PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DEL ALGORITMO METAHEURÍSTICO CUCKOO SEARCH PARA LA OPTIMIZACIÓN DE CORTES EN DOS DIMENSIONES DE PRODUCTOS CERÁMICOS CON DEFECTOS PARA LA PRODUCCIÓN DE PIEZAS DECORATIVAS

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

Javier Alexander Monzón Durand

Asesor:

Mag. Rony Cueva Moscoso

Lima, abril de 2019

Resumen

Los residuos generados por los cortes de cerámicos son uno de los principales factores de desperdicio en la industria de baldosas y cerámicos, el cual se estima en una pérdida de alrededor 40% del material cerámico utilizado. Por este motivo, la reducción de los residuos de materiales utilizados en la fabricación de los productos cerámicos es una parte fundamental para la reducción de costos de producción. Asimismo, es importante mencionar que en esta industria es posible encontrar defectos en el material a recortar, una restricción de la cual carecen la mayoría de investigaciones que abordan el problema.

Seleccionar el ordenamiento con menor desperdicio de las piezas a recortar, en términos de complejidad computacional, se considera como un problema del tipo NP-difícil (polinómico no determinístico), el cual toma mucho tiempo para encontrar una solución exacta y lo hace inviable de aplicar en la industria. Es por ello que se justifica el uso de métodos heurísticos para obtener aproximaciones a la solución óptima en un tiempo menor. El presente trabajo de fin de carrera presenta una metaheurística Cuckoo Search para resolver el problema de corte de material expuesto como alternativa de solución al algoritmo genético, muy utilizado en este tipo de problemas de optimización. El algoritmo Cuckoo Search es una técnica de reciente desarrollo y ha mostrado buen desempeño en otro tipo de problemas de optimización y hasta el momento no se ha intentado atacar el problema usan esta metaheurística.

Para medir el desempeño del algoritmo Cuckoo Search, se hace uso de una adaptación del algoritmo genético encontrado en la literatura para la misma variante del problema de corte de material. El algoritmo genético es utilizado en este trabajo para comparar el desempeño del algoritmo Cuckoo Search propuesto mediante una experimentación numérica. Se concluye que el algoritmo genético tiene mejor desempeño que el algoritmo Cuckoo Search para el conjunto de datos utilizado en el proyecto, sin embargo, los resultados obtenidos de este último siguen siendo prometedores para ser utilizado por las empresas de la industria de cerámicos.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades

1.1 Problemática

En la última década, el Perú ha presenciado un crecimiento en el sector construcción. Según el reporte anual del PBI elaborado por el INEI (ver Figura 1), desde el año 2006 este sector viene presentando un crecimiento importante. Si vemos estadísticas de los últimos años (INEI, 2017), este crecimiento ha ido disminuyendo, sin embargo, instituciones como la Cámara de Comercio de Lima prevén que para el año 2018 crecería un 3.9% (CCL 2018). Además, este es un sector que suele ser impulsado por el Estado, hecho que vimos cuando suceden eventos como el Fenómeno del Niño del año 1998 y el terremoto en el sur del país en el año 2001, donde el Estado realizó gastos en la reconstrucción (UDEP, 2015).

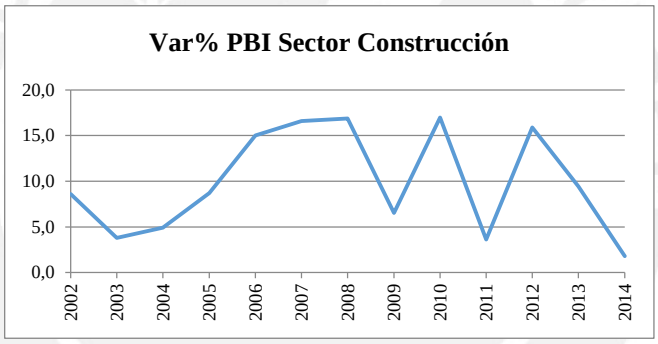


Figura 1. Variación del PBI en el sector construcción

Elaborado a partir de INEI (2017): Producto Bruto Interno según Actividad Económica

Entre las industrias que varían directamente con el sector construcción encontramos a las productoras de cementos, ladrillos de construcción, entre otras. Una que presentó un gran crecimiento es la industria de baldosas y cerámicos, que en el 2014 creció 4.9% (SNI, 2015). Para el mismo año también se vio un incremento en las exportaciones de cerámicos en alrededor de 762% (INEI, 2015). Este crecimiento ha permitido que muchas de las empresas del rubro puedan centrar parte de sus esfuerzos en investigar en factores que le permitan optimizar su producción.

Uno los factores para la optimización de la producción es la reducción de los residuos de materiales utilizados en la fabricación de los productos cerámicos. Entre los productos más importantes tenemos a los listelos, tacos e insertos, que son piezas decorativas que se utilizan en pisos, paredes, baños, cocinas o piscinas y provienen

del corte de cerámicos de mayores dimensiones (ver Figura 3, Figura 4 y Figura 5, respectivamente). Según un estudio realizado en la Universidad Politécnica de Hong Kong, la pérdida generada por el corte en la industria de cerámicos es alrededor del 40%, siendo esta la principal causa de desperdicio de material (Galarza, 2011). De ahí la importancia que tiene el problema de minimizar la generación de los residuos cerámicos derivados del corte.

El problema presentado, en su forma general, es conocido en la literatura como problema de corte de material *(cutting stock problem)*. Se trata de encontrar el mejor ordenamiento de las piezas a recortar en materiales disponibles de mayor dimensión de modo tal que el material que ya no pueda utilizarse para otras piezas sea mínimo. Sin embargo, la cantidad de posibles ordenamientos para este problema, incluso en instancias pequeñas, suele ser muy grande, lo que hace intratable un análisis de todas las posibles soluciones para luego escoger a la más óptima.

