212 (NUEVE DE JULIO) a 120415 (MARCO) hay 31.0821 km.
De 120415 (MARCO) a 120101 (HUANCAYO) hay 59.908 km.

------------------------------------------------------------------------------------------
2) Ruta de LOS OLIVOS a SAN MIGUEL:
De 150117 (LOS OLIVOS) a 150135 (SAN MARTIN DE PORRES) hay 2.72579 km.
De 150135 (SAN MARTIN DE PORRES) a 070101 (CALLAO) hay 4.46826 km.
De 070101 (CALLAO) a 150136 (SAN MIGUEL) hay 7.50797 km.

------------------------------------------------------------------------------------------
```

En la imagen se puede observar una parte del archivo del detalle de las rutas generadas para dirigirnos de los puntos iniciales a los puntos destino que están solicitando los recursos. Por ejemplo en la imagen podemos apreciar las rutas a seguir para ir desde Los Olivos a Huancayo y a San Miguel

Tabla 41. Resumen de plan de rutas

Resumen de plan de rutas

Type	From Code	From Name	To Code	To Name	Distance (km)
inicio	150117	LOS OLIVOS	120101	HUANCAYO	542.185715
inicio	150117	LOS OLIVOS	150136	SAN MIGUEL	14.702021
inicio	080101	CUSCO	120101	HUANCAYO	2017.929653
inicio	080101	CUSCO	150136	SAN MIGUEL	2829.164522

En este reporte se visualiza la cantidad de kilómetros que tienen las rutas especificadas en plan de rutas como tal.

Anexo 12: Caso de prueba para la ejecución de código genético

Tabla 42. Parámetros utilizados para la ejecución de prueba del algoritmo genético

Parámetros

```
int populationSize = 100;
int geneLength = 10;
int generations = 100;
```

- populationSize: Tamaño de la población
- geneLength: Longitud de los genes
- generations: Número de generaciones

Tabla 43. Salida de plan de distribución de recursos

Plan de distribución de recursos

```
Envio de: [Cookies] (BC001)
Puntos Iniciales        HUANCAYO        SAN MIGUEL      TORRES CAUSANA
CUSCO                   141919          134655          222581
LOS OLIVOS              128660          164120          177075
```

En esta imagen podemos observar el plan de distribución de los recursos que se deberían enviar del recurso “Cookies” a Huancayo, San Miguel y Torres Causana tomando como puntos de partidas los almacenes en Cusco y Los Olivos

Tabla 44. Salida de resumen plan de distribución de recursos

Resumen plan de distribución de recursos

```
TOTAL ENTREGADO de: [Cookies] (BC001)
CUSCO                         499155
LOS OLIVOS                    469855
Total entregado de Cookies: 969010
```

Recurso	Stock Inicial	Stock Final
Analgesics	600000	595616
Antihistamines	600000	592505
Blankets	800000	793148
CannedBeans	1300000	1287769
CannedOlives	1200000	1128
CannedPeas	1800000	1785243
Cookies	970000	990
Cotton	1600000	1589931
Diapers	900000	892966
DrinkingWater	1200000	1189228
Helmets	650000	646286
InstantNoodles	980000	971239
MedicinalAlcohol	800000	791692
Milk	1200000	1188244
Nuts	980000	970177
Rice	1400000	1386854
Soap	1000000	988614
SterileGauze	790000	784396
ToiletPaper	1700000	1686700
Towels	900000	893073
Tuna	1500000	1486961
WetWipes	700000	694104

Podemos observar la cantidad en total de “Cookies” que se mandaron desde los almacenes iniciales (Cusco y Los Olivos) a los puntos de destino, de igual forma se puede visualizar al final el stock inicial de los recursos y también el stock final luego de la repartición de recursos.

Tabla 45. Salida de plan de rutas

Plan de rutas

