PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Implementación de un algoritmo memético para optimizar la carga de hornos para la producción de sanitarios

Tesis Para optar por el Título de Ingeniero Informático que presenta la bachiller:

Natalia Gabriela Palomares Melgarejo

20131229

Asesor: Mg. Rony Cueva Moscoso

Lima, Setiembre de 2019

Resumen

La continua competencia entre las empresas del sector fabricación de cerámicos y sanitarios ha hecho que estas busquen mejorar su calidad y su eficiencia, en el proceso productivo, para aumentar sus ganancias y reducir pérdidas. Una de las formas en que esto se ha manifestado es mediante el uso de soluciones informáticas. Sin embargo, aunque estas se enfocan en varios aspectos como la gestión de personal, almacenes, registros de ventas, entre otros, dejan un vacío en la optimización de las etapas del proceso de fabricación en sí.

Tal es el caso de la fase de cocción, que tiene una larga duración y carece de una estrategia para la selección óptima de piezas que serán cargadas en el horno, ocasionando un cuello de botella en el proceso. La variedad de modelos a elaborar, la cantidad de piezas que los componen, los colores, así como la demanda, las restricciones de peso y volumen (de las vagonetas y hornos), dificultan la selección de piezas.

Este problema no solo se presenta en el sector fabricación de sanitarios, sino también en otros, donde se manufacturan y ensamblan productos compuestos por varias partes, razón por la cual se han realizado varias investigaciones para desarrollar soluciones basadas en algoritmos que generen buenos resultados en tiempos razonables.

La clase de algoritmos más usados en estos casos son los metaheurísticos y, dentro de esta categoría, el algoritmo genético debido a su simplicidad. Sin embargo, en investigaciones recientes se ha visto que los algoritmos meméticos generan buenas soluciones en un número menor de evaluaciones y con mejor calidad.

Tomando en cuenta lo mencionado anteriormente, este proyecto de fin de carrera tuvo como objetivo: diseñar e implementar un algoritmo memético que genere una selección de piezas priorizando aquellas que aprovechen la capacidad de peso y volumen de hornos y vagonetas, considerando la demanda de los sets de productos. Este algoritmo luego fue calibrado para mejorarlo y finalmente se lo comparó con el algoritmo genético para determinar cuál de ellos es el mejor para este tipo de problemas.

Tema FCI

FACULTAD DE CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS

PARA OPTAR	: Título profesional de Ingeniero Informático
TEMA	: Implementación de un algoritmo memético para optimizar la carga de hornos para la producción de sanitarios.
ÁREA	: Ciencias de la computación
ASESOR	: Magíster Rony Cueva Moscoso.
ALUMNO	: Natalia Gabriela Palomares Melgarejo – 20131229
FECHA	: 02 de agosto del 2019
MÁXIMO	: 100 páginas

DESCRIPCIÓN

La continua competencia entre las empresas del sector fabricación de cerámicos y sanitarios ha hecho que estas busquen mejorar su calidad y su eficiencia en el proceso productivo, para aumentar sus ganancias y reducir pérdidas. Una de las formas en que esto se ha manifestado es mediante el uso de soluciones informáticas. Sin embargo, aunque estas se enfocan en varios aspectos como la gestión de personal, almacenes, registros de ventas, entre otros, dejan un vacío en la optimización de las etapas del proceso de fabricación en sí.

Tal es el caso de la fase de cocción, que tiene una larga duración y carece de una estrategia para la selección óptima de piezas que serán cargadas en el horno, ocasionando un cuello de botella en el proceso. La variedad de modelos a elaborar, la cantidad de piezas que los componen, los colores, así como la demanda, las restricciones de peso y volumen (de las vagonetas y hornos), dificultan la selección de piezas.

Este problema no solo se presenta en el sector fabricación de sanitarios, sino también en otros donde se manufacturan y ensamblan productos compuestos por varias partes, razón por la cual se han realizado varias investigaciones para desarrollar soluciones basadas en algoritmos que generen buenos resultados en tiempos razonables.

La clase de algoritmos más usados en estos casos son los metaheurísticos y, dentro de esta categoría, el algoritmo genético debido a su simplicidad. Sin embargo, en

investigaciones recientes se ha visto que los algoritmos meméticos generan buenas soluciones en un número menor de evaluaciones y con mejor calidad.

Tomando en cuenta lo mencionado anteriormente, este proyecto de fin de carrera tuvo como objetivo: diseñar e implementar un algoritmo memético que genere una selección de piezas priorizando aquellas que aprovechen la capacidad de peso y volumen de hornos y vagonetas, considerando la demanda de los sets de productos. Este algoritmo luego fue calibrado para mejorarlo y finalmente se lo comparó con el algoritmo genético para determinar cuál de ellos es el mejor para este tipo de problemas.

OBJETIVO GENERAL

Implementar un algoritmo memético para optimizar la carga de hornos en la producción de sanitarios.

OBJETIVOS ESPECÍFICOS

Los objetivos específicos son:

O 1. Definir la función objetivo a ser usada en los algoritmos genético y memético.

O 2. Adaptar el algoritmo genético para elaborar una alternativa de solución al problema de carga de hornos en la producción de sanitarios.

O 3. Diseñar un algoritmo memético como alternativa de solución al problema de carga de hornos en la producción de sanitarios.

O 4. Implementar los algoritmos genético y memético.

O 5. Realizar la experimentación numérica para comparar el desempeño de los algoritmos genético y memético.

NOTA

Complete el formato y solicite el visto bueno de su asesor. Tenga en cuenta que la Facultad verificará que el tema de tesis propuesto, cumpla los siguientes requisitos:

1. Usted debe adjuntar un archivo ZIP conteniendo el tema de tesis en Word y en formato PDF con el visto bueno del asesor.
2. Usted no debe contar con un Tema de tesis asignado anteriormente. De darse el caso, deberá efectuar el trámite de cambio del tema de tesis en la Facultad.
3. Usted debe encontrarse matriculado o haber aprobado el segundo curso de Tesis de su especialidad.
4. En caso de que el tema de tesis mencione a una organización, deberá adjuntar la autorización del representante legal de dicha organización.

En caso de alguna consulta adicional, puede contactarnos a la cuenta: titulacion-fci@pucp.edu.pe

Dedicatoria

A mi familia, por su gran apoyo todos estos años.

A mis amigos, por estar siempre presentes.

A Kairo, por su apoyo moral.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades

1.1 Problemática

En los últimos años, las empresas están buscando constantes mejoras en la gestión, tecnología y canales de comercialización, entre otros; para reducir costos, aumentar eficiencia y ganancias (Misthal, Mueller, Bono, & Pillsbury, s. f.). El sector fabricación de cerámicos y sanitarios no es la excepción y las empresas han ido automatizando, cada vez más, las etapas de su proceso de producción (Ceramic Industry, 2015).

En el Perú, las empresas de este sector incrementaron el uso de máquinas automáticas o semiautomáticas para mejorar la calidad de sus productos (Porras, 2018) y así competir con empresas de Ecuador y China que vinieron a comercializar en el país atraídas por el crecimiento económico regional (Azurín, Timaná, & Pérez, 2018).

Además, ante el aumento del volumen de datos necesarios para gestionar la cadena de valor, están usando soluciones informáticas como ERPs adaptados a las características del sector como SAP, Roadmap, Microsoft Dynamics 365 Business Central, etc. Sin embargo, tienden a estar más orientados al manejo de personal, almacenes, clientes, abastecimiento de suministros, ventas y registro de la producción (Corporate Munim, s. f.) (Microsoft, 2016), dejando un vacío en la optimización del proceso de producción.

Dentro del proceso de fabricación de sanitarios (ver Figura *1*) hay varias etapas que se podrían optimizar, como el caso de la Cocción, que es la fase más larga (toma de 8 a 12 horas) y en la que se somete a las piezas a altas temperaturas en los hornos túneles hasta alcanzar el vidriado final (Díaz, 2004).

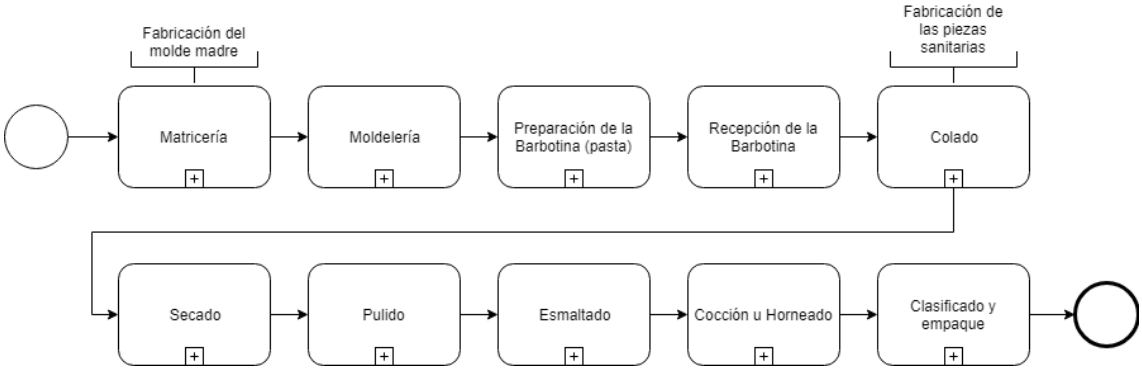

Figura 1 Proceso de fabricación de sanitarios. Fuente: *Elaboración propia.*

Un paso crucial en la etapa de Cocción es la carga de vagonetas, en la que se determina qué piezas (tazas, pedestales, tanques, etc.) se colocarán en ellas (ver Figura *2*)

respetando las restricciones de volumen y peso para no afectar la temperatura del horno túnel.

Figura 2 Vagoneta de carga para los hornos túnel. Fuente: *(SACMI, 2005)*

La falta de una estrategia para la selección óptima de las piezas que irán en las vagonetas y la larga duración de esta etapa generan un cuello de botella, convirtiéndose en un problema dentro del proceso productivo. Por lo que, a veces, se escogen piezas de un solo modelo o color y la producción de los otros modelos se retrasa; en otras ocasiones, se escoge un grupo variado de piezas de diferentes modelos y colores, pero que no conforman ningún set, retrasando las etapas posteriores de ensamblado y empaquetado. Esta situación ocasiona demoras en la producción, causando que la maquinaria y la mano de obra de las etapas posteriores estén ociosas, sin poder realizar su trabajo de ensamblaje y embalaje; esto hace que no se pueda vender y que en el almacén se disponga de pocos modelos completos y gran cantidad de sets incompletos o piezas sueltas, causando que no se pueda atender oportunamente a los clientes.

Este es un problema de optimización combinatoria, conocido como el problema de la mochila (Knapsack problem); el cual es altamente complejo y se considera NP-difícil; consiste en seleccionar un conjunto de objetos que cumplan las restricciones y generen el mayor beneficio (Dorta, León, Rodríguez, Rodríguez, & Rojas, 2003). Otras industrias, como las fundiciones y fábricas, que producen una gran variedad de productos compuestos por varias partes que luego serán ensambladas, también encuentran dificultades similares por lo que han surgido diferentes softwares comerciales (ej: CPLEX) para resolver problemas de optimización; sin embargo, estos aún no se encuentran en la capacidad de resolver problemas complejos o dar soluciones satisfactorias (Duda & Stawowy, 2013). Por ello, se han realizado investigaciones y entre las propuestas de solución que estas plantean se encuentra el uso de metaheurísticas

que tienen la ventaja de no ser específicas a un problema y dar buenas soluciones en un tiempo razonable (Blum & Roli, 2003).

Entre las metaheurísticas propuestas, el método más utilizado y con mejores resultados es el algoritmo genético (Liu, Pan, & Chai, 2015) (Duda & Stawowy, 2013), debido a su simplicidad, perspectiva global y su procesamiento paralelo inherente (Deb, 2004).

Otro tipo de algoritmo que se ha utilizado exitosamente en problemas similares es el memético, que combina búsqueda local con operadores genéticos (Alba & Dorronsoro, 2005) y compensan las habilidades de exploración de los algoritmos evolutivos y las capacidades de explotación de la búsqueda local (Krasnogor & Smith, 2005).

El algoritmo memético es un método que no ha sido tan usado como el primero, pero ha generado buenos resultados (Chen & Hao, 2016), porque a diferencia de otros algoritmos evolutivos requiere un menor número de evaluaciones para encontrar los óptimos y hallar soluciones de mejor calidad (Baesler & Palma, 2014).

Por lo anteriormente expuesto, en este proyecto de fin de carrera, se propone una solución basada en el algoritmo memético para resolver el problema de la carga de hornos en la producción de sanitarios. Esta solución, será luego comparada con otra basada en el algoritmo genético.

1.2 Objetivos

1.2.1 Objetivo general

Implementar un algoritmo memético para optimizar la carga de hornos en la producción de sanitarios.

1.2.2 Objetivos específicos

- O 1. Definir la función objetivo a ser usada en los algoritmos genético y memético.
- O 2. Adaptar el algoritmo genético para elaborar una alternativa de solución al problema de carga de hornos en la producción de sanitarios.
- O 3. Diseñar un algoritmo memético como alternativa de solución al problema de carga de hornos en la producción de sanitarios.
- O 4. Implementar los algoritmos genético y memético.
- O 5. Realizar la experimentación numérica para comparar el desempeño de los algoritmos genético y memético.

1.2.3 Resultados esperados

R 1. Definición de variables, función objetivo y restricciones del problema (O1).

R 2. Estructura de datos que utilizará el algoritmo genético (O2): consiste en la definición de la estructura del cromosoma y su representación.

R 3. Algoritmo genético adaptado (O2).

R 4. Estructura de datos que utilizará el algoritmo memético (O3): consiste en la definición de la estructura y representación de las soluciones.

R 5. Algoritmo memético diseñado (O3): consistirá en el pseudocódigo del algoritmo, que incluirá el método de recombinación, mutación, la selección de los operadores y el número de iteraciones que tendrá la búsqueda local.

R 6. Interfaz para la carga de los datos de prueba (O4): permitirá la carga de los archivos que contengan la información de los productos a fabricar y los parámetros de capacidad del horno.

R 7. Algoritmo genético implementado (O4).

R 8. Algoritmo memético implementado (O4).

R 9. Interfaz para mostrar los resultados obtenidos de los dos algoritmos (O4): mostrará para cada algoritmo el tiempo de ejecución, la lista con la propuesta de los objetos a cargar y la cuantificación del beneficio que traería esta propuesta.

R 10. Informe de la experimentación numérica (O5): en este documento se comparará y cuantificará el desempeño de los dos algoritmos.

1.2.4 Mapeo de objetivos, resultados y verificación

Objetivo 1: Definir la función objetivo a ser usada en los algoritmos genético y memético.		
Resultado	**Meta física**	**Medio de verificación**
1) Definición de variables, función objetivo y restricciones que tendrá el problema.	Documento que contenga: - Función matemática que utilizarán los algoritmos. - Explicación de cada variable que conforma esta función.	Revisión por un equipo de especialistas en la fabricación de productos sanitarios.

	- Explicación y justificación de las restricciones del problema.	
Objetivo 2: Adaptar el algoritmo genético como alternativa de solución al problema de carga de hornos en la producción de sanitarios.		
Resultado	**Meta física**	**Medio de verificación**
1) Estructura de datos que utilizará el algoritmo genético.	Documento que contenga: - Estructura y representación del cromosoma. - Justificación de la estructura planteada.	
2) Algoritmo genético adaptado	Documento que contenga: - Pseudocódigo del algoritmo genético.	Validación del diseño por un experto.
Objetivo 3: Diseñar un algoritmo memético como alternativa de solución al problema de carga de hornos en la producción de sanitarios.		
Resultado	**Meta física**	**Medio de verificación**
1) Estructura de datos que utilizará el algoritmo memético.	Documento que contenga: - Estructura y representación de las soluciones. - Justificación de la estructura planteada.	
2) Algoritmo memético diseñado	Documento que contenga: - Pseudocódigo del algoritmo memético.	Validación del diseño por un experto.

	- Pseudocódigo de los métodos de recombinación y mutación. - Selección de los operadores y número de iteraciones de la búsqueda local.	
Objetivo 4: Implementar los algoritmos genético y memético.		
Resultado	**Meta física**	**Medio de verificación**
1) Interfaz para la carga de los datos de prueba.	Software	- Pruebas de funcionamiento con archivos de prueba para comprobar: si se realiza la carga de datos y si los datos cargados son válidos.
2) Algoritmo genético implementado.	Software	- Pruebas unitarias para asegurar que la solución obtenida sea válida (cumpla con las restricciones). - Validación por un equipo de especialistas en la fabricación de productos sanitarios.
3) Algoritmo memético implementado.	Software	- Pruebas unitarias para asegurar que la solución obtenida sea válida (cumpla con las

		restricciones planteadas). - Validación por un equipo de especialistas en la fabricación de productos sanitarios.
4) Interfaz para mostrar los resultados obtenidos de los dos algoritmos.	Software	- Pruebas de funcionamiento para comprobar que se muestren las soluciones planteadas por los dos algoritmos.
Objetivo 5: Realizar la experimentación numérica para comparar el desempeño de los algoritmos Genético y Memético.		
Resultado	**Meta física**	**Medio de verificación**
1) Informe de la experimentación numérica.	Documento que contenga: - Descripción de las pruebas a realizar en la experimentación numérica. - Resultados obtenidos. - Interpretación de los resultados.	- Comparación estadística de los resultados obtenidos por los algoritmos para determinar cuál de ellos genera mejores resultados.

1.3 Herramientas y Métodos

En esta sección se describirán las herramientas, metodologías y métodos que se emplearán durante el proyecto de investigación.

1.3.1 Mapeo

Resultados esperados	Herramientas, métodos y metodologías
R 1. Definición de variables, función objetivo y restricciones del problema.	- Scrum - Netbeans
R 2. Estructura de datos que utilizará el algoritmo genético.	- Scrum - Netbeans
R 3. Algoritmo genético adaptado.	- Scrum - Netbeans
R 4. Estructura de datos que utilizará el algoritmo memético.	- Scrum - Netbeans
R 5. Algoritmo memético diseñado.	- Scrum - Netbeans
R 6. Interfaz para la carga de los datos de prueba.	- Netbeans - Java - Scrum
R 7. Algoritmo genético implementado.	- Netbeans - Java - Scrum
R 8. Algoritmo memético implementado.	- Netbeans - Java - Scrum
R 9. Interfaz para mostrar los resultados obtenidos de los dos algoritmos.	- Netbeans - Java - Scrum
R 10. Informe de la experimentación numérica.	- ANOVA (Prueba Kolmogorov-Smirnov, F de Fisher y Z) - RStudio

1.3.2 Herramientas

1.3.2.1 Netbeans

Es un entorno de desarrollo integrado (IDE) gratuito, de código abierto y que puede ser instalado en todos los sistemas operativos que soporten Java como Windows, Linux y Mac OS X. Este IDE permite desarrollar aplicaciones web y móviles en Java, JavaScript, HTML5, C/C++, PHP, entre otros (NetBeans, s. f.).

Se utilizará para el desarrollo de los módulos del algoritmo genético y memético.