Si bien este problema presenta muchas variaciones en la literatura (Dyckhoff, 1990), una de las características más importantes es la dimensionalidad, que generalmente es de una o dos dimensiones. Al ser el recorte de material cerámico en forma rectangular, es del tipo bidimensional. Además, en condiciones reales se presentan otras restricciones como el tipo de corte permitido (Wäscher, Haußner, & Schumann, 2007). Debido a que estos cortes en la industria se realizan usualmente con máquinas, solo están permitidos cortes de extremo a extremo y paralelo a los lados (corte de guillotina). Otra característica que se presenta es que muchos de los cerámicos a cortar presentan fallas, por lo que se debe tener en cuenta que el producto recortado no presente estas regiones. Además, mucho de este material cerámico es reciclado, por lo que no esperamos encontrar cerámicos perfectos.

En términos de complejidad computacional, este tipo de problemas son conocidos como NP-difícil (polinómico no determinístico) y las soluciones planteadas para estos problemas no son exactas, pues toman mucho tiempo para resolverse. Es por esta razón que se realizan aproximaciones a la solución óptima a través de métodos aproximados, que proveen soluciones en tiempos razonables.

Entre los métodos aproximados nos encontramos a los algoritmos heurísticos y metaheurísticos. Los algoritmos heurísticos son estrategias que proveen una solución basada en un criterio voraz en el que se restringe el espacio de soluciones a analizar, sin embargo, pueden traer como consecuencia que las soluciones obtenidas puedan

ser buenas solo para el espacio restringido (Martı, 2003). Esto representa una dificultad, pues la solución obtenida puede estar muy alejada de la mejor, lo cual, para el caso del problema expuesto en este proyecto, generaría pérdidas en la empresa debido al desperdicio de cerámico generado. Por otro lado, están los algoritmos metaheurísticos los cuales coordinan técnicas heurísticas generales con determinadas estrategias inteligentes, muchas veces inspiradas en la naturaleza, con el fin de analizar un espacio de soluciones mayor y obteniendo como resultado, generalmente, una solución de mejor calidad que los algoritmos heurísticos (Martı, 2003). Es por ello que se prefiere el uso de algoritmos metaheurísticos sobre los heurísticos para resolver problemas de optimización, como el expuesto en el presente proyecto.

Uno de los algoritmos metaheurísticos más utilizados para los problemas de optimización son los algoritmos genéticos (Gen & Cheng, 2000). Sin embargo, como mencionan en (Beasley, Bull, & Martin, 1993), los algoritmos genéticos obtienen una buena solución pero toman mucho tiempo para converger por lo que se consideran lentos, además que su implementación suele ser compleja. Existen otros algoritmos metaheurísticos que han obtenido un buen rendimiento para diversos problemas de optimización, como el diseñado por los profesores Yang y Deb en 2009, llamado algoritmo Cuckoo Search (Yang & Deb, 2009). Se puede clasificar como un algoritmo de inteligencia de “enjambre” y una de principales ventajas es la velocidad con la que obtiene buena solución y su simplicidad de la implementación, en comparación con otros algoritmos. (Du & Swamy, 2016)

En conclusión, como se mencionó anteriormente, los residuos generados por los cortes de cerámicos son la principal fuente desperdicio en esta industria y para la solución del problema expuesto se utilizan métodos aproximados con el fin de obtener una solución que minimice dicho desperdicio en un tiempo razonable, pues lo contrario representaría pérdidas en la empresa. Este problema será abordado por medio de un algoritmo metaheurístico para obtener una buena solución en un tiempo aceptable con el fin de ser utilizado en ambientes reales de producción. En particular, se utilizará el algoritmo Cuckoo Search, técnica diseñada en los últimos años e inspirada en los algoritmos de inteligencia de partículas (Yang & Deb, 2009). Se parte de la premisa que generará soluciones de mejor calidad que las brindadas por otro algoritmo metaheurístico empleado para resolver este problema según la revisión bibliográfica realizada, en este caso el algoritmo genético.

Para ello, se compararán los resultados de los dos algoritmos: el Cuckoo Search contra el genético. Esto debido a que hasta la actualidad se siguen proponiendo optimizaciones de los algoritmos genéticos para problemas de optimización, entre ellos el de corte de material (Pravesjit & Kantawong, 2017), y se ha optado muy poco por la utilización de otras técnicas de reciente desarrollo, como el algoritmo Cuckoo Search, y que podrían proporcionar mejores resultados. Esta comparación se realizará a través de una experimentación numérica empleando diferentes muestras. Se espera que este proyecto de fin de carrera ayude a las empresas a optimizar la producción de piezas cerámicas como lístelos, tacos e insertos a través de la minimización del desperdicio que se genera al realizar cortes sobre cerámicos rectangulares de mayor dimensión, tomando en cuenta restricciones como presencia de fallas en dichos cerámicos y el corte tipo guillotina permitido.

1.2 Objetivos

1.2.1 Objetivo general

Implementar un algoritmo metaheurístico Cuckoo Search para resolver el problema de corte de piezas cerámicas recicladas o con fallas de forma regular.

1.2.2 Objetivos específicos

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

O 2. Diseñar un algoritmo Cuckoo Search como alternativa de solución al problema de corte tipo guillotina de material cerámico rectangular con defectos.

O 3. Adaptar un algoritmo genético como alternativa de solución al problema de corte tipo guillotina de material cerámico rectangular con defectos.

O 4. Implementar los algoritmos Cuckoo Search y genético en un lenguaje de programación.

O 5. Desarrollar la experimentación numérica para comparar el desempeño de los algoritmos Cuckoo Search y genético.