```
1) Ruta de LOS OLIVOS a TORRES CAUSANA:
De 150117 (LOS OLIVOS) a 150104 (BARRANCO) hay 19.9433 km.
De 150104 (BARRANCO) a 070105 (LA PUNTA) hay 17.4073 km.
De 070105 (LA PUNTA) a 150110 (COMAS) hay 20.8074 km.
De 150110 (COMAS) a 150118 (LURIGANCHO) hay 26.7858 km.
De 150118 (LURIGANCHO) a 150132 (SAN JUAN DE LURIGANCHO) hay 19.3136 km.
De 150132 (SAN JUAN DE LURIGANCHO) a 150105 (BRE♦A) hay 15.2506 km.
De 150105 (BRE♦A) a 150144 (SANTA MARIA DE HUACHIPA /1) hay 13.6312 km.
De 150144 (SANTA MARIA DE HUACHIPA /1) a 070106 (VENTANILLA) hay 25.7885 km.
De 070106 (VENTANILLA) a 150128 (RIMAC) hay 19.2947 km.
De 150128 (RIMAC) a 150143 (VILLA MARIA DEL TRIUNFO) hay 20.4607 km.
De 150143 (VILLA MARIA DEL TRIUNFO) a 150123 (PACHACAMAC) hay 11.917 km.
De 150123 (PACHACAMAC) a 150119 (LURIN) hay 8.31626 km.
De 150119 (LURIN) a 150138 (SANTA MARIA DEL MAR) hay 19.7421 km.
De 150138 (SANTA MARIA DEL MAR) a 150133 (SAN JUAN DE MIRAFLORES) hay 35.3097 km.
De 150133 (SAN JUAN DE MIRAFLORES) a 150120 (MAGDALENA DEL MAR) hay 12.9924 km.
De 150120 (MAGDALENA DEL MAR) a 150141 (SURQUILLO) hay 6.23555 km.
De 150141 (SURQUILLO) a 150108 (CHORRILLOS) hay 8.84446 km.
De 150108 (CHORRILLOS) a 150112 (INDEPENDENCIA) hay 23.2251 km.
De 150112 (INDEPENDENCIA) a 150134 (SAN LUIS) hay 10.9931 km.
De 150134 (SAN LUIS) a 150111 (EL AGUSTINO) hay 3.92454 km.
De 150111 (EL AGUSTINO) a 150140 (SANTIAGO DE SURCO) hay 9.69081 km.
De 150140 (SANTIAGO DE SURCO) a 150115 (LA VICTORIA) hay 7.07355 km.
De 150115 (LA VICTORIA) a 150121 (PUEBLO LIBRE) hay 5.27365 km.
De 150121 (PUEBLO LIBRE) a 070103 (CARMEN DE LA LEGUA REYNOSO) hay 4.48154 km.
De 070103 (CARMEN DE LA LEGUA REYNOSO) a 150107 (CHACLACAYO) hay 35.6432 km.
De 150107 (CHACLACAYO) a 150727 (SANTA CRUZ DE COCACHACRA) hay 23.5894 km.
De 150727 (SANTA CRUZ DE COCACHACRA) a 150109 (CIENEGUILLA) hay 28.9102 km.
```

En la imagen se puede observar una parte del archivo del detalle de las rutas generadas para dirigirnos de los puntos iniciales a los puntos destino que están solicitando los recursos. Por ejemplo en la imagen podemos ver una parte de la ruta a seguir para ir desde Los Olivos a Torres Causana.

Tabla 46. Resumen de plan de rutas

Resumen de plan de rutas

Type	From Code	From Name	To Code	To Name	Distance (km)
inicio	150117	LOS OLIVOS	160110	TORRES CAUSANA	18254.812164
inicio	150117	LOS OLIVOS	120101	HUANCAYO	2233.901569
inicio	150117	LOS OLIVOS	150136	SAN MIGUEL	68.795978
inicio	080101	CUSCO	160110	TORRES CAUSANA	28495.059609
inicio	080101	CUSCO	120101	HUANCAYO	2292.629550
inicio	080101	CUSCO	150136	SAN MIGUEL	1914.163131

En este reporte se visualiza la cantidad de kilómetros que tienen las rutas especificadas en plan de rutas como tal.

Anexo 13: Contenido de los archivos para los algoritmos

1) Archivos utilizados para la prueba de integración en los algoritmos:

- **Archivo de almacen.csv:**

  En este archivo se manda primero información de los tipos de recursos que se tienen, así como el stock total que se tiene en todos los almacenes y de igual forma se manda el stock de cada recurso en cada almacén que se tenga. Esto último hace referencia a la sección Quantity-Código, en el cual Código se refiere al código ubigeo de donde están ubicados los almacenes.