1.3.2.2 Java

Es un lenguaje de programación derivado del lenguaje C, orientado a objetos, de uso general, concurrente y fuertemente tipado (Oracle, 2018). También se caracteriza por su portabilidad, seguridad y por ser multiplataforma (Oracle, s. f.). Se empleará en el desarrollo de los módulos que incorporarán los algoritmos implementados.

1.3.3 Métodos y metodologías

1.3.3.1 Análisis de varianza (ANOVA)

Es un método para evaluar la significancia estadística de la diferencia entre las medias de varios grupos. Tiene como requerimientos:

- Las muestras deben ser independientes.
- Los grupos deben tener la misma varianza.
- Las muestras deben tener una distribución normal (Moore, 2005).

1.3.3.1.1 Prueba Kolmogorov-Smirnov

La prueba de Kolmogorov-Smirnov para una muestra es un procedimiento de bondad de ajuste que mide el grado de concordancia entre la distribución de un conjunto de datos y una distribución teórica específica (García, González, & Jornet, 2010).

Se utilizará para comprobar que las muestras siguen la distribución normal.

1.3.3.1.2 Prueba F de Fisher

Esta prueba permite descubrir si las muestras provienen de poblaciones con varianzas significativamente diferentes (Crawley, 2005).

1.3.3.1.3 Prueba Z

Esta prueba se usa para determinar la diferencia entre las medias poblacionales y se basa en la diferencia entre las medias de las muestras. Para ello estas muestras deben ser aleatorias y seleccionadas independientemente de poblaciones que están distribuidas de forma normal (Berenson & Levine, 2006). Se empleará esta prueba, en lugar de la prueba T-Student, porque los tamaños de las muestras son mayores a 30.

1.3.3.2 Scrum

Es una metodología ágil de desarrollo de software muy conocida. Según Kniberg y Skarin, se caracteriza por:

- Dividir la organización en equipos pequeños e interdisciplinarios.
- Asignar roles a cada participante del proyecto: dueño de producto (establece la visión del producto), equipo (implementa el producto) y Scrum Master (lidera el proceso).
- Requerir de la participación y retroalimentación constante del cliente.
- Dividir la duración total del proyecto en iteraciones cortas.
- Priorizar regularmente los requerimientos en colaboración con el cliente.
- Dividir el trabajo en entregables pequeños que serán presentados al final de cada iteración.
- Priorizar el código funcionando sobre la documentación (Kniberg & Skarin, 2010).

Se usará esta metodología durante el desarrollo de la aplicación, porque enfatiza solo la elaboración de la documentación necesaria y subdivide el proyecto en iteraciones cortas; adaptándose con la presentación de entregables del curso. No se aplicarán todos los aspectos de Scrum como la creación de equipos y la asignación de roles debido a que no se cuenta con un equipo para el desarrollo sino solo con una persona.

1.4 Justificación y viabilidad

1.4.1 Justificación del proyecto de fin de carrera

Este proyecto beneficiará principalmente a las empresas del sector fabricación de sanitarios porque ofrecerá una alternativa de solución efectiva, adaptada al sector, que optimizará la carga de hornos y reducirá las largas jornadas para armar los sets.

Con esta alternativa se podrán reducir: las ventas perdidas por demoras en la etapa de cocción, los tiempos de abastecimiento de productos y sets, las multas por retrasos y los costos de almacenamiento de partes y sets incompletos. También permitirá un mayor aprovechamiento de la capacidad productiva en las etapas de cocción y empaque.

Finalmente, el uso de esta herramienta permitirá que la carga de vagonetas ya no dependa de los trabajadores más experimentados quienes en la actualidad son los que, basados en su habilidad y criterio, deciden cómo se debe hacer la carga. De esta manera, se aliviará la carga laboral de estos trabajadores y se podrán enfocar en otras tareas en lugar de preocuparse por cómo realizar la carga de hornos.

1.4.2 Análisis de la viabilidad del proyecto de fin de carrera

1.4.2.1 Viabilidad Técnica

Se cuenta con los conocimientos necesarios para llevar a cabo el proyecto ya que en los cursos desarrollados durante la carrera se ha utilizado Java y se tiene experiencia programando en este lenguaje.

Se optó por implementar los algoritmos genético y memético, en comparación a los otros métodos vistos en el estado del arte, no solo por el buen desempeño que tienen al resolver problemas de optimización, sino también porque se tiene mayor conocimiento y familiaridad con estos temas al haber sido tratados en cursos anteriores y porque se cuenta con la facilidad de tener bastante documentación al respecto.

1.4.2.2 Viabilidad Temporal

A continuación, se muestra el cronograma que se siguió durante el desarrollo del proyecto:

Tarea	**Inicio**	**Fin**	**Duración**
Elaboración de la función objetivo	09/07/18	15/07/18	7 días
Elaboración de la estructura de datos del algoritmo genético	16/07/18	22/07/18	7 días
Elaboración del pseudocódigo y calibración del algoritmo genético	23/07/18	05/08/18	14 días

Elaboración de la estructura de datos del algoritmo memético	06/08/18	12/08/18	7 días
Elaboración del pseudocódigo y calibración del algoritmo memético	13/08/18	26/08/18	14 días
Correcciones de los algoritmos	27/08/18	03/09/18	8 días
Desarrollo de la interfaz para la carga de los datos de prueba	04/09/18	08/09/18	5 días
Implementación del algoritmo genético	09/09/18	15/09/18	7 días
Desarrollo de las pruebas del módulo del algoritmo genético y realización de correcciones	16/09/18	17/09/18	2 días
Implementación del algoritmo memético	18/09/18	24/09/18	7 días
Desarrollo de las pruebas del módulo del algoritmo memético y realización de correcciones	25/09/18	26/09/18	2 días
Desarrollo de la interfaz que muestra los resultados obtenidos	27/09/18	01/10/18	5 días
Ejecución y análisis de la experimentación numérica	02/10/18	07/10/18	6 días
Correcciones finales	08/10/18	21/10/18	14 días

1.4.2.3 Viabilidad Económica

Para el desarrollo del proyecto no se requerirá de una gran inversión económica debido a que se utilizarán herramientas disponibles de manera gratuita (como Netbeans) o con las que ya se cuentan, como es el caso de Microsoft Excel y Word.

1.5 Alcance, Limitaciones y Riesgos

1.5.1 Alcance

El presente proyecto de investigación plantea un algoritmo metaheurístico como propuesta de solución al problema de la selección de piezas para la carga de hornos. Este algoritmo seleccionará qué componentes de productos se cargarán en las vagonetas para aprovechar al máximo los hornos, sin exceder el peso máximo que estos pueden cargar, y reducir la cantidad de sets incompletos.

Para la selección de piezas solo se tomará en cuenta las siguientes características:

- Set al que pertenece, siendo un set el conjunto de productos del mismo color y modelo que se venden como un todo.
- Prioridad: valor que será determinado por la importancia del cliente, fecha de entrega de los pedidos y por la cantidad requerida.
- Peso: es el peso de la pieza en kilogramos.
- Volumen: es el volumen de la pieza en cm^3.
- Dimensiones de la pieza en cm.

Para elaborar esta propuesta de solución se definirá la función objetivo y las estructuras de datos que usarán los algoritmos, genético y memético, métodos seleccionados por ser los más utilizados para resolver este tipo de problemas y porque generan soluciones de mejor calidad. Posteriormente, se procederá a implementarlos y compararlos en la experimentación numérica, demostrando y cuantificando cuál es el mejor.

1.5.2 Limitaciones

El tiempo de ejecución que se obtendrá de la prueba de los algoritmos variará según las características del hardware de la computadora en la que se ejecutan estas pruebas.

El tiempo máximo permitido para la ejecución de los algoritmos con el que se trabajará será determinado por el proceso de carga de hornos en la etapa de cocción.

1.5.3 Riesgos

Riesgo identificado	Impacto en el proyecto	Medidas correctivas para mitigar
Demora en la obtención de información sobre	Bajo: retraso en el desarrollo del proyecto.	Generar los datos de manera aleatoria.

modelos, piezas, productos, sets, etc.		
Las pruebas de experimentación numérica muestran que los algoritmos tienen un desempeño bastante pobre.	Alto: posible fracaso del proyecto de tesis.	Al iniciar el desarrollo de los algoritmos, se harán pruebas periódicas para conocer el desempeño de los algoritmos y detectar problemas en una etapa temprana.

Capítulo 2. Marco Legal/Regulatorio/Conceptual/otros

2.1 Introducción

En este capítulo se describen ciertos conceptos necesarios para comprender, a mayor profundidad, la problemática y la solución propuesta. Se iniciará con el desarrollo de conceptos relacionados al proceso productivo de inodoros y se describirá brevemente las etapas que componen el proceso. Finalmente, se explicará los términos relacionados a la solución propuesta donde se incluirá una definición del tipo de problema a resolver y los algoritmos a utilizar.

2.2 Set

Obbink define set como un conjunto conformado por productos que comparten ciertas características en común (Obbink, 2005). Para nuestro caso, el set es la agrupación de productos de un mismo modelo que se venden juntos. En la Figura *3* se muestra dos ejemplos de sets: el primero (set A) se compone por un sanitario, un lavatorio y accesorios del modelo 123 en color blanco y el segundo (set B) se compone solo por un sanitario y lavatorio del modelo 456 en color azul.

Figura 3 Ejemplos de sets. Fuente: *Elaboración propia.*

2.3 Horno túnel

Es un canal rectilíneo, constituido por paredes verticales, una cubierta y rieles para que pueda desplazarse la carga (ver Figura *4*), constituyendo un equipo con carga móvil y fuente de calentamiento fija (Gómez Gutiérrez, C., 2010).

Estos hornos están orientados para la producción continua y cargas más homogéneas, por lo que se emplean en la fabricación de distintos productos como: arcilla roja, porcelana, loza sanitaria, gres y decoración (CERAMIFOR Kilns & Equipment, s. f.).

Figura 4 Horno túnel. Fuente: *(Thermal Engineering, s. f.)*

2.4 Vagoneta

Es un vagón pequeño y descubierto (ASALE, s. f.), utilizado para el transporte de las piezas sanitarias a través de los hornos túnel.

Las dimensiones y el material de las vagonetas que se emplearán dependen de las características del proceso (duración y temperatura máxima de cocción) y del horno en el que se utilizarán (dimensiones), esto asegura un buen proceso de cocción (Sarmiento, Vladimir, & Reinoso Avecillas, 2012).

Figura 5 Vagoneta. Fuente: *(Comain, s. f.)*

2.5 Proceso de fabricación de sanitarios y accesorios

Las etapas del proceso de fabricación no han cambiado en los últimos años por lo que la siguiente descripción realizada por Miguel Díaz (2004) sigue vigente:

- Matricería: etapa en la que se fabrica la matriz (molde madre).
- Moldelería: consiste en vaciar yeso en la matriz para generar los moldes que se utilizan en la etapa de Colaje.
- Preparación de la Barbotina: mezcla de materias primas para la preparación de una pasta llamada Barbotina.

- Recepción de Barbotina y Ajuste: comprende la recepción y almacenamiento de la Barbotina para su ajuste a los niveles de viscosidad y densidad necesarios.
- Colado: etapa en la que se fabrican las piezas sanitarias, para esto se vierte la Barbotina en los moldes que se elaboraron en las etapas anteriores.
- Secado: utiliza cámaras de secado en las que se regula la temperatura y la humedad para evitar la aparición de grietas.
- Pulido: para eliminar defectos superficiales en las piezas. En esta etapa también se realizan los cortes para los orificios de descarga y entrada.
- Esmaltado: se aplica esmalte para dar el acabado final, otorgando color a la pieza.
- Cocción u Horneado: las piezas son cargadas en las vagonetas para ser sometidas a altas temperaturas hasta llegar al vidriado final. Esta etapa dura de 8 a 12 horas.
- Clasificado y empaque: consiste en la inspección de las piezas para verificar su calidad. Si la pieza cumple con los estándares de calidad, ingresa a los almacenes para su venta y despacho. Aquellas que no cumplan, serán trituradas (Díaz, 2004).

2.6 Carga del horno

Consiste en colocar los objetos a cocer, en placas o sobre estantes ubicados en las vagonetas, formando múltiples hileras verticales (Avgustinik, 1983). Si la cerámica no es vidriada y no se cuece a una temperatura en que empiece la deformación, las piezas se apilan unas sobre otras; sino, se dispondrán para que no se peguen (Rhodes, 2004).

Las placas y los estantes en los que se colocan las piezas permiten que la carga del horno se realice de manera eficaz y, generalmente, son elaborados con materiales que facilitan la conductibilidad térmica (Rhodes, 2004). Una vez cargadas las vagonetas, son llevadas a los hornos túnel.

2.7 Optimización Combinatoria

Forma parte de la categoría de problemas de optimización, en los que se busca la mejor configuración de un set de variables de manera que se alcance las metas deseadas.

En el caso de los problemas de optimización combinatoria se busca un objeto (típicamente un entero, un subconjunto, una permutación o la estructura de un grafo) dentro de un conjunto finito (Papadimitriou & Steiglitz, 1998).

Se puede definir por:

- Un set de variables $X = \{x_1, \dots, x_n\}$
- Dominios variables $D_1, \dots, D_n$

- Restricciones de las variables
- Función objetivo $f: D_1 \times \ldots \times D_n \rightarrow \mathbb{R}^+$ que debe ser minimizada.

El set con todas las posibles asignaciones es:

$$S = \{ s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i, s\ satisface\ todas\ las\ restricciones\}$$

S, llamado espacio de búsqueda, tiene como elementos candidatos de solución y para resolver el problema de optimización combinatoria se debe encontrar la solución óptima s^* que tenga el menor valor de función objetivo.

Dentro de esta categoría se encuentran problemas típicos como: el del vendedor ambulante, el de asignación cuadrática, el de programación de horarios, entre otros (Blum & Roli, 2003).

2.8 NP-hard o NP-difícil

Un problema de búsqueda X es NP-Difícil si para algún problema NP-Completo Y hay una reducción de Turing en tiempo polinomial de Y a X (Meurant, 2014).

Esta categoría está compuesta por problemas que son tan complejos como los problemas NP más difíciles (también conocidos como NP-Completos) y en el peor de los casos no podrán ser resueltos por algoritmos completos en un tiempo polinomial. Para resolver estos problemas los métodos completos requieren de un tiempo exponencial en el peor escenario, por lo que se prefiere el uso de los métodos aproximados (que dan buenas soluciones en un menor tiempo) (Blum & Roli, 2003).

2.9 Problema de la mochila

Es un problema de Optimización Combinatoria de tipo NP-hard, es decir, un problema no probabilístico ya que existe una combinación exponencial de instancias que, en su totalidad, no pueden ser resueltas (Fuentes-Penna, Vélez-Díaz, Moreno-Gutiérrez, Martínez-Cervantes, & Sánchez-Muñoz, 2015).

Dorta et al, definen así el Problema de la mochila: se dispone de un contenedor (mochila) de capacidad C y de un conjunto de n objetos, donde los objetos son indivisibles. Para los autores, el problema consiste en:

> Averiguar qué objetos colocar en la mochila sin exceder su capacidad total.
>
> $$Sujeto\ a \sum_{j=1}^{n} w_j x_j \leq C$$

Obteniendo el máximo beneficio:

$$Maximizar\ \sum_{j=1}^{n} b_j x_j\ ,\ x_j \in \{0,1\}, j = 1,2,\dots,n.$$

Donde:

x_j: Variables de decisión

b_j: Beneficio del objeto j

w_j: Peso w del objeto j

C: Capacidad total del contenedor (mochila)

n: número de objetos (Dorta, León, Rodríguez, Rodríguez, & Rojas, 2003)

2.10 Algoritmo Metaheurístico

Algoritmo aproximado que trata de combinar métodos heurísticos básicos en marcos de alto nivel, dirigidos a explorar de manera eficiente y efectiva un espacio de búsqueda.

Es un proceso de generación iterativo que guía una heurística subordinada mediante la combinación inteligente de diferentes conceptos para explorar y explotar el espacio de búsqueda, usa estrategias de aprendizaje para estructurar la información en orden y encontrar soluciones eficientes cercanas a las óptimas (Osman & Laporte, 1996).

Propiedades de las metaheurísticas:

- Son estrategias que guían el proceso de búsqueda.
- Tienen como propósito explorar de manera eficiente el espacio de búsqueda para encontrar una solución cercana a la óptima.
- Son aproximadas y usualmente no determinísticas.
- No son específicas a un problema (Blum & Roli, 2003).

Dentro de esta categoría se encuentran: el Algoritmo genético (GA), Optimización basada en Colonia de Hormigas (ACO) y Algoritmo memético, entre otros.

2.11 Algoritmo genético

Fue desarrollado por Holland y está inspirado en la teoría de la evolución de Charles Darwin (Holland, 1992).

El algoritmo genético comienza su búsqueda con un set aleatorio de soluciones (P(t), población), cada una de estas soluciones es evaluada por la función objetivo y se le asigna un valor fitness, el cual indica que tan adecuada es la solución (Deb, 2004).

Luego, la población de soluciones es modificada, formando una nueva, mediante la aplicación de los siguientes operadores:

- Mutación: crea nuevos individuos aplicando cambios a un individuo ya existente.
- Cruce: crea nuevos individuos combinando partes de dos individuos ya existentes.

Los nuevos individuos, C(t), llamados descendientes, son evaluados y una nueva población, P(t+1), es formada mediante la selección de los más aptos de la población padre, P(t), y de los descendientes (Gen & Cheng, 2000).

El algoritmo genético seguirá trabajando de manera iterativa aplicando estos operadores a cada generación hasta satisfacer la condición de finalización (Deb, 2004).

A continuación, se muestra la estructura de este algoritmo:

```
Algoritmo_Genético
Inicio
 1    t ← 0
 2    Inicializar P(t) //Set de soluciones potenciales del problema
 3    Evaluar P(t) //Evalúa cada solución con la función objetivo para obtener el valor fitness
 4    Mientras (no se cumpla la condición de finalización) hacer
 5        Recombinar P(t) para producir C(t)
 6        Evaluar C(t)
 7        Seleccionar P(t+1) de P(t) y C(t)
 8        t ← t+1
 9    Fin Mientras
Fin
```

Figura 6 Estructura básica del algoritmo genético. Fuente: *(Michalewicz, 2013)*

Tiene 5 componentes básicos:

- Una representación genética de las soluciones del problema.
- Una forma de crear una población inicial de soluciones.
- Una función de evaluación de soluciones en términos de su aptitud.
- Operadores genéticos que alteran la composición genética de los hijos durante la reproducción.
- Valores para los parámetros (tamaño de la población, probabilidad de aplicar operadores genéticos, etc.). (Michalewicz, 2013)

El algoritmo genético se ha aplicado exitosamente a una gran variedad de problemas por su simplicidad, perspectiva global y procesamiento paralelo inherente (Deb, 2004).

2.12 Algoritmo memético

La búsqueda memética propone la combinación de los algoritmos evolutivos con la búsqueda local. Al dar la posibilidad de que el proceso de búsqueda pueda explorar efectivamente el espacio del óptimo local, los algoritmos meméticos han probado ser

efectivos en la resolución de problemas de optimización combinatoria, en especial el problema de la mochila (Chen & Hao, 2016).