1.2.3 Resultados esperados

R 1. Estructuras de datos a usar en los algoritmos Cuckoo Search y genético. (O 1)

R 2. Función objetivo que mida la pérdida generada por el corte de un céramico. (O 1)

R 3. Estructuras de datos que soporten la implementación del algoritmo Cuckoo Search. (O 2)

R 4. Algoritmo Cuckoo Search diseñado. (O 2)

R 5. Estructuras de datos que soporten la implementación del algoritmo genético. (O 3)

R 6. Algoritmo genético adaptado al problema. (O 3)

R 7. Módulo para la carga de pedidos de piezas a recortar, considerando sus dimensiones (ancho y alto), cantidad y las regiones defectuosas en stock. (O 4)

R 8. Módulo del algoritmo Cuckoo Search implementado. (O 4)

R 9. Módulo del algoritmo genético implementado. (O 4)

R 10. Calibración de variables de la experimentación numérica. (O 5)

R 11. Interfaz para mostrar de manera gráfica los resultados obtenidos por el algoritmo Cuckoo Search y el algoritmo genético. (O 5)

R 12. Informe de experimentación numérica en el que se detalle las pruebas realizadas, sus resultados e interpretación de estos. (O 5)

1.2.4 Mapeo de objetivos, resultados y verificación

En esta sección se detallan los resultados esperados y medios de verificación a usar en cada uno de los objetivos específicos del proyecto.

Tabla 1. Mapeo del objetivo 1.

O 1. Definir la función objetivo a ser usada en los algoritmos Cuckoo Search y genético.		
Resultado	**Meta física**	**Medio de verificación**
R 1. Estructuras de datos a usar en los algoritmos Cuckoo Search y genético.	Software	• Representación de soluciones con las estructuras propuestas.
R 2. Función objetivo que mida la pérdida generada por el corte de un céramico.	Software	• Juicio experto para determinar la efectividad de la función. • Implementación de la función objetivo en un lenguaje de programación.

Tabla 2. Mapeo del objetivo 2.

O 2. Diseñar un algoritmo Cuckoo Search como alternativa de solución al problema de corte tipo guillotina de material cerámico rectangular con defectos.		
Resultado	**Meta física**	**Medio de verificación**
R 3. Estructuras de datos que soporten la implementación del algoritmo Cuckoo Search.	Software	• Representación de soluciones del algoritmo Cuckoo Search con las estructuras propuestas.
R 4. Algoritmo Cuckoo Search diseñado.	Software	• Pseudocódigo del algoritmo y funciones auxiliares.

Tabla 3. Mapeo del objetivo 3.

O 3. Adaptar un algoritmo genético como alternativa de solución al problema de corte tipo guillotina de material cerámico rectangular con defectos.		
Resultado	**Meta física**	**Medio de verificación**
R 5. Estructuras de datos que soporten la implementación del algoritmo genético	Software	• Representación de soluciones del algoritmo genético con las estructuras propuestas.
R 6. Algoritmo genético adaptado al problema.	Software	• Pseudocódigo del algoritmo y funciones auxiliares.

Tabla 4. Mapeo del objetivo 4.

O 4. Implementar los algoritmos Cuckoo Search y genético en un lenguaje de programación.		
Resultado	**Meta física**	**Medio de verificación**
R 7. Módulo para la carga de pedidos de piezas a recortar, considerando sus dimensiones (ancho y alto), cantidad y las regiones defectuosas en stock.	Software	• Verificación del contenido de los archivos con los datos de entrada del problema.
R 8. Módulo del algoritmo Cuckoo Search implementado.	Software	• Código de la implementación del algoritmo y funciones

		auxiliares. • Salida del algoritmo debe ser una solución al problema.
R 9. Módulo del algoritmo genético implementado.	Software	• Código de la implementación del algoritmo y funciones auxiliares. • Salida del algoritmo debe ser una solución al problema.

Tabla 5. Mapeo del objetivo 5.

O 5. Desarrollar la experimentación numérica para comparar el desempeño de los algoritmos Cuckoo Search y genético.		
Resultado	**Meta física**	**Medio de verificación**
R 10. Calibración de variables de la experimentación numérica.	Documento	• Mejores resultados del algoritmo con los parámetros escogidos en comparación con otros valores para los parámetros del algoritmo.
R 11. Interfaz para mostrar de manera gráfica los resultados obtenidos por el algoritmo Cuckoo Search y el algoritmo genético.	Software	• Ventana indicando de forma gráfica el ordenamiento de las piezas que serán recortadas.
R 12. Informe de experimentación numérica en el que se detalle las pruebas realizadas, sus resultados e interpretación de estos.	Documento	• Informe de experimentación numérica.

1.3 Herramientas y Métodos

1.3.1 Introducción

A continuación, se presentarán las herramientas, métodos y procedimientos a usar en el presente proyecto de fin de carrera, las cuales son especificadas en las siguientes secciones.

Tabla 6. Herramientas, métodos y procedimientos

N°	Resultado	Herramientas o metodologías a usar
R 1.	Estructuras de datos a usar en los algoritmos Cuckoo Search y genético.	
R 2.	Función objetivo que mida la pérdida generada por el corte de un céramico.	
R 3.	Estructuras de datos que soporten la implementación del algoritmo Cuckoo Search.	
R 4.	Algoritmo Cuckoo Search diseñado.	
R 5.	Estructuras de datos que soporten la implementación del algoritmo genético.	
R 6.	Algoritmo genético adaptado al problema.	
R 7.	Módulo para la carga de pedidos de piezas a recortar, considerando sus dimensiones (ancho y alto), cantidad y las regiones defectuosas en stock.	- Scrum - C# - Microsoft Visual Studio
R 8.	Módulo del algoritmo Cuckoo Search implementado.	- Scrum - C# - Microsoft Visual Studio
R 9.	Módulo del algoritmo genético implementado.	- Scrum - C# - Microsoft Visual Studio
R 10.	Calibración de variables de la experimentación numérica.	- Scrum - C# - Microsoft Visual Studio
R 11.	Interfaz para mostrar de manera gráfica los resultados obtenidos por el algoritmo Cuckoo Search y el algoritmo genético.	- Scrum - C# - Microsoft Visual Studio
R 12.	Informe de experimentación numérica en el que se detalle las pruebas realizadas, sus resultados e interpretación de estos.	- Prueba de Kolmogorov-Smirnov - Prueba F de Fisher - Prueba Z - Microsoft Excel