```
Resource,UnitOfMeasurement,QuantityTotal,Quantity-080101,Quantity-150117,Code
DrinkingWater,Bottle (7L),1200000,800000,400000,DW001
Tuna,Fish Can,1500000,700000,800000,TN001
Blankets,Units,800000,200000,600000,BK001
MedicinalAlcohol,Bottle (1L),800000,300000,500000,MA001
SterileGauze,Units,790000,200000,590000,SG001
Cotton,Bag (50g),1600000,900000,700000,CT001
Analgesics,Tablet (8 units),600000,400000,200000,PK001
Antihistamines,Tablet (8 units),600000,400000,200000,AH001
ToiletPaper,Package (2 units),1700000,1000000,700000,TP001
Towels,Units,900000,500000,400000,TW001
WetWipes,Units,700000,400000,300000,WW001
Soap,Units,1000000,300000,700000,SP001
Rice,Kg,1400000,600000,800000,RC001
CannedPeas,Kg,1800000,1200000,600000,PT001
Diapers,Units,900000,400000,500000,DP001
Helmets,Units,650000,300000,350000,HL001
InstantNoodles,Units,980000,490000,490000,IN001
Nuts,Kg,980000,500000,480000,NT001
Milk,Units,1200000,400000,800000,MK001
CannedBeans,Kg,1300000,600000,700000,FR001
CannedOlives,Kg,1200000,800000,400000,VE001
Cookies,Units,970000,500000,470000,BC001
```

- **Archivo de demanda.csv:**

  En este archivo se manda primero la información general de los lugares que solicitan recursos y luego se detalla la cantidad de recursos solicitados de cada uno de los elementos que se tiene en stock.

```
DEPARTMENT,PROVINCE,DISTRICT,CAPITAL,UBIGEO,UrgencyLevel,DrinkingWater,Tuna,Blankets,MedicalAlcohol,
AMAZONAS,BAGUA,ARAMANGO,ARAMANGO,010202,3,4001,3863,2657,2546,2722,3976,1301,2808,4601,2247,2342,
AMAZONAS,BAGUA,BAGUA,BAGUA,010201,1,3510,4400,2796,2619,2718,4824,2693,1063,4416,2482,2144,3100,3340,
AMAZONAS,BONGARA,CUISPES,CUISPES,010305,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
AMAZONAS,BONGARA,FLORIDA,FLORIDA (POMACOCHAS),010306,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

- **Archivo de informacion_general.csv:**

  En este archivo se manda la información general de los lugares que se tendrán mapeados tanto para la distribución de recursos como para la planificación de las rutas. En este archivo se

ponen datos como la latitud y longitud de los lugares, el nombre de los distritos, la capital, el código ubigeo, su nivel de urgencia, entre otros.