Se inspira en el concepto de evolución cultural: el meme (unidad de trasmisión cultural) se propaga y muta dentro de una población. Los memes evolucionan y las ideas fuertes resisten y se propagan mientras que las débiles desaparecen (Neri & Cotta, 2012).

A continuación, se muestra la estructura básica de este algoritmo:

```
Algoritmo_Memético
Inicio
 1    pop ← Inicializar (parámetros, problema)
 2    Repetir
 3         nuevaPop1← Cooperar(pop, parámetros, problema)
 4         nuevaPop2 ← Mejorar(nuevaPop1, parámetros, problema)
 5         pop ← Competir(pop, nuevaPop2)
 6         Si Degenerado(pop) entonces
 7              pop ← Reiniciar(pop, parámetros)
 8         Fin Si
 9    Hasta (Condición de finalización)
10    Retornar MejorSolución (pop); //obtener la mejor solución de la población
Fin
```

Figura 7 Estructura básica del algoritmo memético. Fuente: *(Neri & Cotta, 2012)*

Donde:

a. pop: conjunto de soluciones, cuenta con más de un elemento.
b. parámetros: conjunto entre los que se encuentran el número y los operadores que se aplicarán en la función Cooperar, número de soluciones que se preservarán cuando se invoque a la función Reiniciar, etc.
c. Cooperar(pop, parámetros, problema): función basada en dos operadores para seleccionar soluciones de la población y recombinarlas.
d. Mejorar(nuevaPop1, parámetros, problema): función que aplica el procedimiento de búsqueda local a soluciones en la población.
e. Competir(pop, nuevaPop2): función que reconstruye la población actual, usando la población antigua (pop) y la nueva generada (nuevaPop2).
f. Degenerado(pop): función que comprueba la degeneración de la población, si es así, devolverá el valor Verdadero.
g. Reiniciar(pop, parámetros): función que trata de alejar a la población de la ubicación actual en el espacio de búsqueda. Su implementación varía, pero típicamente se mantiene una fracción de la población actual y se genera nuevas soluciones para completarla.

h. Condición de finalización: puede ser alcanzar el límite del número total de iteraciones, llegar al máximo número de iteraciones sin mejoras o haber reiniciado la población un número N de veces (Neri & Cotta, 2012).

Capítulo 3. Estado del Arte

3.1 Introducción

Esta investigación, del estado del arte, se hizo con el propósito de conocer los métodos de solución que se pueden aplicar a este tipo de problema y obtener referentes para comparar la solución propuesta.

Al no encontrar soluciones para el problema exacto, se presentan investigaciones desarrolladas por otros equipos que buscaron resolver problemas similares.

3.2 Metodología aplicada a la investigación

Para la revisión del estado del arte se utilizó el método de revisión sistemática que se detalla en el Anexo A.

3.3 Revisión y discusión

3.3.1 Investigaciones

En esta sección se describe brevemente los problemas y las soluciones planteadas en las investigaciones encontradas.

- **Planeación de la producción y programación de las operaciones en Flow shop flexibles: caso de estudio de la industria de azulejos**
  Estudia cómo armar lotes de manera eficiente en entornos en los que hay varios productos, etapas y máquinas por etapa, tomando en cuenta la demanda externa. Entre los aspectos que buscan optimizar está la maximización del uso de la capacidad disponible de la maquinaria y la reducción del costo total de producción, inventario y adquisición externa de productos para completar órdenes.
  Para lograr esto, proponen un modelo de programación de enteros mixtos (MIP) y plantean el uso de dos algoritmos, Optimización de Enjambre de Partículas (PSO por sus siglas en inglés) y Rolling Horizon Heuristic (RHH). Luego de varias pruebas, compararon los resultados obtenidos y concluyeron que el PSO genera mejores resultados, pero requiere de un mayor tiempo de procesamiento que el RHH (Ramezanian, Sanami, & Nikabadi, 2017).
- **Modelado del problema de agrupación de material magnético para su horneado y la especialización del algoritmo genético**
  Su propósito es reducir el costo de producción con la elaboración de un método que agrupe las órdenes de trabajo de forma eficiente, considerando la fecha de entrega,

prioridad y demanda de los productos. Para ello implementa el Algoritmo Genético Especializado (SGA por sus siglas en inglés), que mejora con un operador voraz de recombinación en tres puntos, controlando de manera más efectiva la evolución, alcanzando una mayor precisión y velocidad de convergencia; además, aplica un operador de mutación para mejorar la explotación del algoritmo propuesto.

Luego, realiza pruebas a este algoritmo y a otros dos métodos ya existentes (Algoritmo genético de regulación dinámica de parámetros y Algoritmo genético con Colonia de hormigas) y compara los resultados obtenidos, concluyendo que el algoritmo propuesto resuelve el problema de manera más exacta y estable en un tiempo corto de procesamiento (Liu, Pan, & Chai, 2015).

- **Programación coordinada de la producción y el transporte en Flow shops de ensamblaje de dos etapas**

  Busca reducir el costo total de envío y el tiempo promedio de entrega de productos ensamblados en dos etapas. Para ello descompone el problema en dos partes: i) determina la secuencia de trabajos que hará cada máquina en las dos etapas de ensamblaje; y ii) asigna los productos a un lote determinado para su entrega. Considerando que las máquinas tienen una capacidad limitada y que los productos a elaborar se componen de varias partes, se necesita que todas estas sean fabricadas para poder ensamblar el producto.

  Por eso presenta una nueva metaheurística HGA-OVNS, que es la hibridación del algoritmo genético, Búsqueda de vecindad variable (VNS) y Optimización basada en aprendizaje (OBL). Eligieron el algoritmo genético porque es usado en un amplio rango de problemas de optimización combinacional; VNS, por ser un procedimiento de búsqueda local efectivo; y OBL, para mejorar la eficiencia del VNS.

  Una vez implementado HGA-OVNS, lo compara con un algoritmo genético simple y un algoritmo genético híbrido con VNS (HGA-VNS) y concluye que el HGA-OVNS tiene un mejor desempeño, converge más rápido y genera soluciones de mejor calidad que los otros dos algoritmos (Wang, Ma, Luo, & Qin, 2016).

- **El problema de la mochila como herramienta para resolver el problema de la planeación de producción en pequeñas fundiciones**

  Busca resolver una dificultad en el sector de fundiciones: la planeación de la producción -que se compone de dos etapas: determinar la aleación que se fusionará y los lotes que se producirán- para aumentar las ganancias, reducir el costo de

producción, inventario de productos incompletos y terminados, así como, las pérdidas causadas por desperdicio de materiales y multas debido a retrasos.

Se propone un algoritmo genético que toma en consideración el número de periodos para los cuales se hará la planeación, la capacidad de los hornos, la demanda de las aleaciones y tipos de productos a fabricar. Una vez implementado, compara el desempeño del algoritmo genético con la heurística de carga balanceada de hornos (BFLH), notando que el algoritmo genético provee mejores soluciones (alejadas un 4% de la solución óptima) que las de BFLH (alejadas un 11% de la solución óptima) (Camargo, Mattiolli, & Toledo, 2012).

- **Métodos de optimización para el problema de tamaño de lote en una fundición automática**

  Se pretende minimizar los costos generados por el retraso de la producción, el almacenamiento de productos terminados y preparación por cambio de aleación; considerando que los hornos tienen capacidad limitada y que en cada lote solo puede haber productos de un solo tipo de aleación.

  Plantean varios métodos para resolver el problema (Algoritmo Genético, Búsqueda Tabú y Evolución diferenciada) y los ponen a prueba con problemas de diferentes escalas de tamaño y con límites de tiempo específicos para cada escala.

  Analizan los resultados y los comparan con los obtenidos de las pruebas de CPLEX y Programación de restricciones (CP), mostrando que: el algoritmo genético tiene mejor desempeño que CPLEX en todas las escalas de tamaño, Evolución Diferenciada (DE) solo tuvo buen desempeño con problemas de pequeña escala, la búsqueda Tabú tuvo peores resultados que todos excepto CP -porque opera en una sola solución mientras que los otros operan en una población de soluciones- y CP tuvo el desempeño más pobre.

  Enfatizan en que las heurísticas tienen la ventaja de poder manejar problemas más complejos, mientras que los métodos propuestos por soluciones comerciales no son satisfactorios y tienen la limitación de solo poder resolver problemas formulados como programación de enteros mixtos (Duda & Stawowy, 2013).

- **Estudio en la combinación de carga para forjas basada en el algoritmo de Salto de Rana Aleatorio Discreto**

  Trata de reducir el consumo de energía en el proceso de cocción, para esto se tiene que reducir el número de lotes que serán cargados a los hornos, maximizar la carga promedio de los lotes y disminuir el tiempo y temperatura promedio a la que se

someten; respetando el peso máximo que pueden soportar las forjas y asegurando que las piezas tengan una temperatura y duración de cocción compatibles.

Se propone el uso del algoritmo Salto de Rana Aleatorio Discreto (DSFLA por sus siglas en inglés) para optimizar la combinación de piezas que se colocarán en los hornos. Este algoritmo cuenta con las ventajas del algoritmo Memético y de la Optimización de Enjambre de Partículas (PSO), y se caracteriza por: su concepto simple, contar con pocos parámetros para ajustar, su rapidez de cálculo, su capacidad de optimización de búsqueda global y la facilidad de su implementación.

Se comparó el desempeño del DSFLA con la eficiencia obtenida por el procedimiento manual y se comprobó que lograba reducir el consumo de energía en el proceso de cocción (Baiqing, Haixing, Shaobu, Yifei, & Fei, 2016).

- **Problema de empaquetado en contenedor tridimensional con lotes heterogéneos**

  Busca reducir los costos de las operaciones de tratamiento térmico y el tiempo de procesamiento de estas operaciones, respetando la fecha de entrega en la mayor cantidad de trabajos posibles y considerando que se puede procesar lotes heterogéneos (que tienen productos de diferentes clases).

  Para determinar los lotes y programar su procesamiento utiliza un algoritmo de evolución basado en una clave aleatoria (EA por sus siglas en inglés) debido a su versatilidad y a su aplicación común a problemas de optimización combinatoria. Luego, prueba y compara los resultados del EA con los de un algoritmo Maximum box constructive algorithm (CA), observando que el EA reduce el valor de la función fitness hasta en un 64% del valor del CA pero hay ocasiones en los que no encuentra una mejor solución. Sin embargo, en todas las pruebas consiguió reducir el tiempo de procesamiento en 10 horas.

  Finalmente, señala que el principal problema que se notó es que el EA requiere un tiempo considerable de optimización (desde 3 minutos hasta una hora y media) en comparación al CA que solo toma segundos (Koblasa, Vavrousek, & Manlig, 2017).

3.3.2 Soluciones comerciales

- **CPLEX Optimization Studio**

  Paquete de software que permite desarrollar y desplegar modelos de optimización para aumentar eficiencia operacional, ganancias y reducir costos (IBM, 2018).

Cuenta con el optimizador CPLEX que se caracteriza por su flexibilidad y por sus solucionadores de programación matemática de alto rendimiento para programación lineal, programación de enteros mixtos, programación cuadrática y programación de problemas cuadráticos con restricciones (IBM, s. f.).

3.4 Conclusiones

Luego de desarrollar la investigación del estado del arte se ha notado lo siguiente:

- No se encontró soluciones para resolver el problema exacto, pero si se halló otras para problemas similares que consideran algunos de los aspectos que conforman el problema que se va a abarcar, como: capacidad limitada en la maquinaria (tanto en peso como en superficie), familias de productos con características diferentes, demanda de los clientes por un tipo específico de productos, fabricación planificada de productos conformados por varias piezas que luego serán ensambladas, etc.
- Las soluciones comerciales actuales, aunque han mejorado en los últimos años, no están en la capacidad de resolver problemas muy complejos o dan soluciones poco satisfactorias (Duda & Stawowy, 2013).

Capítulo 4. Definición de la función objetivo

4.1 Introducción

En este capítulo se desarrolla el objetivo 1 que consiste en definir la función objetivo a ser usada en los algoritmos para alcanzar el resultado 1 que abarca la definición de variables, función objetivo y restricciones que tendrá el problema.

El capítulo está organizado así: primero, se detallan las variables que se utilizarán en las secciones posteriores. Luego, se establece las restricciones del problema que se deben cumplir. Finalmente, se procede a definir la función fitness, que se utilizará para evaluar la calidad de las soluciones halladas.

4.2 Definición de las variables

- Variables del horno

Nombre	Descripción
$VolMaxHorno$	Volumen máximo
W	Cantidad de vagonetas para los que se planeará la carga.
$PesoMaxVagon$	Peso máximo (kg)

- Variables de los compartimentos

Nombre	Descripción
N	Número de sitios o compartimentos en una vagoneta
L_i	Longitud del compartimento i (m)
W_i	Ancho del compartimento i (m)
H_i	Alto del compartimento i (m)

- Variables de los productos y sets

Nombre	Descripción
S	Cantidad de sets
D	Cantidad de productos
X_{ds}	Indica si el producto d forma parte del set s. Tendrá valor 1 si el producto pertenece al set; sino, tendrá valor 0.

$AlmS_s$	Cantidad de sets s que hay en el almacén de sets terminados
$AlmD_d$	Cantidad de productos d que hay en el almacén de productos terminados

- Variables de las piezas

Nombre	Descripción
Y	Cantidad de diferentes tipos de piezas
PH_y	Cantidad de piezas y pendientes por hornear
X_{iwy}	Indica si la pieza y es colocada en el compartimento i en la vagoneta w. Tendrá valor 1 si la pieza está en ese compartimento; sino, tendrá valor 0.
X_{yd}	Indica si la pieza y forma parte del producto d. Tendrá valor 1 si es que la pieza pertenece al producto; sino, tendrá valor 0.
$AlmY_y$	Cantidad de piezas y que hay en el almacén de piezas terminadas

- Variables de las piezas colocadas en los compartimentos

Nombre	Descripción
P_{iw}	Peso de la pieza colocada en el compartimento i en la vagoneta w (kg)
V_{iw}	Volumen de la pieza colocada en el compartimento i en la vagoneta w
l_{iw}	Longitud de la pieza colocada en el compartimento i en la vagoneta w (m)
w_{iw}	Ancho de la pieza colocada en el compartimento i en la vagoneta w (m)
h_{iw}	Alto de la pieza colocada en el compartimento i en la vagoneta w (m)

- Variables de los pedidos

Nombre	Descripción
P	Cantidad de pedidos

X_{ps}	Indica si el pedido p es de sets del tipo s. Tendrá valor 1 si el pedido es por el set s; sino, tendrá valor 0.
$CantPedida_p$	Cantidad de sets solicitados en el pedido p
$ProxFecha_p$	Valor que indica qué tan próxima es la fecha de entrega del pedido p. Está en un rango de 1 al 5 y se asigna así: – 1: si falta más de un mes hasta la fecha de entrega. – 2: si falta entre 16 días a un mes. – 3: si falta entre 1 semana a 15 días. – 4: si falta menos que una semana. – 5: si ya pasó la fecha de pedido.
$PriorCliente_p$	Prioridad del cliente que realizó el pedido p. Este valor se encuentra en un rango de 1 al 3, siendo: – 1: para clientes que no cobran mora por retraso. – 2: para clientes que cobran mora, como tiendas grandes. – 3: para las exportaciones.
$CantFaltante_y$	Valor que indica la cantidad de piezas faltantes para completar un pedido. Toma en cuenta la cantidad de piezas que se encuentran en el almacén.

4.3 Definición de las restricciones del problema

RE1: El peso total de las piezas cargadas no debe superar al peso máximo que puede soportar la vagoneta.

$$\sum_{i=1}^{N} P_{iw} \leq PesoMaxVagon \ , \forall \ w \in \ W$$

RE2: El volumen de las piezas cargadas no debe superar al volumen máximo que puede soportar el horno.

$$\sum_{w=1}^{W} \sum_{i=1}^{N} V_{iw} \leq VolMaxHorno$$

RE3: La pieza debe caber en el compartimento en el que se coloca.

$$\max\{l_{iw}, w_{iw}, h_{iw}\} \leq \max\{L_i, W_i, H_i\}$$

$$\min\{l_{iw}, w_{iw}, h_{iw}\} \leq min\{L_i, W_i, H_i\}$$

$$l_{iw} + w_{iw} + h_{iw} - \max\{l_{iw}, w_{iw}, h_{iw}\} - \min\{l_{iw}, w_{iw}, h_{iw}\}$$
$$\leq L_i + W_i + H_i - \max\{L_i, W_i, H_i\} - min\{L_i, W_i, H_i\}$$

$$\forall\, i \in\ N, \forall\, w \in\ W$$

RE4: Solo se puede colocar como máximo una pieza en cada compartimento.

$$\sum_{y=1}^{Y} X_{iwy} \leq 1\,,\ \ \forall\, i \in\ N, \forall\, w \in\ W$$

RE5: La cantidad de piezas y asignadas a los compartimentos no deben exceder el número inicial de piezas y pendientes por hornear.

$$\sum_{w=1}^{W} \sum_{i=1}^{N} X_{iwy} \leq PH_y\ , \forall\, y \in\ Y$$

4.4 Definición de la función objetivo

Luego de haber analizado las restricciones se ha definido la siguiente función objetivo:

$$Max \sum_{w=1}^{W} \left[\left(\frac{W * \sum_{i=1}^{N} V_{iw}}{VolMaxHorno} \right) * CoefV + \left(\frac{\sum_{i=1}^{N} P_{iw}}{PesoMaxVagon} \right) * CoefP \right] + \left(\frac{SP * CoefDem}{MaxPrioridadV} \right)$$

Donde:

- $CoefDem$, $CoefV$ y $CoefP$: importancia del factor demanda, volumen, peso. Estos coeficientes suman 1.
- SP: es la sumatoria de las prioridades de las piezas cargadas considerando las penalidades (para más detalle ver la sección Aplicación de penalidades en la selección de piezas). SP se expresa de la siguiente manera:

$$SP = \sum_{y=1}^{Y} C_y * \left\{ PPromP_y * \sum_{m=0}^{np_y - 1} rango \left(\frac{CantFaltante_y - m}{maxCantFaltante} \right) \right\}$$

- C_y: indicador que tiene valor 0 si es que no se ha cargado ninguna pieza del tipo y a alguna vagoneta. En caso contrario, tendrá valor 1.
- $maxCantFaltante$: es la máxima cantidad faltante.

$$maxCantFaltante = max\{CantFaltante_y : y = 1..N\}$$

- $PPromPy$: es la prioridad promedio de la pieza y, la cual es igual a la prioridad promedio del set al que pertenece.

$$PPromPy = PrioridadProm_s, \forall X_{yd} \times\ X_{ds} = 1, \forall\, y \in\ Y, \forall\, s \in\ S$$

- $PrioridadProm_s$: es la prioridad promedio que tiene el set pedido; se obtiene del cálculo del promedio ponderado de todos los pedidos del set s.

$$PrioridadProm_s = \frac{\sum_{p=0}^{P}\left(X_{ps} * CantPedida_p * ProxFecha_p * PriorCliente_p\right)}{\sum X_{ps} * CantPedida_p}$$

- $Rango()$: función que sirve para asignar un nivel al valor recibido como parámetro; devuelve un valor entre 0 y 10.

$$rango(valor) = \begin{cases} a+1, & si\ b > 0 \\ a, & si\ b = 0 \end{cases}, \qquad donde: a * 10 + b = valor * 100$$

  Por ejemplo: si la cantidad faltante de la pieza y representa el 75% de la $maxCantFaltante$ entonces el valor de su rango será 8.