1.3.2 Herramientas

Microsoft Visual Studio

Microsoft Visual Studio es un IDE (Entorno de Desarrollo Integrado) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, tales como C++,

C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc. (Microsoft, 2017b).

Se ha elegido esta IDE debido a que ofrece funcionalidades que el trabajo de programación, como referencias y autocompletado de códigos, y una depuración avanzada. Una razón adicional de la elección de este IDE es la compatibilidad con el lenguaje de programación que se planea utilizar (C#) y además de que es una herramienta que el autor conoce.

C#

C# (pronunciado “C Sharp”) es un lenguaje de programación orientado a objetivos y que está diseñado para crear una variedad de aplicaciones en la plataforma .NET Framework. (Microsoft, 2017a)

Se ha elegido este lenguaje debido a que es simple y a la vez fuertemente tipado, de modo que ayuda a evitar errores en la programación y facilita la depuración de los programas, principalmente de los más extensos como de los algoritmos. Otro motivo para elegir este lenguaje es que ya se cuenta con el conocimiento del mismo.

1.3.3 Métodos

Comparaciones múltiples de medias

El estudio de la comparación de múltiples medias es un tipo de prueba estadística que se basa en la comparación de las medias de dos o más poblaciones (Porras, 2000).

Para realizar correctamente esta comparación se puede utilizar la prueba Z, la cual requiere condiciones específicas en la distribución y varianza de las muestras, por lo tanto, se deben realizar pruebas adicionales para poder asegurar dichas condiciones. A continuación, se explican cada una de estas pruebas en el orden que deben llevarse a cabo.

1. Prueba Kolmogorov-Smirnov

Es una prueba que ayuda a determinar la bondad de ajuste de dos distribuciones, es decir, si la muestra de una población sigue una distribución especificada (Ledolter & Robert, 2010).

Se justifica el uso de esta prueba debido a que una condición para realizar la prueba Z es tener una distribución normal de las muestras a comparar.

2. Prueba F de Fisher

Es una prueba estadística que permite determinar si una medida estadística sigue una distribución F si la hipótesis nula no puede ser rechazada (Ledolter & Robert, 2010). Una de las aplicaciones más comunes es en el análisis de la varianza en las poblaciones.

Se justifica el uso de esta prueba debido a que una condición para realizar la prueba Z es tener varianzas homogéneas para las muestras a comparar.

3. Prueba Z

Es una de las pruebas estadísticas para comparaciones múltiples más frecuentemente utilizadas. La prueba Z se utiliza para comparar las medias de dos muestras que tiene distribución normal (Porras, 2000).

Se justifica el uso de esta prueba porque permite determinar el algoritmo que obtiene mejores resultados para el problema de optimización del corte a través de la comparación sus medias, indicando cuál algoritmo tiene una media significativamente menor a la del otro. Además, dado que se utilizarán por lo menos 30 muestras en los algoritmos Cuckoo Search y genético, el tamaño de las muestras es el adecuado para la prueba Z.

1.3.4 Metodologías

SCRUM

Es una metodología para la gestión de proyectos de ágiles desarrollada por Ikujiro Nonaka e Hirotaka Takeuchi a principios de los 80 como resultado del análisis de los procesos de desarrollo de grandes empresas (Schwaber & Sutherland, 2013).

En este proyecto se planea usar ciertas prácticas de la metodología, las cuales mencionamos a continuación:

- Product Backlog: Determinar las tareas que se deben realizar y objetivos a cumplir.
- Sprint Backlog: Permite tener un entregable del proyecto en un plazo corto (dos a cuatro semanas).
- Daily Scrum: Esta reunión permite evaluar el progeso para cumplir el objetivo del Sprint y proyectar el trabajo los siguientes días.

- Sprint Retrospective: Permite inspeccionar el estado del proyecto y decidir las acciones de mejora.

Debido a que este proyecto involucra la implementación de algoritmos y el desarrollo de módulos de carga y visualización, se justifica el uso de esta metodología porque permite enfocarse más en la programación que en otros aspectos como documentación o planificación. Además, existe una confianza en la metodología pues viene siendo usada desde principios de los 90 y cuenta con una gran cantidad de documentación disponible.

1.4 Alcance y limitaciones

1.4.1 Alcance

El presente proyecto pertenece al área de ciencias de la computación, específicamente a la rama de algoritmos de optimización y tiene como objetivo implementar un algoritmo metaheurístico Cuckoo Search para resolver el problema de corte de piezas cerámicas recicladas de forma regular. Para alcanzar dicho objetivo, se ha visto conveniente solo considerar como variables la cantidad y dimensiones tanto de las piezas a fabricar como del material del cual se va a recortar, y de estos últimos las regiones que contienen fallas.

A continuación, se procederá a implementar el algoritmo Cuckoo Search y adaptar un algoritmo genético encontrado en la literatura para resolver el problema. Se desarrollarán las estructuras de datos que servirán de apoyo en la implementación de los algoritmos. Luego, se diseñará una función de mérito que será utilizada en ambos algoritmos. Finalmente, se debe de desarrollar la aplicación para finalmente llevar a cabo la experimentación numérica que permitirá comparar los resultados de los dos algoritmos.