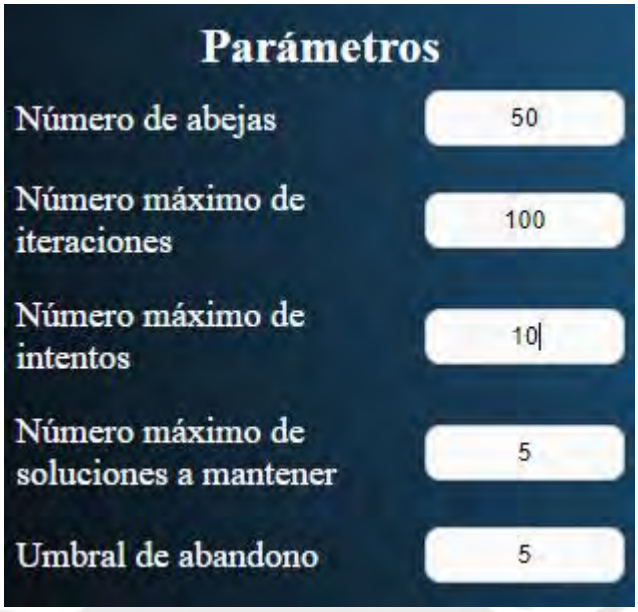
```
LATITUD,LONGITUD,CCDD,NOMBDEP,CCPP,NOMBPROV,CCDI,NOMBDIST,CAPITAL,UBIGEO,IDPROV,CODIGO,CNT_CCPP,DESCRIPCIO,POBLACION,FECHA,DAT_POB,Nivel de Urgencia,
-5.38043114766,-78.4483373822,01,AMAZONAS,02,BAGUA,02,ARAMANGO,ARAMANGO,010202,0102,010202,87,ARAMANGO,10733,2020,INEI,3,
-5.58091273601,-78.5209827359,01,AMAZONAS,02,BAGUA,01,BAGUA,BAGUA,010201,0102,010201,24,BAGUA,29538,2020,INEI,1,
-5.91114537432,-77.9147927248,01,AMAZONAS,03,BONGARA,05,CUISPES,CUISPES,010305,0103,010305,23,CUISPES,690,2020,INEI,0,
-5.82360433687,-77.9592017998,01,AMAZONAS,03,BONGARA,06,FLORIDA,FLORIDA (POMACOCHAS),010306,0103,010306,42,FLORIDA,6520,2020,INEI,0,
-5.94537389736,-78.0234317792,01,AMAZONAS,03,BONGARA,07,JAZAN,PEDRO RUIZ GALLO,010307,0103,010307,26,JAZAN,8190,2020,INEI,0,
-5.95174280364,-77.8256583611,01,AMAZONAS,03,BONGARA,01,JUMBILLA,JUMBILLA,010301,0103,010301,30,JUMBILLA,1372,2020,INEI,0,
-5.93922111946,-77.7753586409,01,AMAZONAS,03,BONGARA,08,RECTA,RECTA,010308,0103,010308,11,RECTA,210,2020,INEI,0,
-5.99638868782,-77.8790594036,01,AMAZONAS,03,BONGARA,09,SAN CARLOS,SAN CARLOS,010309,0103,010309,18,SAN CARLOS,488,2020,INEI,0,
-5.88374674985,-78.052089155,01,AMAZONAS,03,BONGARA,10,SHIPASBAMBA,SHIPASBAMBA,010310,0103,010310,13,SHIPASBAMBA,1639,2020,INEI,0,
```

2) Prueba de integración con el algoritmo de colonia de abejas:

- Primero se ingresan los archivos de datos correspondientes y se selecciona el algoritmo de colonia de abejas.

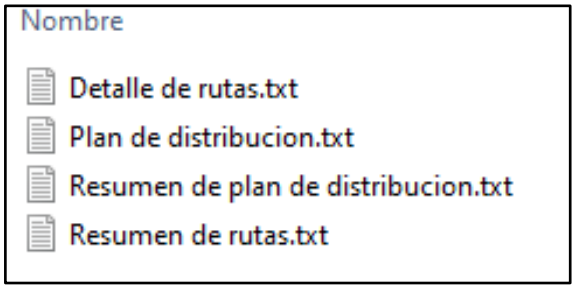
- Luego de escoger los archivos de datos, en la pantalla se visualizarán los datos de los almacenes iniciales. Seguidamente, se procede a escoger los lugares de destino.

- De igual forma se ingresan los parámetros del algoritmo de colonia de abejas.