- $CantFaltante_y$: cantidad de piezas necesaria para completar los pedidos.

$$CantFaltante_y = Max\left\{0, \left(\sum_{p=1}^{P} X_{ps} * CantPedida_p\right) - AlmS_s - AlmD_d - AlmY_y\right\}$$

$$\forall X_{yd} \times X_{ds} = 1, \forall\ y \in Y, \forall\ d \in D, \forall\ s \in S$$

- np_y: es la cantidad de piezas del tipo y que se considerarán en la suma de prioridades. Este valor es el mínimo entre la cantidad de piezas cargadas al horno o la cantidad faltante para completar un pedido.

$$np_y = \min\left(\sum_{w=1}^{W}\sum_{i=1}^{N} X_{iwy}, CantFaltante_y\right)$$

Ejemplo de aplicación:

En la Figura *8*, se muestra cómo se calcula el valor fitness para una solución.

$$CoefV = CoefP = CoefDem$$

Datos del horno	
Volumen Máximo	8 m^3
Cant. de vagonetas	2
Peso Máximo (por vagoneta)	25 kg
Max prioridad (por vagoneta)	450

Pieza	Peso	Volumen	Prioridad promedio	Cant. faltante	Cant. pendiente por hornear
1	4.5 kg	1 m^3	8	2	4
2	15 kg	2 m^3	6	4	5
3	5 kg	0.9 m^3	15	3	1

Calculando el fitness obtenido por:

- Volumen cargado

  V1: 3.9 m^3/(8m^3/2) = 0.98

  V2: 4 m^3/(8m^3/2) = 1.00

  1.98

  Fitness: 1.98 x CoefV = **0.66**

- Peso cargado

  V1: 24.5 kg/25 kg = 0.98

  V2: 24 kg /25 kg = 0.96

  1.94

  Fitness: 1.94 x CoefP = **0.65**

- Demanda de las piezas cargadas

  Pieza 1: 8 x rango(2/4) + 8 x rango(1/4) = 40 + 24 = 64

  Pieza 2: 6 x rango(4/4) + 6 x rango(3/4) = 60 + 48 = 108

  Pieza 3: 15 x rango(3/4) = 120

  292

  Fitness: 292 x CoefDem / 450 = **0.22**

Hallando el fitness de la solución: 0.66 + 0.65 + 0.22 = **1.53**

Figura 8 Cálculo del fitness de una solución. Fuente: *Elaboración propia.*

Capítulo 5. Estructuras de datos

5.1 Introducción

En esta parte se muestra las estructuras que usarán los algoritmos memético y genético para representar la solución. Después, se detalla las estructuras auxiliares que usarán para almacenar datos y cálculos intermedios necesarios para hallar la solución.

5.2 Estructura de la solución del algoritmo memético

En este proyecto, el problema a resolver es determinar qué piezas se deben cargar al horno, cada una de las piezas seleccionadas deberá ser colocada en un compartimento diferente, pudiendo haber varias piezas de un mismo tipo en el grupo seleccionado.

Considerando estos aspectos, se ha definido la estructura de la solución como una matriz de 2 dimensiones (compartimento x vagoneta), que se presenta a continuación:

	Vagoneta 1	**Vagoneta 2**	**Vagoneta 3**	**…**
Compartimento 1	P1	P10	P10	
Compartimento 2	P5	P9	P6	
Compartimento 3	Py	P9	0	
…				

Figura 9 Estructura de la solución. Fuente: *Elaboración propia.*

Cada fila se identifica con un compartimento concreto, así, la fila 1 contiene todas las piezas que se colocarán en el compartimento 1; la segunda, las del compartimento 2 y así sucesivamente. En el caso de las columnas cada una representa una vagoneta, así, la columna 1 representa la vagoneta 1; la columna 2, la vagoneta 2 y así sucesivamente.

El valor dentro de cada una de las celdas, señala la pieza que irá en un compartimento determinado en una vagoneta específica. Por ejemplo: en la Figura *9*, la pieza Py representa a una pieza con código “y” que ha sido colocada en el compartimento 3 en la primera vagoneta. En los compartimentos vacíos se ha colocado el valor 0.

Resumiendo, la estructura planteada permite asignar un mismo tipo de pieza a varios compartimentos y facilita la creación y modificación de soluciones porque rápidamente se puede comprobar qué pieza ha sido asignada a cada sitio. También, ayuda a cumplir la restricción R4 que establece que solo puede haber una pieza por espacio.

5.3 Estructura del cromosoma

Para el algoritmo genético, se usará esta estructura para representar una solución:

Vagoneta 1					Vagoneta 2					…
C1	**C2**	**C3**	**C4**	**C5**	**C1**	**C2**	**C3**	**C4**	**C5**	...
1	4	3	2	2	1	3	3	2	0	

Figura 10 Estructura del cromosoma. Fuente: *Elaboración propia.*

Cada columna del arreglo representa un compartimento y todos los compartimentos de una vagoneta se encuentran colocados de manera sucesiva en la estructura. Es decir, las columnas que representan los compartimentos de la vagoneta 1 son consecutivos en la estructura; después de todos los compartimentos de una misma vagoneta, se encuentran los compartimentos de la siguiente.

El valor que se encuentra en cada una de las celdas, es el código de la pieza colocada en ese compartimento de la vagoneta. Por ejemplo: en la Figura *10*, se ha asignado una pieza de código 2 a los compartimentos 4 y 5 de la vagoneta 1 y al compartimento 4 de la vagoneta 2. En los compartimentos vacíos, se pondrá 0 en la celda de esa columna.

5.4 Estructuras auxiliares

A continuación, se listan las estructuras auxiliares que emplearán ambos algoritmos.

- **Horno**: tiene los datos básicos a conocer para planear la carga, el volumen máximo que se puede cargar y el número de vagonetas que entran.

Máximo volumen (m^3)	Número de vagonetas
108.1	46

- **Vagoneta**: estructura modelo de todas las vagonetas. Sus datos son: peso máximo a cargar, número de compartimentos y lista de compartimentos.

Máximo peso (kg)	Número de compartimentos	Compartimentos
1000	20	{C1, C2, C3, …, CN}

- **Compartimento**: esta estructura contiene las dimensiones del compartimento.

Id	Ancho (m)	Largo (m)	Alto (m)
1	0.300	0.400	0.400

- **Gestor de sets**: agrupa todos los datos relacionados a los sets.

Lista de sets

{Set 1, Set 2, Set 3, …, Set N}	Resumen de sets

- **Set:** almacena los datos de un set específico como: modelo, color, cantidad y lista de productos que lo componen.

Código	Modelo	Color	Número de productos	Lista de productos
1	Manantial	Blanco	3	{Prod1, Prod2, Prod3}

- **Resumen de sets**: facilitará el cálculo de la prioridad de las piezas (valor utilizado en la función objetivo). Tiene como datos: cantidad total pedida, cantidad de sets en el almacén de sets terminados, cantidad faltante (igual a la cantidad pedida menos la que está en almacén) y la prioridad promedio del set.

	Set 1	**Set 2**	**Set 3**	**…**	**Set S**
Cantidad pedida	100	20	4		0
Cant. en almacén	20	5	0		0
Cantidad faltante	80	15	4		0
Prioridad promedio	7	14	35		0

- **Gestor de productos**: agrupa todos los datos relacionados a los productos.

Lista de productos

{Prod 1, Prod 2, Prod 3, …, Prod D}	Resumen de productos

- **Producto**: tiene como datos: nombre, número y lista de piezas que lo componen.

Código	Nombre	Número de piezas	Lista de piezas
56	Inodoro A	2	{P1, P2}

- **Resumen de productos**: tiene forma y función similar al resumen de sets, pero con datos de productos. La cantidad pedida es igual a la cantidad faltante del set al que pertenece y su prioridad promedio es igual a la del set del que forma parte.

	Prod 1	**Prod 2**	**Prod 3**	**…**	**Prod D**
Cant. pedida	100	20	4		0
Cant. almacén	20	5	0		0
Cant. faltante	80	15	4		0

Prioridad promedio	7	14	35		0

- **Gestor de Piezas**: agrupa todos los datos relacionados a las piezas.

Lista de piezas	
{Pieza 1, Pieza 2, Pieza 3, …, Pieza Y}	Resumen de piezas

- **Pieza**: tiene datos de la pieza como nombre, dimensiones, peso y volumen.

Id	Nombre	Ancho (m)	Largo (m)	Alto (m)	Volumen (m^3)	Peso (kg)
7	Taza	0.335	0.390	0.180	0.0235	5

- **Resumen de piezas**: es similar a la de resumen de sets, pero tiene 1 campo adicional: la cantidad de piezas pendientes para hornear. La cantidad pedida de cada pieza es igual a la cantidad faltante del producto del que forma parte y su prioridad promedio es igual a la del producto al que pertenece.

	P1	**P2**	**P3**	**…**	**PY**
Cant. pedida	100	20	4		0
Cant. almacén	20	5	0		0
Cant. faltante	80	15	4		0
Cant. pendiente de hornear	50	20	35		3
Prioridad promedio	7	14	35		0

- **Pedido**: contiene datos como el set que se está solicitando, la cantidad pedida, la fecha de entrega y la prioridad que tiene el cliente que realiza el pedido.

Código	Set pedido	Cantidad	Fecha de entrega	Prioridad del cliente
00001	1	50	25/10/2018	3

- **Matriz de dimensiones**: facilitará la construcción de soluciones ya que permitirá verificar rápidamente si una pieza cabe en un compartimento determinado.

	C1	**C2**	**C3**	**…**
Pieza 1	V	V	F	
Pieza 2	V	V	F	
…				

- **Contador de piezas colocadas**: cada solución creada tendrá un contador para registrar la cantidad de piezas de cada tipo colocadas en los compartimentos. Se empleará en la asignación de piezas, para más información sobre su uso revisar la sección Aplicación de penalidades en la selección de piezas.

Pieza 1	Pieza 2	Pieza 3	…	Pieza Y
5	11	8		0

Capítulo 6. Generación de la población inicial

6.1 Introducción

En este capítulo se detalla cómo se generará la población inicial para los algoritmos memético y genético. Se comienza explicando de manera general el algoritmo GRASP y luego se procede a detallar cada paso.

6.2 GRASP

La teoría señala que conviene generar la población inicial aplicando un algoritmo metaheurístico, porque se parte de soluciones suficientemente buenas (Feo & Resende, 1995); por eso se usará el algoritmo GRASP fase construcción (ver Figura *11*) para generar la población inicial de la cual partirán el algoritmo memético y el genético.

```
GRASP (tamPoblacion, alpha)
Inicio
 1    CS ← Inicializar población vacía
 2    i ← 0
 3    Mientras (i< tamPoblacion) hacer
 4        s ← ConstruirSolucion(alpha, mDimension, rPiezas, lPiezas)
 5        Si solucionValida(s) entonces
 6                AgregarSolucion(CS,s)
 7                i ← i+1
 8        Fin Si
 9    Fin Para
10    Retornar(CS)
Fin
```

Figura 11 Pseudocódigo del algoritmo GRASP. Fuente: *Elaboración propia.*

Con el algoritmo GRASP se creará una población inicial con tamPoblacion soluciones. Para crear cada solución se utilizará la función ConstruirSolucion (ver Figura *12*), que selecciona las piezas a ser cargadas y especifica su ubicación. Una vez creada la solución, se verifica su validez y se agregará al conjunto de soluciones CS.

La función ConstruirSolucion, hace una lista con las prioridades de las piezas que caben en el compartimento k (línea 5); luego, si la lista no es vacía, forma un subconjunto llamado RCL con las mejores piezas de esa lista (línea 7). Después, elige una pieza al azar del RCL (línea 8) para añadirla a la solución en el compartimento k en la vagoneta w, actualizando el contador de piezas colocadas de este tipo (línea 9). Posteriormente, se procede a llenar el compartimento k de la siguiente vagoneta repitiendo los pasos de las líneas 5 al 11. Una vez llenos todos los compartimentos k de las vagonetas, o agotadas las piezas pendientes que caben en ese compartimento, se pasará a cargar los compartimentos k+1 y así sucesivamente hasta llenar todas las vagonetas.

```
ConstruirSolucion(alpha, mDimension, rPiezas, lPiezas)
Inicio
 1    solución ← Inicializar
 2    k ← 0
 3    w ← 0
 4    Mientras k < CantCompartimentos hacer
 5        prioridades ← actualizarPrioridad(k, solucion, mDimension, rPiezas, lPiezas)
 6        Si no esVacia(prioridades) entonces
 7                RCL ← seleccionarRCL(prioridades, alpha)
 8                pieza ← seleccionarPiezaAlAzar(RCL)
 9                actualizarSol(solución, k, w, pieza)
10                w ← w+1
11        Fin Si
12        Si w = CantVagonetas o esVacia(prioridades)entonces
13                k ← k+1
14                w ← 0
15        Fin Si
16    Fin Mientras
17    Retornar(solución)
Fin
```

Figura 12 Pseudocódigo de la función ConstruirSolucion. Fuente: *Elaboración propia.*

La función actualizarPrioridad (ver Figura *13*) devuelve una lista de las piezas candidatas con sus prioridades en función a la demanda y los porcentajes de ocupación con respecto a la capacidad de peso y volumen del compartimento.

Durante el cálculo del factor demanda (línea 7), presente en la fórmula de la prioridad de la pieza, se asignará una penalidad cada vez que se seleccione ese tipo de pieza; para mayor detalle revisar Aplicación de penalidades en la selección de piezas.

```
actualizarPrioridad(k, solucion, mDimension, rPiezas, lPiezas)
Inicio
 1    prioridades ← Inicializar
 2    Para i=1 hasta NumPiezas hacer
 3      Si mDimension(i,k) y rPiezas.pendientes(i)>0 entonces
 4            hayPorHornear ← solucion.contador(i) < rPiezas.pendientes(i)
 5            Si hayPorHornear entonces
 6                pedidosPorCompletar ← max(rPiezas.faltantes(i) - solucion.contador(i), 0)
 7                factorDemanda ← (rPiezas.prioridadProm(i) * rango(pedidosPorCompletar /
                                  maxFaltante)) / Max_Prioridad
 8                factorPeso ← lPiezas(i).peso/pesoLimite(k)
 9                factorVol ← lPiezas(i).volumen/volLimite(k)
10                valor ← 100 * (factorDemanda * Coef_Dem + factorPeso * CoefP + factorVol
                          * CoefV)
11                Si pedidosPorCompletar<1 entonces
12                      valor ← valor/100
13                Fin Si
14                agregarElemento(prioridades, i, valor)
15            Fin Si
16      Fin Si
18    Fin Para
19    Retornar(prioridades)
```

Fin

Figura 13 Pseudocódigo de la función actualizarPrioridad. Fuente: *Elaboración propia.*

La función seleccionarRCL lista las piezas con prioridad mayor al límite inferior, que es:

$$LimInf = \max(prioridades) - \alpha(\max(prioridades) - \min(prioridades))$$

Aplicación de penalidades en la selección de piezas

Para evitar que las piezas con mayor demanda inicial sean favorecidas y seleccionadas siempre, se ha incorporado penalidades para reducir el factor demanda de un tipo de pieza cada vez que se coloque en un compartimento. En la Figura *14*, se explicará mediante un ejemplo cómo se aplican las penalidades.

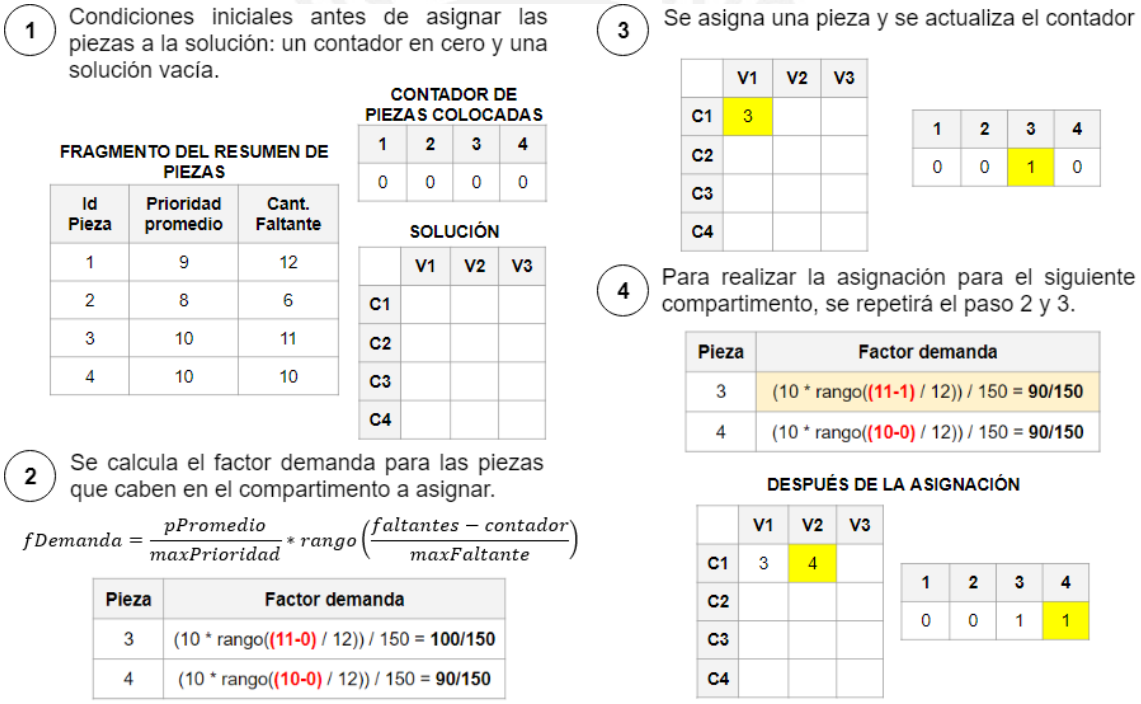

Figura 14 Explicación de la aplicación de penalidades. Fuente: *Elaboración propia.*

Las penalidades se aplicarán durante la construcción de la solución con el algoritmo GRASP y cada vez que se agregue o quite una pieza a la solución durante el desarrollo de los algoritmos memético y genético.

6.2.1 Calibración del algoritmo GRASP

Para obtener un conjunto de buenas soluciones se debe calibrar dos de sus parámetros: alfa y el tamaño que tendrá la población. Por ello se realizarán pruebas con diferentes valores, que se repetirán 40 veces cada una.

Alfa

Para calibrar este parámetro se probó con un valor fijo para el número de iteraciones igual a mil y se varió el valor del alfa desde 0.1 hasta 0.9. Como resultado, se obtuvo los datos que están en el Anexo B y que se encuentran resumidos en la Tabla *1*.