Asimismo, la solución considerará como cortes permitidos en los cerámicos únicamente los de tipo guillotinado y que la forma de las piezas es regular. Se han tomado en cuenta estas consideraciones debido a las características de las empresas de la industria de cerámicos, pues la mayoría cuentan con herramientas que solo permiten hacer cortes perpendiculares a los lados y que los productos decorativos como listelos, tacos e insertos, son rectangulares. Una solución que resuelva otras variantes del problema de corte de material va más allá del alcance del proyecto.

Cabe mencionar que el proyecto no abarca el desarrollo de un sistema de información como soporte a los algoritmos implementados, pero sí contará con una interfaz que permitirá visualizar los resultados.

1.4.2 Limitaciones

La principal limitante del proyecto es el tiempo de ejecución de ambos algoritmos, el cual puede verse afectado dependiendo de las características del hardware y software de la computadora en que se ejecute. Este tiempo será calculado como la diferencia entre la hora del sistema al inicio de la ejecución del algoritmo y la hora del sistema al final de la ejecución del algoritmo. Dichas ejecuciones serán realizadas en la misma computadora para simular las mismas condiciones.

Asimismo, para este proyecto, el tamaño de las variables de los pedidos y dimensiones de los cerámicos con los que se ejecutarán los algoritmos serán en base a la producción de una empresa de cerámicos en Lima.

1.5 Viabilidad

En el siguiente apartado se presentan los factores que influyen en la viabilidad del proyecto de fin de carrera.

1.5.1 Viabilidad técnica

Las herramientas que se utilizarán en el proyecto de tesis, entre ellas el lenguaje C#, el IDE Microsoft Visual Studio, son conocidas por el autor, pues a lo largo de la carrera de Ingeniería Informática y en su experiencia laboral ha trabajado con ellas. Todas estas tienen versiones estables y presentan una gran documentación, además que ya se encuentran instaladas en los equipos de la universidad y por lo tanto son de fácil acceso.

Asimismo, se cuenta el apoyo del asesor que domina el tema de algoritmos metaheurísticos, que son la clase de algoritmos a implementar. Además, en la revisión del estado del arte se han encontrado proyectos que utilizan los mismos algoritmos, los cuales permitirán aclarar conceptos y servirán como referencia.

Finalmente, se usarán metodologías que proporcionan un conjunto de buenas prácticas y que servirán de apoyo para cumplir con los objetivos del proyecto.

Por lo expuesto, se puede decir que no habrá un obstáculo para el desarrollo normal del proyecto en el aspecto técnico.

1.5.2 Viabilidad temporal

El tiempo con el que se dispone para llevar a cabo el proyecto es de cinco meses. Para asegurar que durante este periodo se pueda terminar el proyecto, se seguirá el plan de proyecto (ver Figura 2) a partir de la fase de ejecución, con las fechas de entrega de los resultados esperados, el cual empezará inmediatamente después de culminar el curso de Proyecto de Tesis 1.

Por lo tanto, con respecto a la viabilidad temporal, a pesar de que el alcance del proyecto es moderado, se cuenta con el suficiente tiempo para ejecutar cada una de las actividades involucradas en el plan de proyecto.

1.5.3 Viabilidad económica

Con respecto a la viabilidad económica acerca de las herramientas a utilizar, así como el software que deben ser comprados, todas son gratuitas y como se mencionó, ya se encuentran instaladas en los equipos de la universidad, por lo cual no es necesaria una inversión adicional y se puede decir que la viabilidad económica está cubierta.

1.5.4 Viabilidad de datos

Las variables que servirán de entrada a los algoritmos, como la cantidad de pedidos o dimensiones de las piezas, serán en base a la producción de una empresa de cerámicos en Lima, lo cual se puede reemplazar con datos que se encuentran en la literatura o que el mismo autor puede generar debido a que no es muy compleja, por lo que no habría problema en este aspecto.

Nombre de tarea	Duración	Comienzo
PROYECTO DE FIN DE CARRERA	**264 días**	**lun 20/03/17**
Implementación del algoritmo metaheurístico Cuckoo Search para la optimización de cortes en dos dimensiones de productos cerámicos con defectos para la producción de piezas decorativas de forma regular	**264 días**	**lun 20/03/17**
1. Fase de planeación	**102 días**	**lun 20/03/17**
1.1. Problemática	**33 días**	**lun 20/03/17**
Definición del problema	14 días	lun 20/03/17
Investigación del estado del arte	19 días	lun 3/04/17
1.2. Marco conceptual y objetivos	**28 días**	**sáb 22/04/17**
Desarrollo del marco conceptual	17 días	sáb 22/04/17
Definición de objetivo general y específicos	11 días	mar 9/05/17
1.3. Resultados y herramientas	**21 días**	**sáb 20/05/17**
Definición de resultados esperados	7 días	sáb 20/05/17
Invesigación sobre herramientas	14 días	sáb 27/05/17
1.4. Alcance, limitaciones y viabilidad	**20 días**	**sáb 10/06/17**
Definición de alcance y limitaciones	5 días	sáb 10/06/17
Elaboración de la justificación del proyecto	6 días	jue 15/06/17
Análisis de riesgos y de viabilidad	9 días	mié 21/06/17
2. Fase de ejecución		
2.2. Iteración 1	**50 días**	**jue 13/07/17**
Diseñar la función objetivo que mida la pérdida generada por el corte de un céramico.	12 días	jue 13/07/17
Estructuras de datos que soporten la implementación del algoritmo Cuckoo Search.	8 días	mar 25/07/17
Algoritmo Cuckoo Search diseñado.	13 días	mié 2/08/17
Módulo para la carga de datos del algoritmo Cuckoo Search.	13 días	mar 15/08/17
2.2. Iteración 2	**30 días**	**vie 1/09/17**
Adaptar la función objetivo utilizada en el algoritmo genético que mida la pérdida generada por el corte.	9 días	vie 1/09/17
Estructuras de datos que soporten la implementación del algoritmo genético.	6 días	dom 10/09/17
Algoritmo genético adaptado al problema.	7 días	sáb 16/09/17
Módulo para la carga de datos del algoritmo Cuckoo Search y genético.	8 días	sáb 23/09/17
2.3. Iteración 3	**49 días**	**sáb 21/10/17**
Informe de experimentación numérica	3 días	sáb 21/10/17
Calibración de variables de la experimentación numérica.	2 días	mar 24/10/17
Interfaz para mostrar de manera gráfica los resultados de los algoritmos	15 días	jue 26/10/17
Correcciones finales	10 días	vie 10/11/17