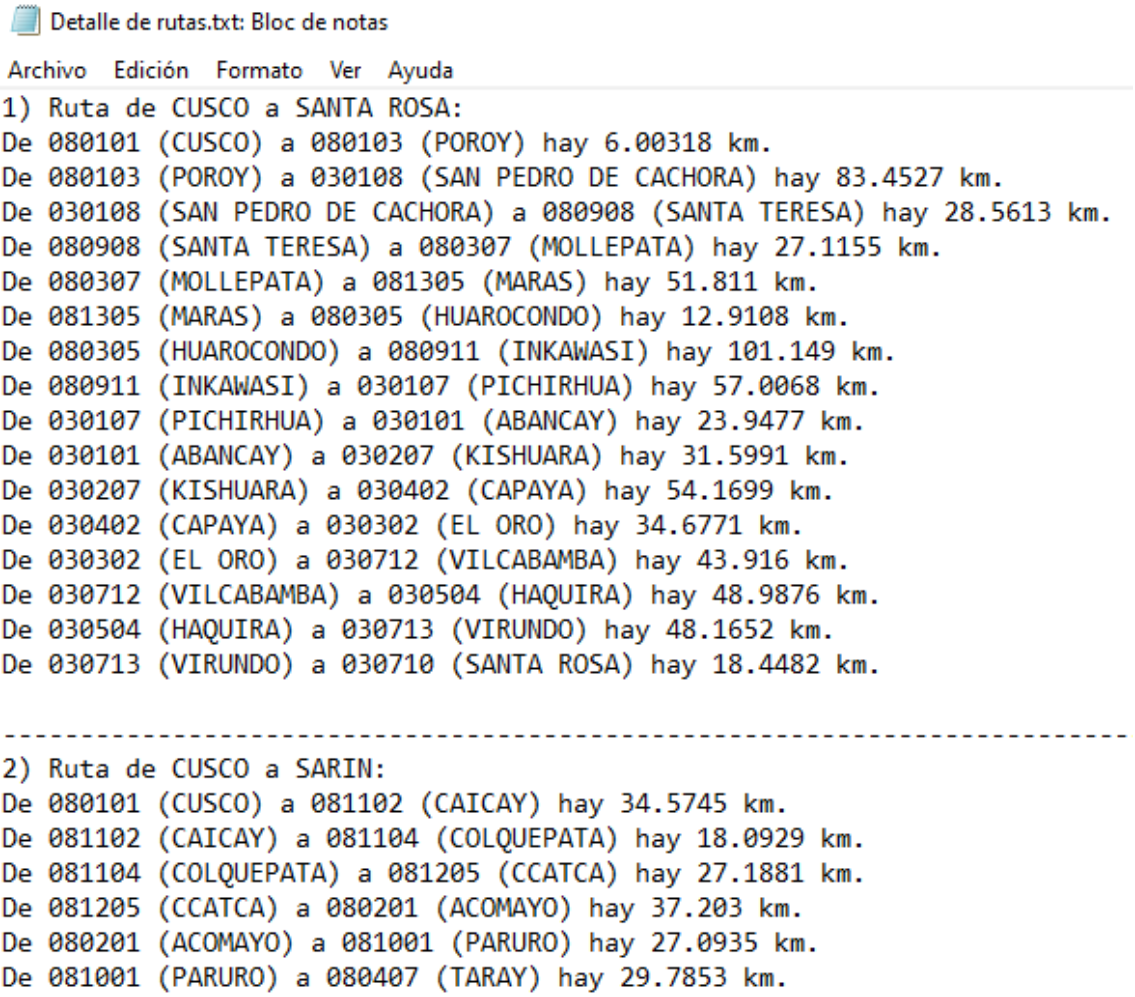
- Después de llenar todos los datos damos clic en el botón que dice “Empezar”. Luego de hacer esto, en la pantalla nos mostrará una animación mientras se termina de ejecutar el algoritmo.

- El algoritmo se demora aproximadamente 12 minutos en ejecutar, luego de este tiempo se podrá visualizar los archivos de texto que son la salida de la ejecución del algoritmo de abejas.



- En estos archivos estará la información tanto de los planes de distribución de recursos como el plan de rutas para la entrega de dichos recursos.

```
Envio de: [Cookies] (BC001)
Puntos Iniciales            HUANCAYO            SAN MIGUEL          TORRES CAUSANA
CUSCO                       148440              156640              194880
LOS OLIVOS                  121540              143340              205080

Envio de: [CannedBeans] (FR001)
Puntos Iniciales            HUANCAYO            SAN MIGUEL          TORRES CAUSANA
CUSCO                       1008                1209                1132
LOS OLIVOS                  2674                3206                3002

Envio de: [InstantNoodles] (IN001)
Puntos Iniciales            HUANCAYO            SAN MIGUEL          TORRES CAUSANA
CUSCO                       835                 955                 825
LOS OLIVOS                  2214                2530                2187

Envio de: [CannedOlives] (VE001)
Puntos Iniciales            HUANCAYO            SAN MIGUEL          TORRES CAUSANA
CUSCO                       452000              318500              29500
LOS OLIVOS                  148000              181500              70500
```