Tabla 1 Resumen de la calibración del alfa

Alfa	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Promedio	**13.5031**	13.3935	13.1306	13.0918	13.0400	12.9227	12.7752	12.4093	11.2675
Mínimo	13.4919	13.3794	13.0814	13.0190	12.9673	12.8540	12.6842	12.3265	10.9613
Máximo	13.5149	13.4081	13.2171	13.1820	13.2089	13.0118	12.9260	12.5008	11.6510
Desv. Estándar	0.0050	0.0079	0.0261	0.0363	0.0434	0.0357	0.0527	0.0500	0.1914

Se tuvo mejores resultados con valores de alfa bajos. Debido a que se obtuvieron las mejores soluciones cuando el GRASP era muy voraz (con alfa igual a 0.1) se decidió no utilizar el valor hallado en la calibración sino emplear un alfa igual a 0.4, valor que mantiene un mayor equilibrio entre aleatoriedad y voracidad.

Tamaño de la población

Luego de calibrar alfa, se hizo lo mismo con el tamaño de la población. Se realizaron varias pruebas en las que se modificó este parámetro desde 1000 hasta 10000 en intervalos de 500 y para cada valor del tamaño de población se realizaron 40 pruebas.

Estas pruebas se hicieron en un conjunto de datos de 500 piezas y en otro de 700 piezas. En la Tabla *2* se muestran los resultados obtenidos.

Tabla 2 Resumen de la calibración del tamaño de la población

Pruebas con 500 piezas		
Tamaño de la población	**Fitness promedio**	**Tiempo prom. (s)**
1000	12.55	14.2731
1500	12.59	21.2662
2000	12.60	28.3401
2500	12.62	35.1122
3000	12.61	42.2832
3500	12.61	49.4617
4000	12.62	56.5394
4500	12.63	63.5281
5000	12.63	70.4587
5500	12.65	77.2373
6000	12.64	84.6654
6500	12.64	92.9680

Pruebas con 700 piezas		
Tamaño de la población	**Fitness promedio**	**Tiempo prom. (s)**
1000	12.43	25.2845
1500	12.44	37.8960
2000	12.45	51.7486
2500	12.45	64.0504
3000	12.46	80.1699
3500	12.47	89.4006
4000	12.47	114.2423
4500	12.47	121.8354
5000	12.48	140.4264
5500	12.48	155.7993
6000	12.48	158.1698
6500	12.48	173.2735

7000	12.69	100.0011
7500	**12.82**	106.0205
8000	**12.83**	112.7001
8500	12.66	121.8808
9000	12.65	132.0804
9500	12.66	137.7404
10000	12.65	142.1998

7000	12.56	187.0630
7500	**12.71**	189.0035
8000	12.62	209.8037
8500	12.58	239.6100
9000	12.58	247.6866
9500	12.58	248.3374
10000		

Además, en la Figura *15* se puede visualizar la variación de los valores del fitness según el tamaño de la población.

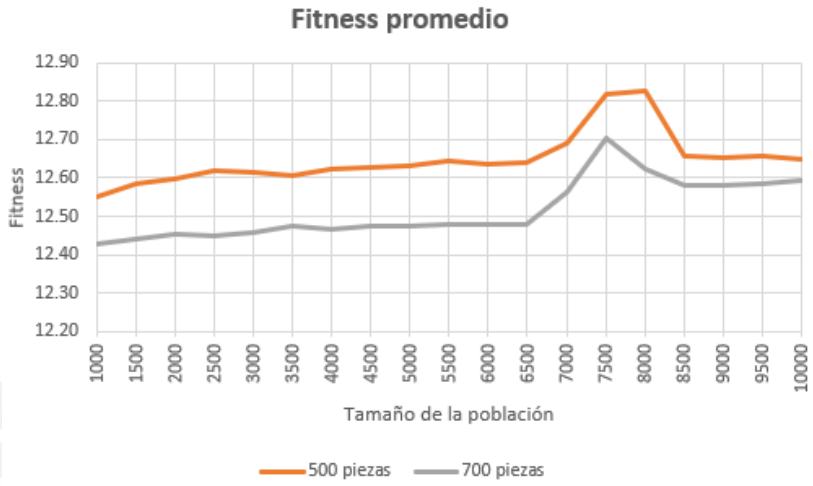

Figura 15 Fitness promedio por tamaño de la población. Fuente: *Elaboración propia.*

En el gráfico se nota que con un tamaño de población igual a 7500 se tiene resultados considerablemente mejores para ambos conjuntos de datos, por esta razón se seleccionó este valor.

Capítulo 7. Algoritmo memético

7.1 Introducción

En este capítulo se presenta al algoritmo memético, describiendo brevemente las funciones que lo componen. Después se detalla las funciones encargadas de la aplicación de los operadores, la actualización de la población y la restauración de ésta.

7.2 Algoritmo memético

Se comparará con el genético, uno de los métodos más usados con esta clase de problemas según lo revisado en el estado del arte. A continuación, se presenta el pseudocódigo del algoritmo memético (ver Figura *16*):

```
Algoritmo_Memetico (poblacionInicial)
Inicio
 1    t ← Inicializar temporizador
 2    generacion ← 0
 3    sinMejora ← 0
 4    pop ← poblacionInicial
 5    mejorSol ← obtenerMejor(pop)
 6    Repetir
 7         nuevaPoblacion ← GenerarNuevaPoblacion(pop)
 8         pop ← ActualizarPoblacion(pop, nuevaPoblacion)
 9         mejorSolActual ← obtenerMejor(pop)
10         Si fitness(mejorSol) >= fitness(mejorSolActual) entonces
11              sinMejora ← sinMejora+1
12         Si no
13              mejorSol ← mejorSolActual
14              sinMejora ← 0
15         Fin Si
16         Si (sinMejora = MaxSinMejora) entonces
17              pop ← RestaurarPoblacion(pop)
18         Fin Si
19         generacion ← generacion +1
20    Hasta generacion > MaxGeneraciones o t > tiempoLímite
21    Retornar(mejorSol)
Fin
```

Figura 16 Pseudocódigo del algoritmo memético. Fuente: *Elaboración propia.*

Este algoritmo recibe como parámetro inicial la población obtenida por el algoritmo GRASP. Luego, en la línea 7, genera una nueva población mediante la aplicación de los operadores de cruce, mutación y búsqueda local (ver Aplicación de operadores).

Después, en la línea 8, unifica la población anterior con la nueva para obtener otra con las mejores soluciones de ambas (ver Actualización de la población) y busca la mejor solución de esta población (línea 9), para comprobar si hay alguna mejora con respecto a la generación anterior (línea 10), si no la hay se aumentará el contador sinMejora.

Finalmente, en la línea 16, verifica si la población se degeneró (es decir, si se estancó en algún óptimo local), de ser así la restaurará (ver Restauración de la población).

Las líneas 7 a la 19 se repiten hasta cumplir alguna de las condiciones de parada: alcanzar el número máximo de generaciones o sobrepasar el tiempo límite (línea 20).

7.2.1 Aplicación de operadores

En esta sección, se detalla los operadores utilizados (selección, recombinación, mutación y búsqueda local) y su aplicación en el pseudocódigo de la función GenerarNuevaPoblacion.

- **Operador de selección:** se usa para escoger a los individuos (padres) que serán afectados por el operador de recombinación y para elegir las soluciones a las que se aplicará la búsqueda local. Para la selección se emplea el método de ruleta, que consiste en asignar a cada solución un sector circular de la ruleta proporcional a su valor fitness (ver Figura *17*), de manera que, al girar la ruleta las mejores soluciones tengan una mayor probabilidad de ser escogidas. La ruleta a implementar tendrá búsqueda binaria debido a que con esta le tomará en el peor de los casos O(log n) comparaciones para encontrar el valor seleccionado (Lipowski & Lipowska, 2012).

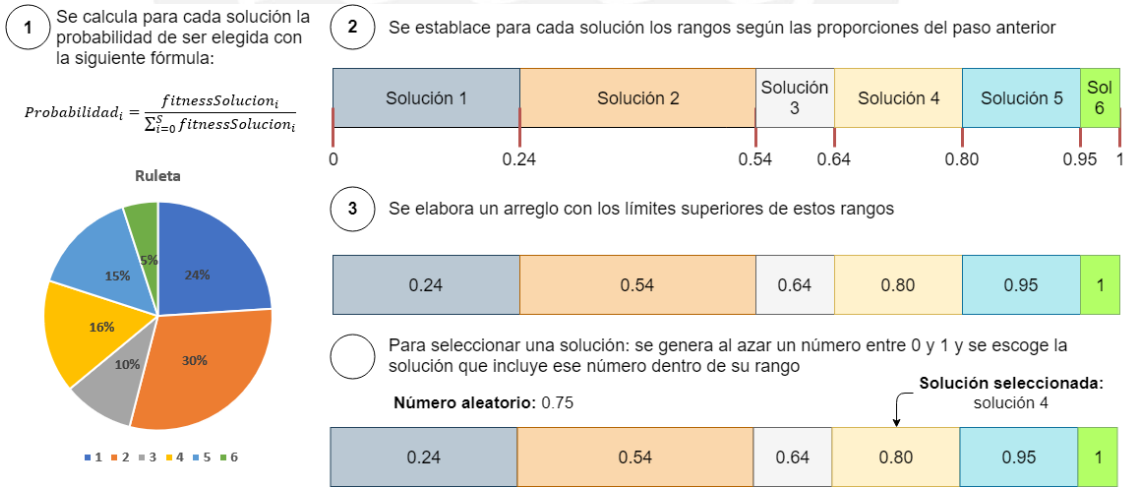

Figura 17 Elaboración de la ruleta y selección de soluciones. Fuente: *Elaboración propia*

- **Operador de recombinación:** se emplea para formar una nueva población. Usa los individuos escogidos por el operador de selección y tiene como parámetro la tasa de casamiento, que determina el número de individuos que se generarán. Para realizar el cruce se aplicará recombinación uniforme, que consiste en generar un número aleatorio entre 0 y 1 para cada elemento que compone una solución, si este

número es menor que la probabilidad p_c entonces se asigna el elemento del primer padre al primer hijo y el del segundo padre al segundo hijo, sino, se invertirá la asignación (Magalhaes-Mendes, 2013). En la Figura *18* se muestra un ejemplo con 2 soluciones para hornos de 3 vagonetas con 2 compartimentos para un p_c de 0.5.

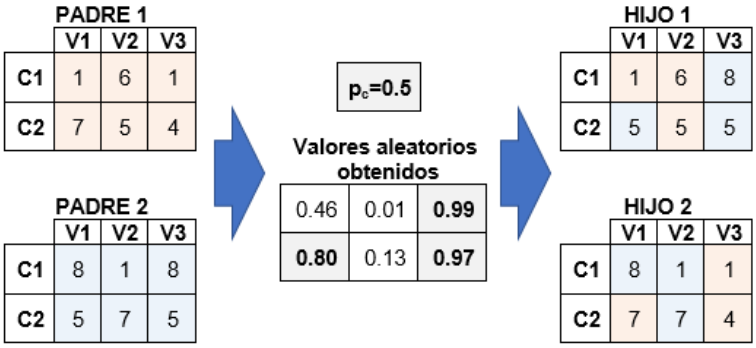

Figura 18 Ejemplo de recombinación uniforme. Fuente: *Elaboración propia.*

- **Operador de mutación:** se usa para modificar ligeramente una solución; para ello emplea una tasa de mutación y un valor generado al azar entre 0 y 1, si el valor generado es menor a la tasa se aplicará el operador. La mutación es el reemplazo de una pieza asignada por otra que quepa en el mismo compartimento. En la Figura *19*, se ejemplifica la mutación reemplazando el elemento del compartimento 2 de la vagoneta 2 (se cambia una pieza de código 5 por otra de código 7).

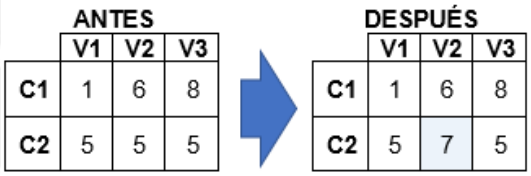

Figura 19 Ejemplo de mutación. Fuente: *Elaboración propia.*

- **Operador de búsqueda local:** emplea la heurística k-opt, que reemplaza *k* elementos presentes en la solución actual por otros que no forman parte de esta. Esta búsqueda, basada en la investigación de Ishibuchi, Tanigaki, y otros, solo se hará cada gLs iteraciones, se aplicará a un número reducido de individuos de la población (determinado por probLS) y solo visitará a nLS vecinos (Ishibuchi, Tanigaki, Akedo, & Nojima, 2013).

A continuación, se muestra la función GenerarNuevaPoblacion (ver Figura *20*):

```
GenerarNuevaPoblacion(pop, numGen)
Inicio
 1    nuevaPoblacion ← Inicializar población vacía
 2    rangosRuleta ← prepararRuleta(pop) // devuelve rangos para la ruleta búsqueda binaria
```

```
 3   Para i=1 hasta (tasaC * pop.size) hacer
 4       padre1 ← ruleta(pop, rangosRuleta)
 5       padre2 ← ruleta(pop, rangosRuleta)
 6       hijos ← uniformCrossover(padre1, padre2)
 7       Para j=1 hasta 2 hacer
 8           valorR ← random(0,1)
 9           Si valor< tasaMut entonces
10               hijos[j] ← mutar(hijos[j], nCambiar)
11           Fin Si
12           Si esValida(hijos[j]) entonces
13               AgregarSolucion(nuevaPoblacion, hijo[j])
14           Fin Si
15       Fin Para
16   Fin Para
17   Si (numGen div gLS = 0) entonces
18       tamañoPob ← pop.size
19       rangosNuevaRuleta ← prepararRuleta(nuevaPoblacion)
20       Para i=1 hasta (probLs * pop.size) hacer
21           actual ← ruleta(nuevaPoblacion, rangosNuevaRuleta)
22           mejor ← actual
23           cambio ← falso
24           Para j=1 hasta nLS hacer
25               actual ← mutar(actual, nCambiarLS)
26               Si esValida(actual) y (fitness(actual) > fitness(mejor)) entonces
27                   mejor ← actual
28                   cambio ← verdadero
29               Fin Si
30           Fin Para
31           Si cambio entonces
32               AgregarSolucion(nuevaPoblacion, mejor)
33           Fin Si
34       Fin Para
35   Fin Si
36   Retornar(nuevaPoblacion)
Fin
```

Figura 20 Pseudocódigo de la funcion GenerarNuevaPoblacion. Fuente: *Elaboración propia.*

7.2.2 Actualización de la población

Una vez generada la nueva población se unificará con la actual, seleccionando los mejores elementos de ambas poblaciones para conformar un grupo compuesto por la misma cantidad de individuos que la población actual.

Se escogió la estrategia de adición porque asegura que no se empeoren los valores de la función objetivo y, a diferencia de la estrategia de coma, es más rápido y no requiere una población de descendientes tan grande (Datoussaid, Verlinden, & Conti, 2002).

7.2.3 Restauración de la población

Para determinar si una población se ha degenerado, se utilizará como indicador la cantidad de generaciones que han pasado sin encontrar alguna solución mejor a la

actual. Si esta cantidad supera el límite establecido (MaxSinMejora) se procederá a restaurar la población con la siguiente función (ver Figura *21*):

```
RestaurarPoblacion(pop)
Inicio
 1    nuevaPoblacion ← Inicializar población vacía
 2    numPreservados ← tamaño(pop) * %preservar
 3    i ← 0
 4    Mientras i < numPreservados hacer
 5        sol ← extraerMejor(pop)
 6        AgregarSolucion(nuevaPoblacion, sol)
 7        i ← i+1
 8    Fin Mientras
 9    Mientras i < tamaño(pop) hacer
10        s ← ConstruirSolucion(alpha, mDimension, rPiezas, lPiezas)
11        mutar(s, nCambiar)
12        Si solucionValida(s) entonces
13            AgregarSolucion(nuevaPoblacion, s)
14            i ← i+1
15        Fin Si
16    Fin Mientras
17    Retornar(nuevaPoblacion)
Fin
```

Figura 21 Pseudocódigo de la función RestaurarPoblacion. Fuente: *Elaboración propia.*

En RestaurarPoblacion, se conserva un grupo pequeño con las mejores soluciones de la población actual y el resto se desecha. Para completar la población se genera nuevas soluciones mediante GRASP, que serán mutadas antes de agregarse a la población.

7.3 Calibración del algoritmo Memético

Para obtener mejores soluciones se calibrarán los parámetros: tasa de recombinación, probabilidad de la recombinación uniforme, tasa de mutación y el porcentaje de soluciones que se conservarán en la restauración de la población. Por ello se realizarán pruebas con diferentes valores, las cuales se repetirán 40 veces cada una.

También, se podría calibrar los parámetros de la búsqueda local (intervalo de aplicación, porcentaje de aplicación y el máximo de vecinos visitados en cada búsqueda), pero se usarán los valores indicados en la investigación de Ishibuchi, Tanigaki, y otros, porque la búsqueda local del algoritmo memético diseñado se basa en dicho estudio (Ishibuchi, Tanigaki, Akedo, & Nojima, 2013). Los valores usados se muestran en la Tabla *3*.

Tabla 3 Tabla de parámetros de la búsqueda local

Parámetros de la búsqueda local	**Valor utilizado**
Intervalo de aplicación (gLs)	1 generación
Porcentaje de aplicación (probLs)	5%
Máximo de vecinos visitados (nLs)	100 vecinos por cada búsqueda

Tasa de recombinación

Para calibrar este parámetro se realizaron pruebas variando su valor desde 5% hasta 95%. Los resultados se encuentran en el Anexo C y se resumen en la Tabla *4*.

Tabla 4 Resumen de la calibración de la tasa de recombinación

Tasa de recombinación	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
5%	12.7376	13.7752	8.1473%	0.0045
15%	12.7423	13.7911	8.2324%	0.0036
25%	12.7790	13.8645	8.4967%	0.0048
35%	12.7879	13.8652	8.4265%	0.0051
45%	12.8022	13.8960	8.5449%	0.0039
55%	12.8085	13.9115	8.6127%	0.0040
65%	12.7922	13.9114	**8.7498%**	0.0039
75%	12.8028	13.8967	8.5460%	0.0045
85%	12.8215	13.9119	8.5057%	0.0046
95%	12.8044	13.9107	8.6414%	0.0044

En la Figura *22* se ve la variación del porcentaje de la mejora promedio según la tasa de recombinación.

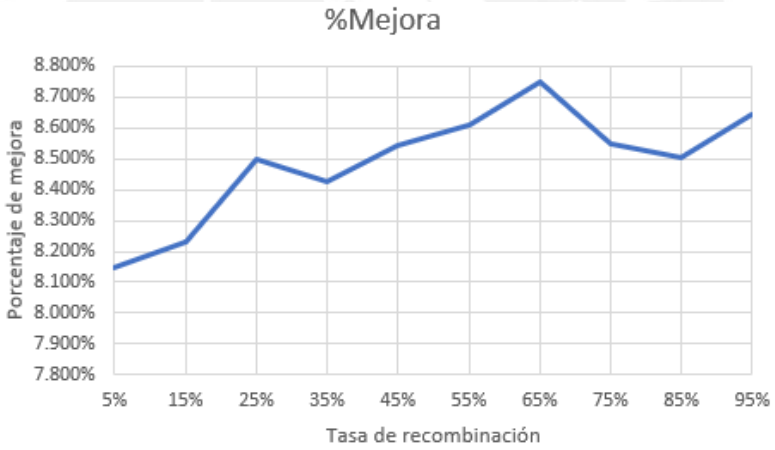

Figura 22 Porcentaje de mejora promedio por tasa de recombinación. Fuente: *Elaboración propia.*

Se tuvo el mayor porcentaje de mejora con una tasa de recombinación igual a 65%, por lo que se realizaron pruebas adicionales con valores cercanos para hallar el valor con el que se obtiene las mejores soluciones. Los resultados están resumidos en la Tabla *5*.