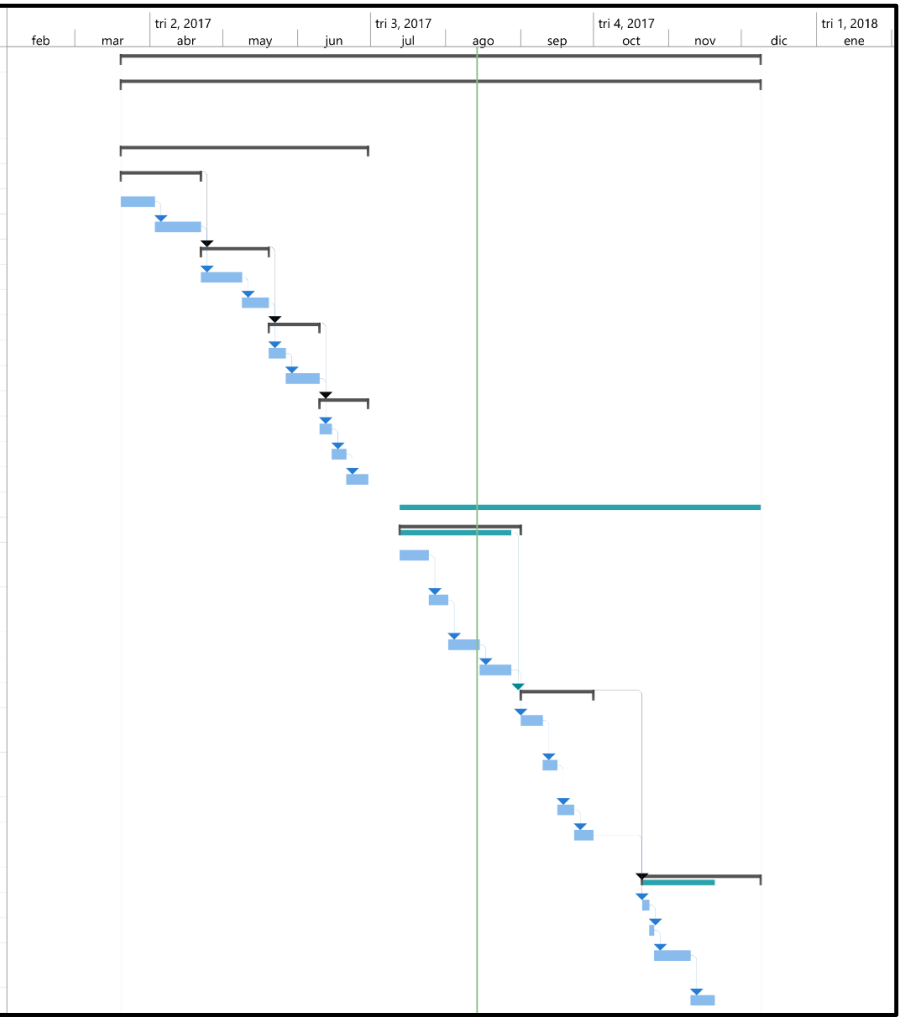


Figura 2. Cronograma del proyecto

1.6 Riesgos

En la siguiente tabla se presentan los riesgos identificados junto con el impacto que tendrán, su nivel y la estrategia de mitigación que se propone.

Tabla 7. Riesgos del proyecto

RIESGO	IMPACTO	ESTRATEGIA DE MITIGACIÓN
Mala planificación del proyecto	- Entregables se entregan fuera de tiempo. - Rechazo de los entregables	Reorganizar todas actividades del plan de proyecto, priorizando la implementación de los algoritmos.
Enfermedades o lesiones graves por parte del tesista	- Entregables se entregan fuera de tiempo. - Desaprobar el curso.	Realizar los entregables con anticipación y comunicar la situación a los profesores del curso.
Pérdida de información y avances del proyecto	- Retraso generado por el tiempo en volver a realizar el mismo trabajo. - Confusión en presentar una versión previa.	Mantener un repositorio donde se encuentre el código fuente del proyecto y tener respaldos en la nube actualizados.
Mala comunicación con el asesor	- El asesor no puede corregir los entregables. - No se aceptan los entregables. - Se pueden cometer errores de concepto graves dado que no se cuenta con la asesoría de un experto.	Comunicar a los profesores del curso la situación y en todo caso la asignación de otro asesor.
Acumulación de observaciones en los entregables	- Retraso en la presentación de los siguientes entregables.	Solicitar las observaciones con anticipación a los profesores del curso y reunirse anticipadamente con el asesor.

1.7 Justificación

En los últimos años, las empresas de la industria de cerámicos han visto un crecimiento que les ha permitido investigar en maneras de minimizar los costos a través de la optimización de la producción. Para la fabricación de sus principales productos decorativos como listelos, tacos e insertos, que provienen del corte de cerámicos reciclados de mayor dimensión y muchas veces con presencia de fallas, se ha encontrado que la mayor pérdida se debe al residuo generado por los cortes.