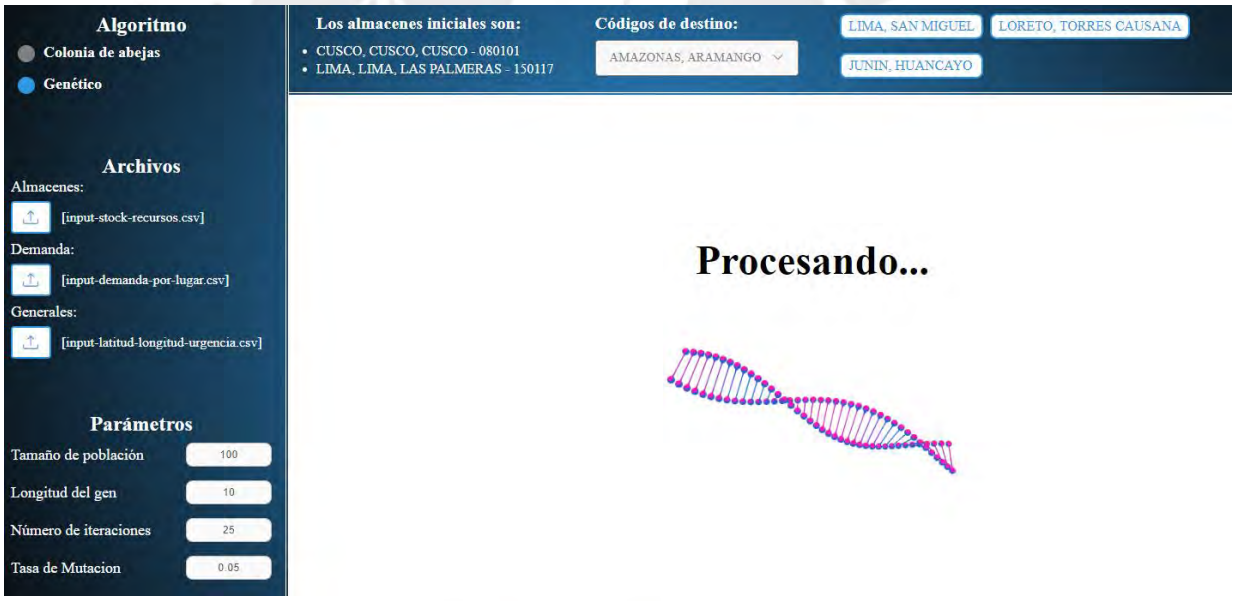
3) Prueba de integración con el algoritmo genético:

- Primero se ingresan los archivos de datos correspondientes y se selecciona el algoritmo genético.

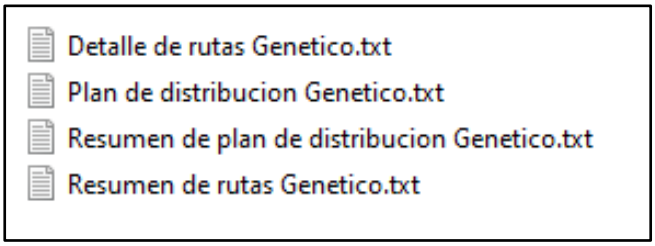
- Luego de escoger los archivos de datos, en la pantalla se visualizarán los datos de los almacenes iniciales. Seguidamente, se procede a escoger los lugares de destino.

- De igual forma se ingresan los parámetros del algoritmo genético.

- Después de llenar todos los datos damos clic en el botón que dice “Empezar”. Luego de hacer esto, en la pantalla nos mostrará una animación mientras se termina de ejecutar el algoritmo.