Tabla 5 Calibración de la tasa de recombinación (Fase dos)

Tasa de recombinación	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
60%	12.8227	13.9268	8.6124%	0.0044
65%	12.7922	13.9114	**8.7498%**	0.0039
70%	12.8170	13.9269	8.662%	0.0051

Los mejores resultados obtenidos siguen siendo aquellos en los que se usó el valor de 65% para la tasa de recombinación por lo que se fijó este valor.

Probabilidad de la recombinación uniforme

Para calibrar este parámetro se hicieron pruebas variando el valor de la probabilidad entre 20% y 80%. No se realizaron pruebas con valores fuera de este rango porque estos favorecen la generación de soluciones hijas bastante idénticas a las soluciones padre, lo que no es deseable porque no permite explorar todo el espacio de búsqueda. Los resultados están en el Anexo C, se resumen en la Tabla *6* y se ven en la Figura *23*.

Tabla 6 Resumen de la calibración de la probabilidad de la recombinación uniforme

Prob. de la recombinación uniforme	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
20%	12.8664	13.9683	8.5656%	0.0045
30%	12.8618	13.9638	8.5721%	0.0065
40%	12.8436	13.9396	8.5327%	0.0045
50%	12.7922	13.9114	8.7498%	0.0039
60%	12.8494	13.9593	8.5705%	0.0053
70%	12.8396	13.9660	**8.7738%**	0.0039
80%	12.8537	13.9697	8.6842%	0.0045

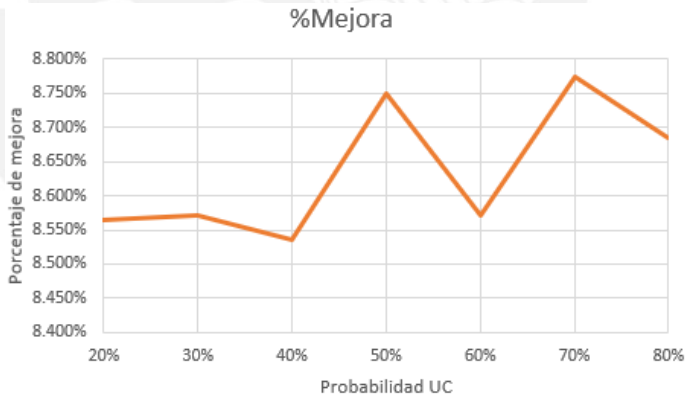

Figura 23 Porcentaje de mejora por probabilidad de recombinación uniforme. Fuente: *Elaboración propia.*

Se tuvo el mayor porcentaje de mejora con una probabilidad de recombinación uniforme igual a 70%, realizándose pruebas adicionales con valores cercanos para hallar el valor que brinda las mejores soluciones. Los resultados se encuentran en la Tabla *7*.Tabla *7* Calibración de la probabilidad de la recombinación uniforme (Fase dos)

Tabla 7 Calibración de la probabilidad de la recombinación uniforme (Fase dos)

Prob. de recombinación uniforme	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
65%	12.8614	13.9699	8.6209%	0.0046

70%	12.8396	13.9660	**8.7738%**	0.0039
75%	12.8635	13.9781	8.6663%	0.0052

Los mejores resultados obtenidos siguen siendo aquellos en los que se usó el valor de 70% para la probabilidad de recombinación uniforme por lo que se fijó este valor.

Tasa de mutación

Para calibrarla se probó variando su valor desde 4% hasta 10%. Los resultados se muestran en el Anexo C, se resumen en la Tabla 8 y se grafican en la Figura *24*.

Tabla 8 Resumen de la calibración de la tasa de mutación

Tasa de mutación	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
4%	12.8396	13.9660	8.7738%	0.0039
5%	12.9090	14.0389	8.7547%	0.0050
6%	12.9815	14.1869	**9.2880%**	0.0053
7%	12.9888	14.1871	9.2279%	0.0051
8%	13.0035	14.1872	9.1050%	0.0050
9%	12.9828	14.1872	9.2790%	0.0051
10%	12.9870	14.1871	9.2421%	0.0044

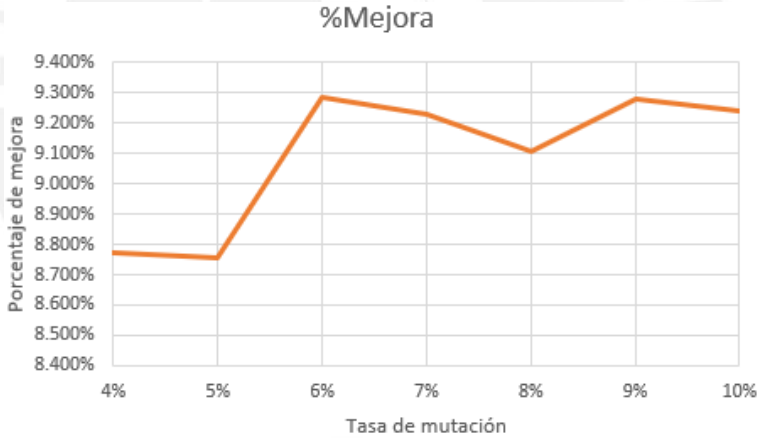

Figura 24 Porcentaje de mejora por tasa de mutación. Fuente: *Elaboración propia.*

Los mejores resultados se obtienen con una tasa de mutación igual al 6%, fijándose este valor para el parámetro.

Porcentaje de soluciones a conservar

Se recomienda que, al restaurar la población, solo se conserve un porcentaje pequeño del total de soluciones para que éstas no impulsen la convergencia de la nueva población hacia al mismo punto en que convergía la población inicial (Moscato & Cotta, 2003). Por ello, en la calibración se probó con valores desde 1% hasta 10%. Los resultados están en el Anexo C, se resumen en la Tabla *9* y se grafican en la Figura *25*.

Tabla 9 Resumen de la calibración del porcentaje de soluciones a conservar

Porcentaje a conservar	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
1%	13.0320	14.2306	9.1991%	0.0050
2%	13.0248	14.2305	9.2599%	0.0054
3%	13.0286	14.2307	9.2287%	0.0052
4%	13.0198	14.2308	9.3032%	0.0054
5%	12.9815	14.1869	9.2880%	0.0053
6%	12.9855	14.1872	9.2567%	0.0055
7%	13.0175	14.2309	**9.3230%**	0.0045
8%	13.0215	14.2318	9.2961%	0.0041
9%	13.0300	14.2315	9.2237%	0.0051
10%	13.0415	14.2316	9.1277%	0.0053

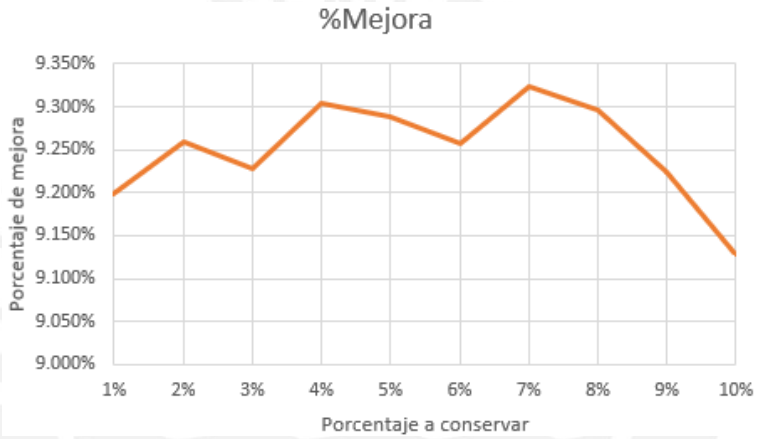

Figura 25 Porcentaje de mejora por porcentaje a conservar. Fuente: *Elaboración propia.*

El mayor porcentaje de mejora se tuvo con un valor de 7% que se fijó para el parámetro.

Máximo de generaciones sin mejora

Para calibrarlo se probó variando su valor desde 100 generaciones hasta 1000. Los resultados están en el Anexo C, se resumen en la Tabla *10* y se ven en la Figura 26.Figura *25*

Tabla 10 Resumen de la calibración del máximo de generaciones sin mejora

Máx. generaciones sin mejora	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Memético		
100	13.1193	14.3457	9.3503%	0.0049
200	13.1203	14.3470	9.3522%	0.0055
300	13.1274	14.3483	9.3037%	0.0059
400	13.1188	14.3499	9.3857%	0.0042
500	13.1161	14.3527	**9.4304%**	0.0051
600	13.1242	14.3517	9.3549%	0.0051
700	13.1182	14.3447	9.3508%	0.0039
800	13.1068	14.3421	9.4272%	0.0051
900	13.0994	14.3317	9.4094%	0.0042

1000	13.1005	14.3314	9.3978%	0.0047

Figura 26 Porcentaje de mejora por máximo número de it. sin mejora. Fuente: Elaboración propia.

Se obtuvo el mayor porcentaje de mejora con el valor 500 para el máximo número de iteraciones sin mejora. Por ello, se fijó este valor para el parámetro.

Capítulo 8. Algoritmo genético

8.1 Introducción

En este capítulo se presenta el algoritmo genético, describiendo brevemente las funciones que lo componen y, luego, se explica con más detalle las funciones encargadas del casamiento, mutación y depuración de la población.

8.2 Algoritmo genético

```
Algoritmo_Genetico(pob)
Inicio
 1    t ← Inicializar
 2    numIteracion ← 0
 3    sinMejora ← 0
 4    población ← convertirPoblacion(pob)
 5    mejorSol ← obtenerMejor(población)
 6    numSoluciones ← tamaño(población)
 7    Repetir
 8         casamiento(población, tasaCasamiento)
 9         mutación(población, tasaMutacion)
10         población ← depurarPoblacion(población, numSoluciones)
11         mejorSolActual ← obtenerMejor(población)
12         Si fitness(mejorSol)<fitness(mejorSolActual) entonces
13              mejorSol ← mejorSolActual
14              sinMejora ← 0
15         Si no
16              sinMejora ← sinMejora + 1
17         Fin Si
18         numIteracion ← numIteracion + 1
19    Hasta t > tiempoMaximo o numIteracion>MaxIteraciones o sinMejora>MaxSinMejora
20    Retornar(mejorSol)
Fin
```

Figura 27 Pseudocódigo del algoritmo genético. Fuente: *Elaboración propia.*

En la Figura *27*, se muestra a grandes rasgos el algoritmo genético. Este recibe como parámetro la población generada por el algoritmo GRASP, cuyos elementos luego serán convertidos a cromosomas (línea 4). A partir de ellos se generan nuevas soluciones mediante los procedimientos de casamiento (línea 8) y mutación (línea 9). Luego, se depura la población (línea 10) para que vuelva a su tamaño inicial. Después, se comprueba si se ha encontrado una solución que supere a la mejor actual (línea 12); si es así, se reemplaza la mejor actual (línea 13); en caso contrario, se aumenta el contador de iteraciones sin mejora (línea 16).

Las líneas 8 a la 18, se repiten hasta que se alcance una de las condiciones de parada (línea 19): agotar el tiempo asignado, superar el máximo de iteraciones o exceder el

número de iteraciones sin mejora, lo que indicaría que el algoritmo se ha estancado. Finalmente, se retorna la mejor solución encontrada (línea 20).

8.2.1 Casamiento

```
Casamiento(pob, tasaCasamiento)
Inicio
 1   numCasamientos ← tamaño(pob) * tasaCasamiento
 2   pobHijos ← Inicializar
 3   rangosRuleta ← prepararRuleta(pob) // devuelve rangos para la ruleta búsqueda binaria
 4   Para contador=0 hasta numCasamientos hacer
 5           padre1 ← ruleta(pob, rangos)
 6           padre2 ← ruleta(pob, rangos)
 7           hijos ← uniformCrossover(padre1, padre2)
 8           Para i=1 hasta 2 hacer
 9               Si esValido(hijos[i]) entonces
10                       AgregarSolucion(pobHijos, hijos[i])
11               Fin Si
12           Fin Para
13   Fin Para
14   juntarPoblaciones(pob, pobHijos)
Fin
```

Figura 28 Pseudocódigo del procedimiento casamiento. Fuente: *Elaboración propia.*

En la Figura *28* se detalla el pseudocódigo del procedimiento. Este recibe como parámetros la población y la tasa de casamiento, que sirve para calcular el numCasamientos que se realizarán (línea 1). Luego, se elabora el arreglo que tiene los límites superiores de los rangos que se emplearán en la ruleta binaria (línea 3).

Para la aplicación del casamiento, se selecciona a los padres mediante el método de ruleta con búsqueda binaria (líneas 5 y 6). Después, se cruzan las dos soluciones mediante el método de recombinación uniforme (línea 7). En la Figura *29*, se muestra el casamiento de dos soluciones para hornos de 3 vagonetas y 2 compartimentos.

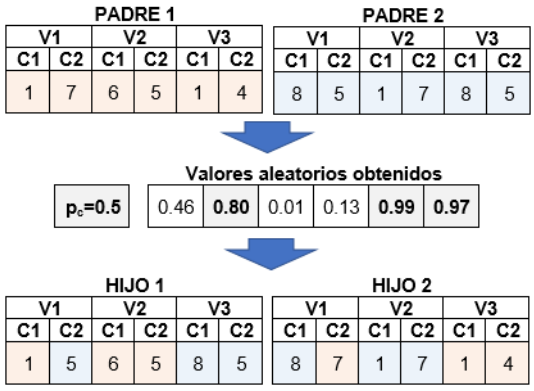

Figura 29 Operador de casamiento. *Fuente: Elaboración propia.*

Después, se verifica si cada una de las soluciones obtenidas es válida para que en caso lo sea se agregue a la población (línea 10). Este proceso (líneas 5 a la 12) se repite hasta completar el número de casamientos determinado al inicio. Finalmente, se agregan las soluciones generadas a la población actual (línea 14).

8.2.2 Mutación

```
Mutación(pob, tasaMutación)
Inicio
 1    numMutaciones ← tamaño(pob) * tasaMutación
 2    pobHijos ← Inicializar
 3    Para contador = 0 hasta numMutaciones hacer
 4            solucionAMutar ← seleccionarAlAzar(pob)
 5            nuevaSolucion ← mutar(solucionAMutar)
 6            Si esValida(nuevaSolucion) entonces
 7                AgregarSolucion(pobHijos, nuevaSolucion)
 8            Fin Si
 9    Fin Mientras
10    juntarPoblaciones(pob, pobHijos)
Fin
```

Figura 30 Pseudocódigo del procedimiento Mutación. Fuente: *Elaboración propia.*

En la Figura *30* se muestra el pseudocódigo de este procedimiento, que recibe como parámetros la población de soluciones y la tasaMutación; con este último dato, se determina el número de veces que se aplicará la mutación (línea 1).

Para aplicar la mutación, se selecciona al azar una solución de la población (línea 4) y se procede a mutarla (línea 5). La mutación consiste en elegir un compartimento al azar, quitar el elemento cargado y reemplazarlo por otra pieza que quepa. En la Figura *31*, vemos un ejemplo de mutación en el que se modifica la pieza cargada en el compartimento 1 de la vagoneta 1 (se cambia la pieza de código 1 por la de código 8).

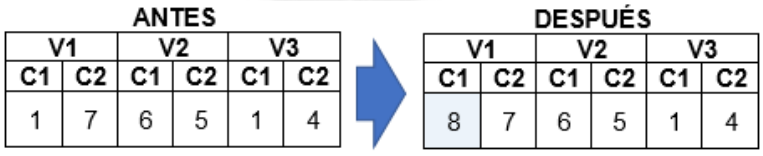

Figura 31 Operador de mutación. Fuente: *Elaboración propia.*

Una vez realizada la mutación, se comprueba si es válida la solución (línea 7) para poder agregarla a la población de hijos. Este proceso (líneas 5 a la 10) se repite hasta completar el número de mutaciones calculado al inicio. Finalmente, se agregan las soluciones de la población de hijos a la población original.

8.2.3 Depuración de la población

Este procedimiento (ver Figura *32*) se encarga de reducir la población, de manera que retorne a su tamaño inicial antes del proceso de casamiento y mutación.

Tiene dos etapas: primero, se extrae un porcentaje de las mejores soluciones de la población; y, después, se elige con el método de ruleta las soluciones necesarias para completar la población. De este modo se asegura conservar las mejores soluciones, así como, mantener una población variada y evitar converger en óptimos locales.

```
depurarPoblacion(pob, numSoluciones)
Inicio
 1    i ← 0
 2    nuevaPoblacion ← Inicializar
 3    cantConservar ← numSoluciones * %conservar
 4    Repetir
 5         solución ← extraerMejor(pob)
 6         AgregarSolucion(nuevaPoblación, solución)
 7         i ← i + 1
 8    Hasta i = cantConservar
 9    Repetir
10         solución ← ruleta(pob)
11         AgregarSolucion(nuevaPoblación, solución)
12         i ← i + 1
13    Hasta i=numSoluciones
14    Fin Si
15    Retornar(nuevaPoblacion)
Fin
```

Figura 32 Pseudocódigo de la función depurarPoblacion. Fuente: *Elaboración propia.*

8.3 Calibración del algoritmo genético

Para obtener mejores soluciones es necesario calibrar los parámetros: tasa de recombinación, probabilidad de la recombinación uniforme, tasa de mutación y el porcentaje de soluciones que se conservarán en la depuración de la población. Sin embargo, para los dos primeros parámetros se utilizarán los valores hallados en la sección Calibración del algoritmo Memético. Para los otros dos parámetros, se realizarán pruebas con diferentes valores, las cuales se repetirán 40 veces cada una.

Tasa de mutación

Para calibrarla se probó variando su valor desde 4% hasta 10%. Como resultado de estas pruebas, se obtuvieron los datos que se encuentran en el Anexo D, se resumen en la Tabla *11* y se grafican en la Figura *33*.