Permitirles a las empresas de esta industria un método que le indique a sus operarios donde realizar los cortes en los cerámicos de manera que se minimice el desperdicio generado es vital para reducir los costos de producción. Actualmente, las soluciones existentes resuelven el problema sin considerar restricciones propias de la industria, como la presencia de fallas en el material a cortar, además de que la mayoría utiliza variantes de algoritmos genéticos. Sin embargo, no se han encontrado estudios para la solución de este problema con otros algoritmos de reciente aparición, como es el caso del algoritmo Cuckoo Search, el cual viene mostrando un buen desempeño para otros problemas de optimización (Yang & Deb, 2009).

En conclusión, para el presente proyecto de fin de carrera se utilizará el algoritmo metaheurístico Cuckoo Search para resolver el problema de corte de piezas cerámicos recicladas de forma regular. El uso de esta clase de algoritmos no solo se justifica, sino que en muchos casos es deseable para este tipo de problemas de optimización, principalmente por la calidad de la solución obtenida y el tiempo que se demoran, que los hace factible de utilizar en un ambiente de producción.

Capítulo 2. Marco Conceptual

2.1 Introducción

Para comprender el problema que se está abordando en el presente proyecto, en este capítulo se explican los conceptos básicos de complejidad, heurística, metaheurística, así como la definición del problema de corte y los algoritmos que se emplearán.

2.2 Productos cerámicos

Los productos cerámicos que han sido moldeados por el "hombre" utilizando una gran cantidad de minerales y rocas, que luego son permanentemente endurecidos por el calor (Adams, 1961). Otra definición más simplificada es la de objeto moldeado con materias primas naturales plásticas y endurecido permanentemente por el calor (Galán & Aparicio, 2006).

2.2.1 Listelos, tacos e insertos

Son productos cerámicos decorativos, generalmente de forma rectangular, que provienen del corte de cerámicos de mayor dimensión. Dicho cerámicos pueden estar decorados previamente o pueden ser decorados luego del corte.

Se utilizan para cualquier ambiente de un hogar como pisos, baños, cocinas, por lo que uso y producción es ampliamente extendido (ver Figura 3, Figura 4 y Figura 5).

Figura 3. Listelos (Kantu, 2016)

Figura 4. Tacos (Kantu, 2016)

Figura 5. Insertos (Kantu, 2016)

2.2.2 Productos cerámicos reciclados con fallas o roturas

Debido a las propiedades en la composición de los productos cerámicos, se pueden presentar grietas en estos productos, los cuales con el paso del tiempo y condiciones ambientales como temperatura pueden llegar perjudicar el material (Pastor, 1993) Asimismo, estos materiales presentan un grado de rotura frágil, por lo que estas fallas se pueden generar en dichos productos si no se tiene un cuidado en su traslado.

2.3 Corte

Se refiere a la acción de dividir o cortar un material, a través del uso de un instrumento o herramienta (Sánchez, 2016).

2.3.1 Corte de guillotina

Se le llama corte de guillotina a los cortes que se realizan de extremo a extremo en una pieza rectangular, con la dirección paralela a los lados de la pieza y que dan a origen a dos piezas rectangulares. El corte de guillotina generalmente se da en las empresas, ya que utilizan máquinas que solo pueden realizar este tipo de cortes (Sánchez, 2016).

2.4 Problemas de corte de material

Es un problema clásico en el área investigación de operaciones y es definido como la mejorar forma de obtener piezas a través del recorte de un material, de modo que la pérdida generada por el recorte se minimice y la demanda total se satisfaga (Talbi, 2009).

Debido a la aparición de variaciones de este problema, muchas de ellas muy similares, motivaron a Dyckhoff a proponer una tipología para clasificar los problemas de corte y empaquetado (Dyckhoff, 1990). En base a dicha investigación, se propuso una tipología mejorada para cubrir algunas deficiencias y tratar con los recientes desarrollos del problema (Wäscher, Haußner, & Schumann, 2007). Esta última tipología tiene en cuenta las siguientes características:

1. Dimensionalidad: Uno, dos, tres.
2. Tipo de a|signación: Maximización, minimización
3. Surtido de objetos pequeños: Idénticos, débilmente heterogéneos, fuertemente heterogéneos.
4. Surtido de objetos grandes: Idénticos, débilmente heterogéneos, fuertemente heterogéneos.
5. Forma de los pequeños ítems: Rectángulos, círculos, cajas, cilindros.

Adicionalmente, se propusieron otras características al problema que hacen referencia a la orientación y patrones de corte (Lodi, Martello, & Vigo, 1999).

1. Orientación: Los ítems pueden requerir de tener una orientación fija o pueden ser rotados 90°.
2. Cortes de guillotina: Los cortes pueden o no requerir de cortes de extremo a extremo en dirección paralela a ambos lados, es decir, si se realiza en una pieza rectangular, generaría solo dos rectángulos.

Estas dos características generan las siguientes 4 restricciones:

- ✓ RF: Las piezas se pueden rotar 90º (R) y no se requiere que todos los cortes sean de guillotina (F)
- ✓ RG: Las piezas se pueden rotar 90º (R) y se requiere que todos los cortes sean de guillotina (G)
- ✓ OF: La orientación de las piezas es fija (O) y no se requiere que todos los cortes sean de guillotina (F)

OG: La orientación de las piezas es fija (O) y se requiere que todos los cortes sean de guillotina (G)

2.5 Problema de Optimización

En su forma más general, un problema de optimización se representa por:

$$\text{optimizar } f(S), \text{ sujeto } S \in F$$

Donde F es un conjunto de soluciones factibles para el problema y f es una función real llamada función objetivo. En el caso de un problema de minimización, se busca una solución que minimice f(S), mientras que en el caso de un problema de maximización se busca una solución que maximice f(S), dentro de todo el dominio F

de soluciones factibles (Resende & Ribeiro, 2016). Básicamente, se pueden clasificar en dos tipos:

2.5.1 Optimización continua

Cuando sus variables pueden tomar valores continuos, es decir, en principio cualquier valor real.