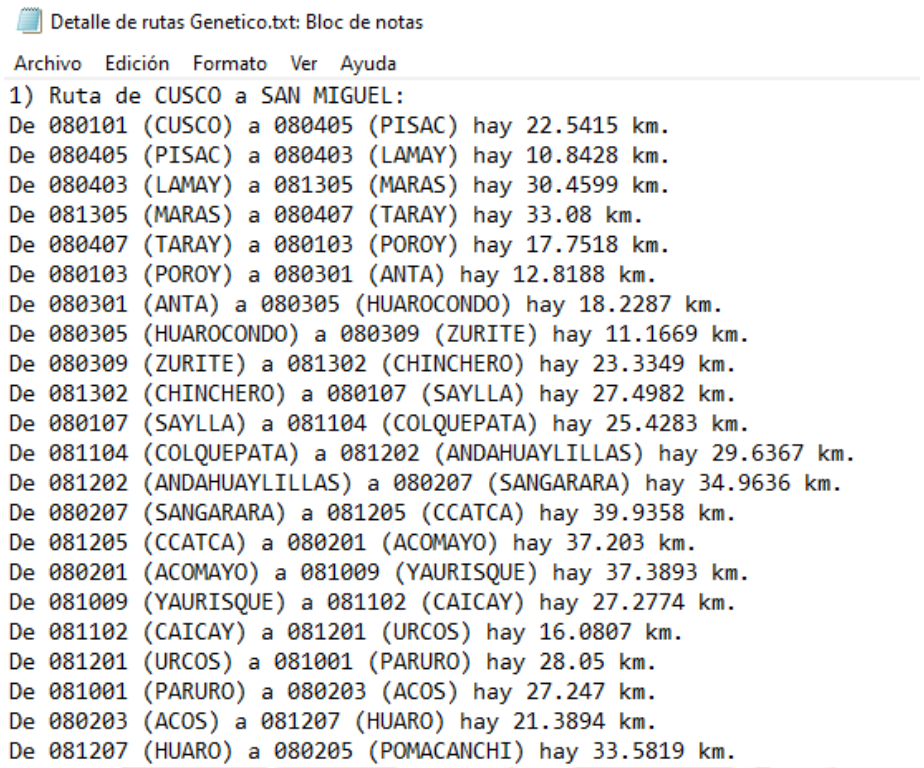


- El algoritmo se demora aproximadamente 8 minutos en ejecutar, luego de este tiempo se podrá visualizar los archivos de texto que son la salida de la ejecución del algoritmo de abejas.



- En estos archivos estará la información tanto de los planes de distribución de recursos como el plan de rutas para la entrega de dichos recursos.

```
Envio de: [Cookies] (BC001)
Puntos Iniciales            HUANCAYO          SAN MIGUEL        TORRES CAUSANA
CUSCO                       145457            149818            202985
LOS OLIVOS                  124171            148677            194819

Envio de: [CannedBeans] (FR001)
Puntos Iniciales            HUANCAYO          SAN MIGUEL        TORRES CAUSANA
CUSCO                       2674              3206              3002
LOS OLIVOS                  1008              1209              1132

Envio de: [InstantNoodles] (IN001)
Puntos Iniciales            HUANCAYO          SAN MIGUEL        TORRES CAUSANA
CUSCO                       2214              2530              2187
LOS OLIVOS                  835               955               825

Envio de: [CannedOlives] (VE001)
Puntos Iniciales            HUANCAYO          SAN MIGUEL        TORRES CAUSANA
CUSCO                       434718            324302            37587
LOS OLIVOS                  174565            189431            33468
```



Anexo 14: Validación de calibración de variables

Para observar el documento de validación de calibración de variables dirigirse al siguiente enlace:

https://docs.google.com/document/d/1931Dz0_51JlCXNn38ZX7dMwWDCmfGo1N/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true

Anexo 15: Validación de experimentación numérica

Para observar el documento de validación de la experimentación numérica en el presente trabajo dirigirse al siguiente enlace:

https://docs.google.com/document/d/1j6ruDym99P3Pui-KdGhUsjV5_f-dwV4X/edit?usp=sharing&ouid=104444307271723976007&rtpof=true&sd=true