Tabla 11 Resumen de la calibración de la tasa de mutación

Tasa de mutación	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Genético		
4%	12.9046	14.0291	8.7167%	0.0056
5%	12.9893	14.1828	9.1910%	0.0054
6%	12.9839	14.1841	9.2460%	0.0047
7%	12.9777	14.1838	**9.2957%**	0.0050
8%	13.0005	14.1837	9.1046%	0.0064
9%	13.0066	14.1816	9.0362%	0.0051
10%	12.9793	14.1826	9.2734%	0.0054

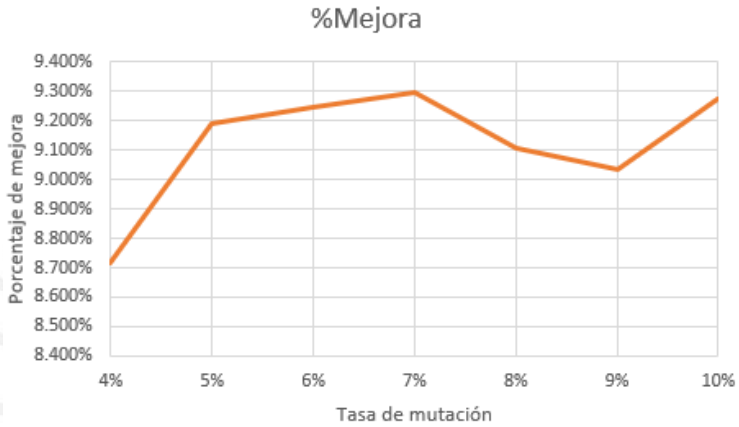

Figura 33 Porcentaje de mejora por tasa de mutación. Fuente: *Elaboración propia.*

Se obtuvo los mejores resultados con una tasa de mutación igual al 7%, por lo que se fijó este valor para el parámetro.

Porcentaje de soluciones a conservar

Para evitar que el algoritmo converja prematuramente, se decidió conservar un pequeño porcentaje de las mejores soluciones por lo que para calibrarlo se probó variando el valor de este parámetro entre 1% y 10%. Los resultados se muestran en el Anexo D, se resumen en la Tabla *12* y se grafican en la Figura *34*.

Tabla 12 Resumen de la calibración del porcentaje de soluciones a conservar

Porcentaje a conservar	Fitness promedio		Porcentaje de mejora	Desviación estándar
	Grasp	Genético		
1%	13.0349	14.0130	7.5065%	0.0057
2%	13.0351	14.1872	8.8400%	0.0048
3%	13.0448	14.2100	8.9354%	0.0055
4%	13.0061	14.2243	9.3685%	0.0050
5%	12.9777	14.1838	9.2957%	0.0050
6%	13.0254	14.2315	9.2618%	0.0057
7%	13.0263	14.2328	9.2657%	0.0062
8%	13.0108	14.2330	9.3957%	0.0045
9%	13.0234	14.2325	9.2861%	0.0046
10%	13.0067	14.2330	**9.4289%**	0.0028

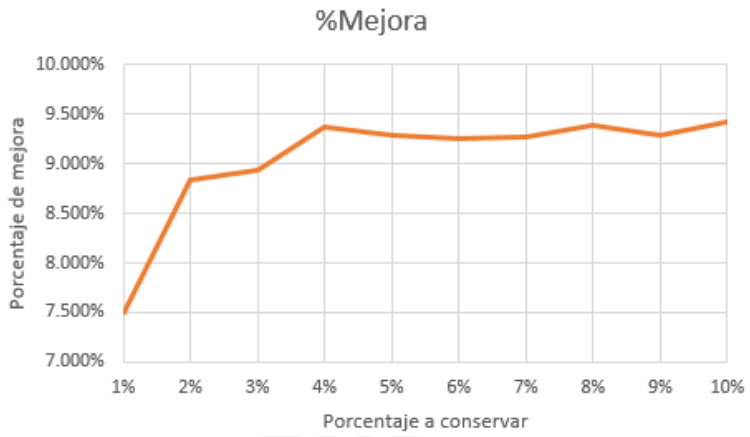

Figura 34 Porcentaje de mejora por porcentaje a conservar. Fuente: *Elaboración propia.*

Se obtuvo los mejores resultados con un porcentaje a conservar de 10% por lo que se fijó este valor para el parámetro.

Capítulo 9. Interfaz gráfica

9.1 Introducción

En este capítulo se explica cada una de las pantallas de la aplicación, que permiten la carga de datos, ejecución de los algoritmos y visualización de los resultados.

9.2 Pantalla de carga de datos

Desde ésta se cargarán los archivos .csv con los datos del horno, sets y pedidos.

El archivo con los datos del horno tendrá en su primera línea los siguientes 4 datos separados por comas: el volumen máximo (m^3), el peso máximo a cargar en las vagonetas (kg), el número de vagonetas y el número de compartimentos. En las siguientes líneas se especificará por cada compartimento: el identificador del compartimento (número entero), el ancho (m), largo (m) y alto (m).

El archivo de sets, productos y piezas estará conformado por 3 bloques. El primer bloque comenzará con la cantidad de sets que será cargada y después indicará por cada set los siguientes datos separados por comas: código, modelo, color, número de productos que lo componen, lista de productos (formada por los códigos de los productos separados por “/”, por ejemplo: 3/28/45) y la cantidad de sets terminados en el almacén.

El segundo bloque comenzará con la cantidad de productos que será cargada y luego listará los siguientes datos de los productos separados por comas: código, descripción, número de piezas que lo componen, lista de piezas (compuesta por los códigos de las piezas separados por “/”, por ejemplo: 3/28/45) y número de productos terminados en el almacén.

El tercer bloque comenzará con la cantidad de piezas que será cargada, luego por cada pieza detallará los siguientes datos separados por comas: código, tipo (tanque, taza, one piece, lavatorio o pedestal), modelo, color, alto (m), ancho (m), largo (m), peso (kg), cantidad pendiente por hornear y número de piezas terminadas en el almacén.

El último archivo, de pedidos, especificará en cada línea los siguientes datos por cada uno: el identificador del pedido, el identificador del set pedido, la cantidad pedida, la fecha de entrega en el formato dd/mm/aaaa y la prioridad del cliente.

Una vez seleccionado el archivo y especificado el tipo de datos que contiene (horno, sets o pedidos), se podrá cargar y ver los datos en el lado derecho de la pantalla (ver Figura *35*).

Después de haber subido los tres archivos requeridos, se habilitará el botón “Siguiente” el cual permite ir a la pantalla encargada de configurar los parámetros del algoritmo.

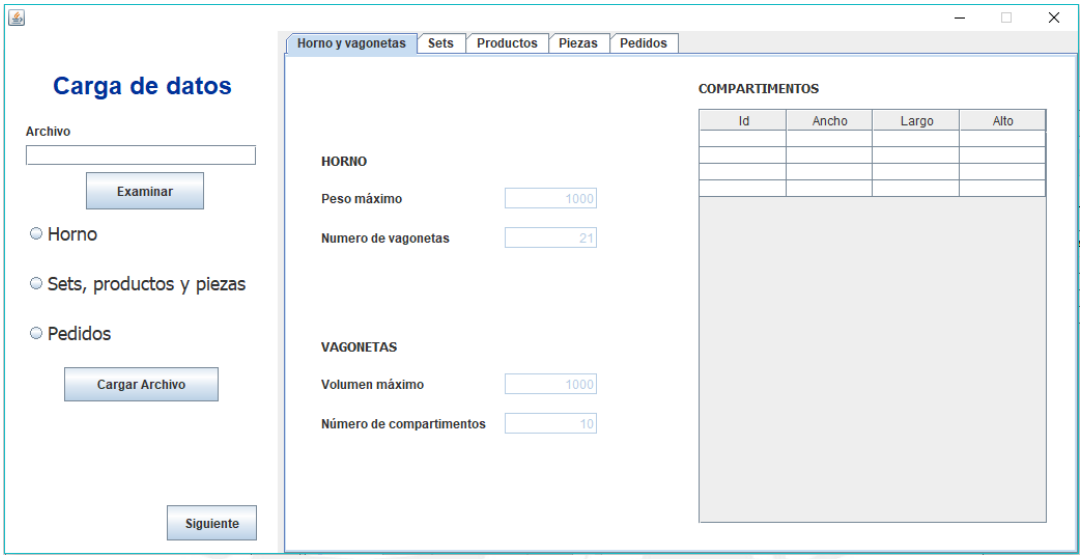

Figura 35 Pantalla de carga de datos. Fuente: *Elaboración propia.*

9.3 Pantalla de ejecución del algoritmo

En esta se configuran los parámetros de los algoritmos, permite iniciar su ejecución y ver el progreso de la misma.

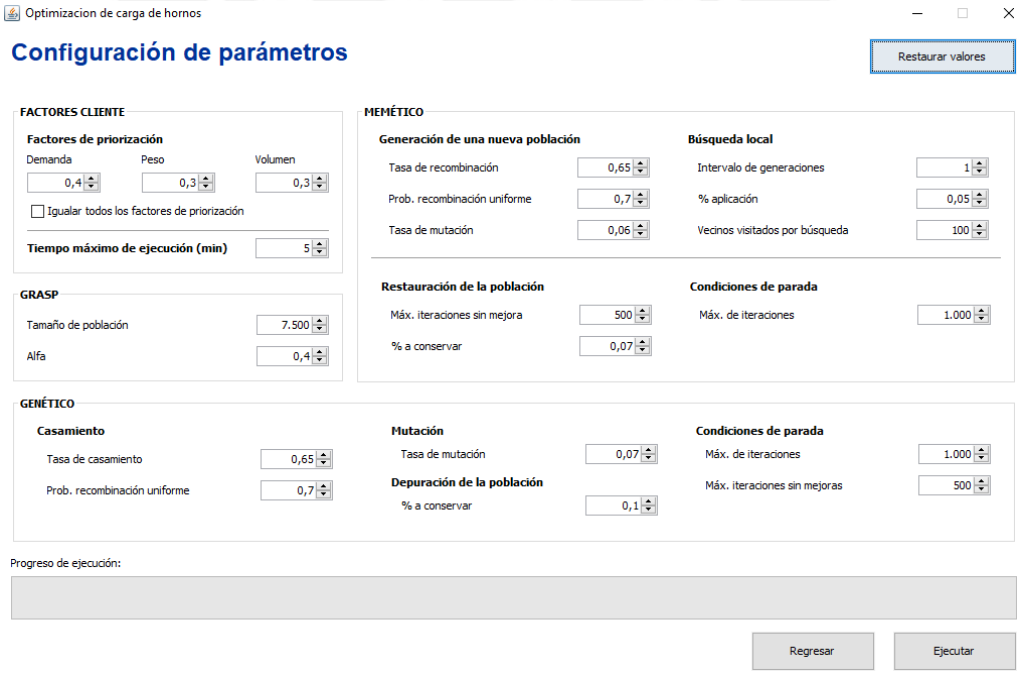

Figura 36 Pantalla de configuración de parámetros. Fuente: *Elaboración propia.*

En Factores cliente (ver Figura *36*,lado superior izquierdo) se especifica en qué medida se quiere considerar cada uno de los 3 factores que se toman en cuenta en el cálculo de la función objetivo (demanda, peso y volumen).

Los valores que se muestran por defecto para las variables del grasp, genético y memético son los resultados de la calibración, pero se podrán modificar si se desea.

Una vez que se haya especificado los valores de cada variable, se podrá ejecutar el algoritmo y su progreso se mostrará en la barra situada sobre los botones.

9.4 Pantalla de visualización de resultados

En ella se visualizan los resultados obtenidos por cada uno de los algoritmos; los resultados del memético, en el lado izquierdo y los del genético, en el derecho.

Si se selecciona la vista “Piezas asignadas” (ver Figura *37*), se mostrarán en las tablas las piezas asignadas a cada compartimento, donde cada columna representa una vagoneta; cada fila, un compartimento y cada celda contiene el código de la pieza asignada o se muestra vacía en caso no contenga ninguna pieza.

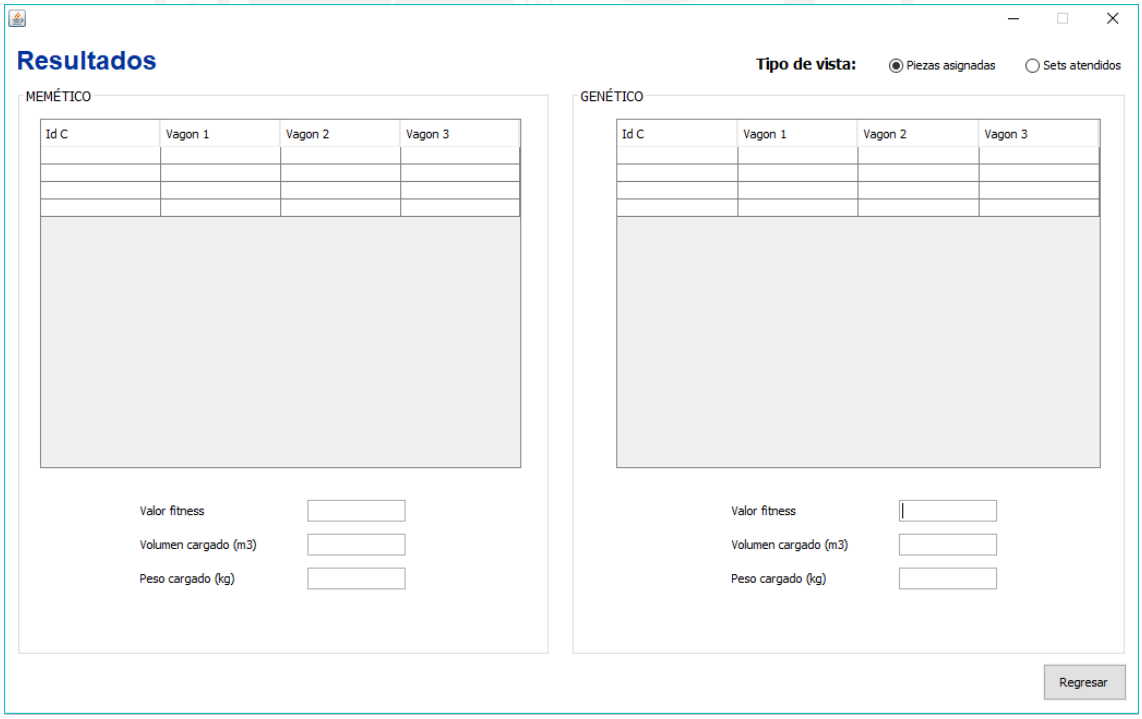

Figura 37 Pantalla de visualización de resultados: vista Piezas asignadas. Fuente: *Elaboración propia.*

Si se selecciona la vista “Sets atendidos” (ver Figura *38*), se mostrarán en las tablas los sets completados y sus cantidades.

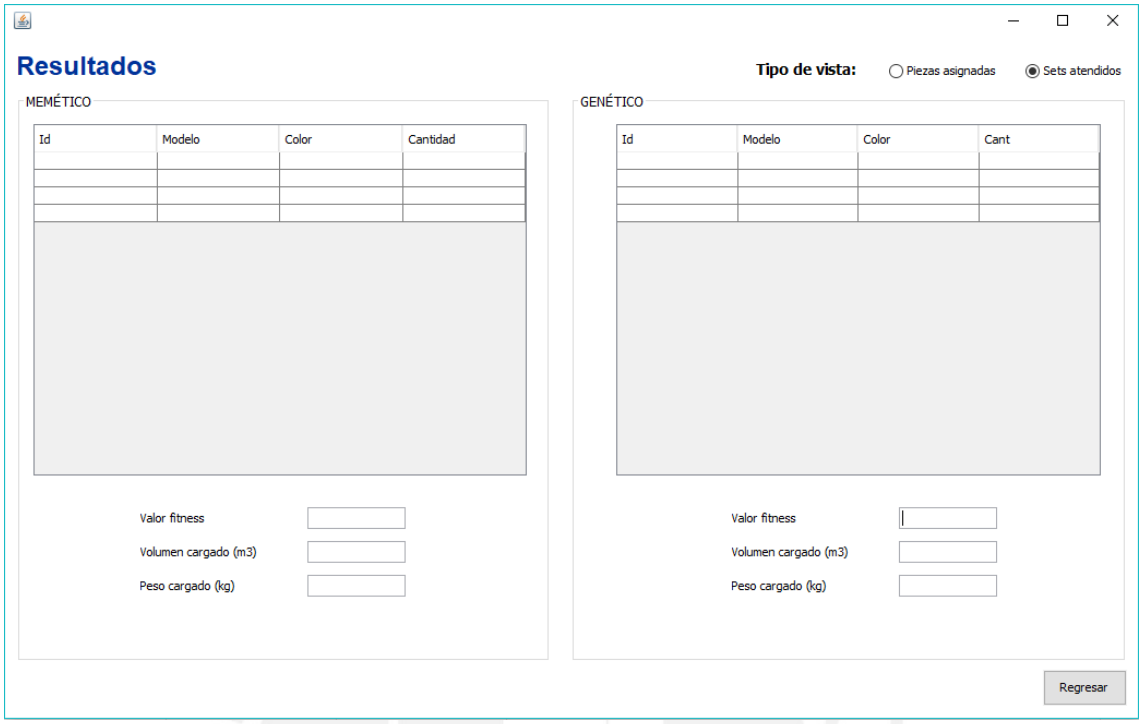

Figura 38 Pantalla de visualización de resultados: vista Sets atendidos. Fuente: *Elaboración propia.*

Capítulo 10. Experimentación numérica

10.1 Introducción

En este capítulo se desarrolla la experimentación numérica para determinar cuál algoritmo es una mejor alternativa de solución al problema de selección de piezas en la carga de hornos. Este capítulo se compone de una sección para cada una de las cuatro etapas: recolección de datos, prueba de Kolmogorov-Smirnov, prueba F de Fisher y prueba Z.

10.2 Recolección de datos

Para realizar las etapas posteriores es necesario contar con datos de prueba de ambos algoritmos, por ello se ejecutaron 40 pruebas con diferentes juegos de datos, que se repitieron 10 veces para obtener un valor promedio del desempeño de cada algoritmo.

En todas las pruebas se consideraron los mismos tipos de sets, productos y piezas, variando solo los archivos de pedidos. Estos últimos fueron generados al azar e incluyeron una variedad de pedidos tanto por la cantidad solicitada, la fecha de entrega y/o la prioridad del cliente. Los resultados se grafican en la Figura *39* (para mayor detalle revisar el Anexo E).

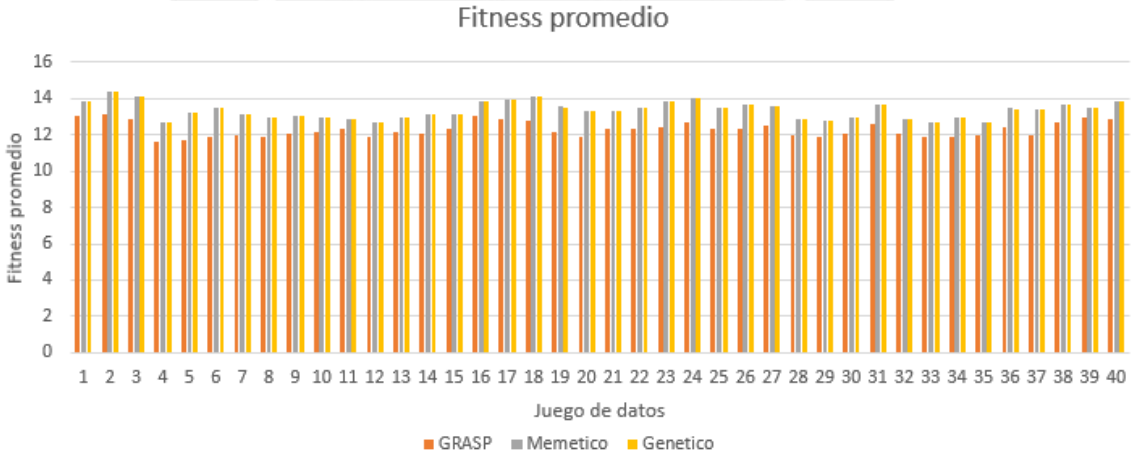

Figura 39 Representación de fitness promedio por muestra utilizada. Fuente: *Elaboración propia.*

10.3 Prueba de Kolmogorov-Smirnov

Se utilizó esta prueba para verificar si las muestras obtenidas siguen una distribución normal y así, poder realizar las pruebas posteriores.

Durante esta etapa se realizaron dos pruebas (una para el algoritmo memético y otra para el genético) y cada una de estas tuvo las siguientes hipótesis:

H_0: la muestra sigue una distribución normal

H_1: la muestra no sigue una distribución normal

El resultado obtenido de la prueba realizada al algoritmo Memético se muestra en la Figura *40* y se detalla en la Tabla *13*.

```
> ks.test(experimentos$Memetico,pnorm,mean(experimentos$Memetico),sd(experimentos$Memetico))

        One-sample Kolmogorov-Smirnov test

data:  experimentos$Memetico
D = 0.1172, p-value = 0.6006
alternative hypothesis: two-sided
```

Figura 40 Prueba K-S Memético. Fuente: *Elaboración propia.*

Tabla 13 Experimentación prueba K-S Memético

Resultados Memético	
Valor del estimador	0.1172
P-valor	0.6006
Grados de libertad	40
Significancia	0.05
Resultado	No se rechaza la hipótesis nula (H_0)

Como se ve en la Tabla *13*, el p-valor obtenido es mayor a la significancia por lo que no se rechaza la hipótesis nula y se concluye que la muestra sigue una distribución nomal.

Análogamente, se realizó la prueba para el algoritmo Genético, cuyos resultados se ven en la Figura *41* y se detallan en la Tabla *14*.

```
> ks.test(experimentos$Genetico,pnorm,mean(experimentos$Genetico),sd(experimentos$Genetico))

        One-sample Kolmogorov-Smirnov test

data:  experimentos$Genetico
D = 0.11683, p-value = 0.6044
alternative hypothesis: two-sided
```

Figura 41 Prueba K-S Genético. Fuente: *Elaboración propia.*

Tabla 14 Experimentación prueba K-S Genético

Resultados Genético	
Valor del estimador	0.11683
P-valor	0.6044
Grados de libertad	40
Significancia	0.05
Resultado	No se rechaza la hipótesis nula (H_0)

En este caso, se cumple que el p-valor es mayor a la significancia por lo que se mantiene la hipótesis nula y se concluye que la muestra sigue una distribución normal.

10.4 Prueba F de Fisher

Una vez que se verificó que ambas muestras siguen una distribución normal, se hizo la prueba F de Fisher para determinar si las muestras tienen varianzas significativamente diferentes o si son homogéneas.

Para esta prueba se plantearon las siguientes hipótesis:

H_0: las varianzas son significativamente homogéneas

H_1: las varianzas son significativamente diferentes

El resultado obtenido se muestra en la Figura *42* y se detalla en la Tabla *15*.

```
> var.test(experimentos$Memetico,experimentos$Genetico,alternative="two.sided")

        F test to compare two variances

data:  experimentos$Memetico and experimentos$Genetico
F = 1.002, num df = 39, denom df = 39, p-value = 0.9951
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.5299426 1.8944484
sample estimates:
ratio of variances
          1.001972
```

Figura 42 Prueba F de Fisher. Fuente: *Elaboración propia.*

Tabla 15 Experimentación Prueba F

	Memético	Genético
Media	13.35877	13.35571
Desviación estándar	0.45950	0.45905
Observaciones	40	40
Grados de libertad	39	39
F	1.002	
P-valor	0.9951	
Resultado	Se acepta la hipótesis nula (H_0)	

Se observa que el p-valor es mayor a la significancia (0.05) por lo que no se rechaza la hipótesis nula y se concluye que las varianzas de ambas muestras son homogéneas.

10.5 Prueba Z

Se procedió a realizar esta prueba, después de comprobar que las muestras siguen una distribución normal y tienen varianzas homogéneas.

Primero, se evaluó si la media de las muestras de ambos algoritmos era igual o no. Para ello se plantearon las hipótesis que se muestran a continuación.

H_0: la media del algoritmo memético es igual a la media del algoritmo genético

H_1: la media del algoritmo memético es diferente a la media del algoritmo genético

Como resultado se obtuvo lo siguiente:

```
> z.test(experimentos$Memetico,experimentos$Genetico,alternative="two.sided",mu=0,sigma.x=sd
(experimentos$Memetico),sigma.y=sd(experimentos$Genetico))

        Two-sample z-Test

data:  experimentos$Memetico and experimentos$Genetico
z = 0.029723, p-value = 0.9763
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.198230  0.204335
sample estimates:
mean of x mean of y
 13.35877  13.35571
```

Figura 43 Prueba Z. Fuente: *Elaboración propia.*

Tabla 16 Experimentación Prueba Z

	Memético	**Genético**
Media	13.35877	13.35571
Desviación estándar	0.45950	0.45905
Observaciones	40	40
Diferencia Hipotética de medias	0	
Z	0.029723	
P-valor	0.9763	
Resultado	Se acepta la hipótesis nula (H_0)	

Se aprecia que el p-valor es mayor a la significancia (0.05) lo que indica que la media de ambos algoritmos es la misma y que ambos dan soluciones de la misma calidad.

Pruebas adicionales

Como la diferencia entre las soluciones obtenidas no es significativa, se decidió realizar pruebas adicionales para ver el comportamiento de los algoritmos. En estas pruebas, se redujo el tiempo de ejecución de cada algoritmo de 10 minutos a 5 minutos.

Al comparar las soluciones obtenidas de los dos algoritmos y realizar la prueba Z, se obtuvo los resultados mostrados en la Figura *44*, concluyéndose que la diferencia entre las soluciones no es significativa ya que el p-valor es mayor a la significancia (0.05)

```
> z.test(experimentos$Memetico,experimentos$Genetico,alternative="two.sided",mu=0,sigma.x=sd(
experimentos$Memetico),sigma.y=sd(experimentos$Genetico))

        Two-sample z-Test

data:  experimentos$Memetico and experimentos$Genetico
z = 0.052473, p-value = 0.9582
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2047528  0.2160178
sample estimates:
mean of x mean of y
 13.48684  13.48121
```

Figura 44 Segunda prueba Z. Fuente: *Elaboración propia.*

Sin embargo, cuando el tiempo de ejecución es más corto, la diferencia entre el algoritmo memético y el genético es mayor (ver Figura *45*), siendo mejor el memético.

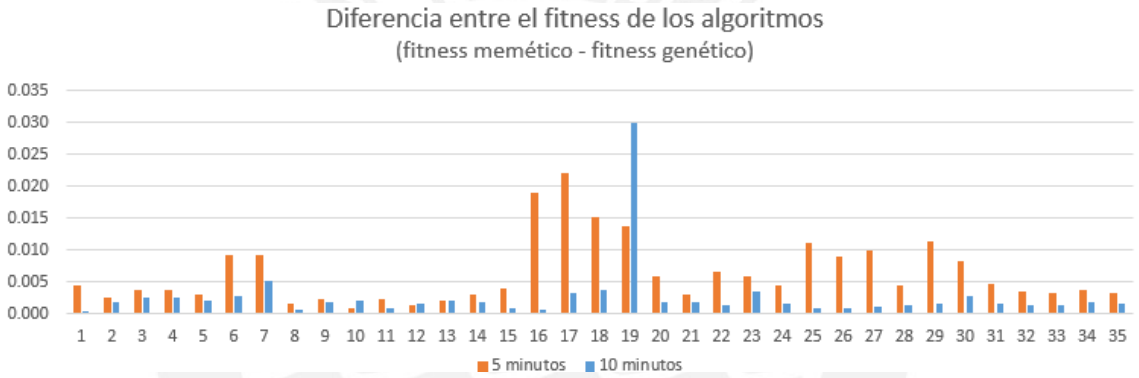

Figura 45 Diferencia entre el fitness de los algoritmos. Fuente: *Elaboración propia.*

En la medida que el tiempo de ejecución es más largo, la diferencia del fitness de las soluciones devueltas por los algoritmos se va reduciendo ya que ambos algoritmos se van acercando a un óptimo.

Capítulo 11. Conclusiones y trabajos futuros

11.1 Conclusiones

Durante este proyecto de tesis se desarrolló una alternativa de solución para el problema de la selección de piezas en la producción de sanitarios; que es un problema complejo, no solo por sus restricciones (peso máximo de las vagonetas, volumen máximo del horno, disponibilidad de piezas, capacidad y dimensiones de los compartimentos) sino también por los factores de priorización considerados (aprovechamiento de la capacidad de peso, volumen y la demanda de las piezas cargadas).

Los dos algoritmos diseñados e implementados en este proyecto generaron soluciones válidas, que fueron mejorando conforme se avanzaba cada una de las etapas de calibración. Como resultado de este proceso de calibración se determinaron los siguientes valores para los parámetros de los algoritmos:

ETAPA	PARÁMETRO	MEMÉTICO	GENÉTICO
Recombinación / Casamiento	Tasa de recombinación	65%	
	Probabilidad de la recombinación uniforme	70%	
Mutación	Tasa de mutación	6%	7%
Búsqueda local	Intervalo de generaciones	1	
	Tasa de aplicación de la búsqueda local	0.05	
	Número de vecinos a visitar	100	
Restauración / Depuración de la población	Porcentaje conservado	7%	10%
	Alfa restaurar	0.4	
	Máximo número de iteraciones sin mejora	500	

Una vez alcanzados los objetivos y después de hacer la experimentación numérica se concluyó que, aunque el algoritmo memético generaba mejores soluciones que el genético en los casos evaluados, la diferencia no es lo suficientemente significativa por lo que se considera que ambos algoritmos generan soluciones igualmente buenas.

11.2 Trabajos futuros

Entre los trabajos futuros que se recomiendan se encuentran:

- Calibración de los parámetros de la etapa de búsqueda local del algoritmo memético (intervalo de aplicación, tasa de aplicación y número de vecinos a visitar) para determinar si así se pueden obtener mejores soluciones que las actuales.
- Integrar el algoritmo desarrollado a un sistema de información para que ayude en una planta productiva.

Referencias

Alba, E., & Dorronsoro, B. (2005). The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE transactions on evolutionary computation*, *9*(2), 126–142.

ASALE, R.-. (s. f.). Diccionario de la lengua española - Edición del Tricentenario. Recuperado 11 de junio de 2018, a partir de http://dle.rae.es/?id=bGK0DGL

Avgustinik, A. I. (1983). *Cerámica*. Reverte.

Azurín, P. A. C., Timaná, L. A. C., & Pérez, J. L. T. (2018). Diagnóstico Operativo de la Corporación Cerámica S.A., 169.

Baiqing, Z., Haixing, L., Shaobu, B., Yifei, T., & Fei, H. (2016). Study on the charging combination optimization for forging production based on discrete shuffled frog leaping algorithm. *Mechanics*, *22*(5), 425–431.

Baesler, F., & Palma, C. (2014). Multiobjective parallel machine scheduling in the sawmill industry using memetic algorithms. *International Journal of Advanced Manufacturing Technology*, *74*(5-8), 757-768. https://doi.org/10.1007/s00170-014-5957-6

Berenson, M. L., & Levine, D. M. (2006). *Estadística para administración*. Pearson Educación.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*(3), 268–308.

Camargo, V. C. B., Mattiolli, L., & Toledo, F. M. B. (2012). A knapsack problem as a tool to solve the production planning problem in small foundries. *Computers & Operations Research*, *39*(1), 86-92. https://doi.org/10.1016/j.cor.2010.10.023

Ceramic Industry. (2015, mayo 1). Case Study: Collaborative Robots in Technical Ceramic Parts Production. Recuperado 21 de mayo de 2018, a partir de https://www.ceramicindustry.com/articles/94688-case-study-collaborative-robots-in-technical-ceramic-parts-production

CERAMIFOR Kilns & Equipment. (s. f.). Hornos túneles. Recuperado 11 de junio de 2018, a partir de http://www.ceramifor.com/es/Ceramics/Hornos/Hornos-tuneles-D16

Chen, Y., & Hao, J.-K. (2016). Memetic Search for the Generalized Quadratic Multiple Knapsack Problem. *IEEE Transactions on Evolutionary Computation*, *20*(6), 908-923. https://doi.org/10.1109/TEVC.2016.2546340

Comain. (s. f.). Comain: Carros y Maquinaria Cerámica. Recuperado 11 de junio de 2018, a partir de http://comain.es/m/index.php

Corporate Munim. (s. f.). Ceramics ERP. Recuperado 21 de mayo de 2018, a partir de http://www.corporatemunim.com/Ceramics

Crawley, M. J. (2005). *Statistics: An Introduction using R*. John Wiley & Sons.

Datoussaid, S., Verlinden, O., & Conti, C. (2002). Application of Evolutionary Strategies to Optimal Design of Multibody Systems. *Multibody System Dynamics*, *8*(4), 393-408. https://doi.org/10.1023/A:1021101912826

Deb, K. (2004). Introduction to Genetic Algorithms for Engineering Optimization. En *New Optimization Techniques in Engineering* (pp. 13-51). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39930-8_2

Díaz, Miguel (2004). *Estudio de información del proceso productivo de la corporación Cerámica S.A*.

Dorta, I., León, C., Rodríguez, C., Rodríguez, G., & Rojas, A. (2003). Complejidad Algorítmica: de la Teoría a la Práctica. *III Jornadas de Enseñanza Universitaria de Informática*.

Duda, J., & Stawowy, A. (2013). Optimization methods for lot-sizing problem in an automated foundry. *Archives of Metallurgy and Materials*, *58*(3), 863–866.
Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*(2), 109–133.
Fuentes-Penna, A., Vélez-Díaz, D., Moreno-Gutiérrez, S., Martínez-Cervantes, M. A., & Sánchez-Muñoz, O. (2015). Problema de la mochila (Knaspack problem). *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, *3*(6).
García, R., González, J., & Jornet, J. (2010). Introduccion al SPSS, Universitat de Valencia. Recuperado 26 de junio de 2018, a partir de https://www.uv.es/innomide/spss/noparam.wiki
Gen, M., & Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. John Wiley & Sons.
Gómez Gutiérrez, C. (2010). *Modelamiento y simulación de un horno túnel industrial*. Universidad Nacional de Colombia, Medellín. Recuperado a partir de http://www.bdigital.unal.edu.co/1882/1/71265369.2010.pdf
He, D., & Hong, Y. (2015). An Improved Tabu Search Algorithm Based on Grid Search Used in the Antenna Parameters Optimization [Research article]. https://doi.org/10.1155/2015/947021
Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press.
IBM. (s. f.). CPLEX Optimizer | IBM Analytics. Recuperado 19 de junio de 2018, a partir de https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer
IBM. (2018, junio 19). IBM ILOG CPLEX Optimization Studio - Visión general - España. Recuperado 19 de junio de 2018, a partir de https://www.ibm.com/es-es/marketplace/ibm-ilog-cplex
Ishibuchi, H., Tanigaki, Y., Akedo, N., & Nojima, Y. (2013). How to strike a balance between local search and global search in multiobjective memetic algorithms for multiobjective 0/1 knapsack problems. En *2013 IEEE Congress on Evolutionary Computation* (pp. 1643-1650). https://doi.org/10.1109/CEC.2013.6557758
Kniberg, H., & Skarin, M. (2010). *Kanban and Scrum - Making the Most of Both*. Lulu.com.
Koblasa, F., Vavrousek, M., & Manlig, F. (2017). Three-dimensional Bin Packing Problem with heterogeneous batch constraints.
Krasnogor, N., & Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, *9*(5), 474–488.
Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, *391*(6), 2193-2196. https://doi.org/10.1016/j.physa.2011.12.004
Liu, Y., Pan, Q. k, & Chai, T. (2015). Magnetic Material Group Furnace Problem Modeling and the Specialization of the Genetic Algorithm. *IEEE Transactions on Engineering Management*, *62*(1), 51-64. Recuperado 28 de mayo de 2018, a partir de https://doi.org/10.1109/TEM.2014.2370392
Magalhaes-Mendes, J. (2013). A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem. *WSEAS transactions on computers*, *12*(4), 164–173.
Meurant, G. (2014). *Algorithms and Complexity*. Elsevier.
Michalewicz, Z. (2013). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Science & Business Media.

Microsoft. (2016, agosto). Microsoft Dynamics NAV (ERP for Ceramic). Recuperado 20 de mayo de 2018, a partir de https://mqdynamicsblog.files.wordpress.com/2016/08/navtile.pdf

Misthal, B., Mueller, M., Bono, R., & Pillsbury, S. (s. f.). 2017 Industrial Manufacturing Trends. Recuperado 21 de mayo de 2018, a partir de https://www.strategyand.pwc.com/trend/2017-industrial-manufacturing-trends

Moore, D. S. (2005). *Estadística aplicada básica*. Antoni Bosch editor.

Moscato, P., & Cotta, C. (2003). A gentle introduction to memetic algorithms. En *Handbook of metaheuristics* (pp. 105–144). Springer.

Neri, F., & Cotta, C. (2012). A Primer on Memetic Algorithms. En *Handbook of Memetic Algorithms* (pp. 43-52). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23247-3_4

NetBeans. (s. f.). NetBeans IDE - Overview. Recuperado 26 de junio de 2018, a partir de https://netbeans.org/features/index.html

Obbink, H. (2005). *Software Product Lines: 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*. Springer Science & Business Media.

Oracle. (s. f.). The Java Language Environment. Recuperado 26 de junio de 2018, a partir de http://www.oracle.com/technetwork/java/intro-141325.html

Oracle. (2018, febrero 20). The Java Language Specification (Java SE 10 Edition). Recuperado 26 de junio de 2018, a partir de https://docs.oracle.com/javase/specs/jls/se10/html/jls-1.html

Osman, I. H., & Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, *63*(5), 511-623. https://doi.org/10.1007/BF02125421

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation.

Porras, L. F. T. (2018). Propuesta de mejora de una empresa de producción de sanitarios y accesorios de baño en Lima Metropolitana, 97.

Ramezanian, R., Sanami, S. F., & Nikabadi, M. S. (2017). A simultaneous planning of production and scheduling operations in flexible flow shops: case study of tile industry. *The International Journal of Advanced Manufacturing Technology*, *88*(9-12), 2389-2403. https://doi.org/10.1007/s00170-016-8955-z

Rhodes, D. (2004). *Hornos para ceramistas*. CEAC.

SACMI. (2005, enero 3). E.I.D. Parry (India) Ltd. now firing with a Sacmi tunnel kiln (325×350). Recuperado 21 de mayo de 2018, a partir de http://www.sacmi.com/System/00/01/16/11690/532632452944663593750.jpg

Sarmiento, A., Vladimir, R., & Reinoso Avecillas, F. (2012). Diseño de un horno-túnel para planta procesadora de arcilla “Bella Azhuquita”.

Wang, K., Ma, W. q., Luo, H., & Qin, H. (2016). Coordinated scheduling of production and transportation in a two-stage assembly flowshop. *International Journal of Production Research*, *54*(22), 6891-6911. https://doi.org/10.1080/00207543.2016.1193246