2.5.2 Optimización discreta

Cuando sus variables pueden tomar valores discretos, es decir, aquellos que solo puede tomar un conjunto finito o infinito contable de valores. Estos son también conocidos como **problemas de optimización combinatoria** (Resende & Ribeiro, 2016).

2.6 Complejidad computacional

La complejidad computacional de un problema está referida a la cantidad de recursos computacionales que intervienen para calcular su solución. Se pueden clasificar en dos tipos:

2.6.1 Complejidad P

La complejidad de clase P (polinómico) es el conjunto de todos los problemas de decisión que pueden ser resueltos por un algoritmo en tiempo polinomial. Usualmente estos problemas son fáciles y conocidos (Du & Swamy, 2016).

2.6.2 Complejidad NP

La complejidad de clase NP (polinómico no determinístico) es el conjunto de todos los problemas de decisión que no pueden ser resueltos por un algoritmo en tiempo polinomial. Un subconjunto de ellos son los problemas NP-díficil que son muy complejos que involucran una gran cantidad de variables, por lo que se hacen intratables y no existen algoritmos que lo resuelvan en un tiempo polinomial (Du & Swamy, 2016).

2.7 Algoritmos heurísticos

Son procedimientos para resolver un problema de optimización combinatoria de forma aproximada e intuitiva. Utiliza la estructura y características propias del problema de forma inteligente para obtener una solución en corto tiempo.

En la siguiente clasificación se ubican a los más conocidos: (Martı, 2003)

2.7.1 Métodos Inductivos

A partir de las propiedades o técnicas aplicadas en casos pequeños o sencillos del problema (que generalmente son más fáciles de analizar) se trata de generalizar al caso completo (Martı, 2003).

2.7.2 Métodos de Reducción

Restringen el espacio de soluciones introduciendo restricciones en base a propiedades que cumplen por la mayoría de soluciones, con el objetivo de simplificar el problema. Existe el riesgo de dejar fuera a las soluciones óptimas (Martı, 2003).

2.7.3 Métodos Constructivos

Construyen la solución paso a paso, de forma determinista escogiendo generalmente al mejor elemento en cada iteración (Martı, 2003).

2.7.4 Métodos de Búsqueda Local

Buscar mejorar la solución del problema de forma progresiva a partir de una solución inicial, obteniendo a través de un *movimiento* una mejor solución (Martı, 2003).

2.8 Algoritmos metaheurísticos

Es un término acuñado por Fred Glover en el año 1986 para referirse al conjunto de técnicas “superiores” que las heurísticas. Son procedimientos para explorar y explotar de forma iterativa y estrategias inteligentes el espacio de búsqueda. Poseen un balance entre las siguientes características: (Du & Swamy, 2016)

- Explotación (intensificación): Orientados a buscar en el espacio actual.
- Exploración (diversificación): Orientados a buscar en espacios distantes o región global.

Es difícil proponer una clasificación de los algoritmos ya que muchos comparten características de otros. Una diferenciación es la siguiente:

2.8.1 Constructivas

La solución se va construyendo iterativamente añadiendo componentes a una solución inicialmente vacía (Du & Swamy, 2016). Ejemplo: Algoritmo GRASP.

2.8.2 Basada en una solución (o basadas en trayectorias)

La solución es reemplazada iterativamente por una solución de mejor calidad vecina a la solución inicial. Se puede decir que presentan una mayor característica de explotación (Du & Swamy, 2016). Ejemplo: Algoritmo Tabú.

2.8.3 Basada en una población

El número de soluciones es actualizado iterativamente (evolucionan) a partir de una población inicial hasta satisfacer una condición, de la cual se obtendrá al mejor individuo de la última población (solución). Se puede decir que presentan una mayor característica de exploración (Du & Swamy, 2016). Ejemplo: Algoritmo Genético.

2.8.4 Basada en inteligencia de enjambre

Este tipo de algoritmos se componen de individuos que interactúan localmente entre ellos y con su entorno. Cada uno de estos agentes es una solución candidata al problema, los cuales cambian de estado siguiendo ciertas reglas y por tanto cubren un mayor espacio de soluciones. (Du & Swamy, 2016).

Debido a que los comportamientos de algunos individuos son inspirados en la naturaleza, especialmente en los sistemas biológicos, muchos de ellos se consideran como bioinspirados. Ejemplos: Algoritmo de Colonia de Hormigas, Algoritmo de Murciélago.

2.9 Algoritmo Cuckoo Search

Es un algoritmo metaheurístico de búsqueda para problemas de optimización. Se puede clasificar como un algoritmo de “inteligencia de enjambre”, pues está inspirado en el comportamiento colectivo de las aves cuco, en el cual algunas especies suelen dejar sus huevos en los nidos de otras aves. Si una de estas aves descubre un huevo que no es suyo, puede optar por tirar dicho huevo o abandonar el nido y construir uno en otro lugar. Sin embargo, muchos de estos huevos son muy similares a los otros, y al estar en varios nidos reduce la posibilidad de ser abandonados (Yang & Deb, 2009).

Aquellos nidos que son “descubiertos” por las otras aves que los habitan son abandonados y removidos de la población, y son remplazadas por nuevos nidos (con nuevas soluciones aleatorias). Se consideran las siguientes reglas idealizadas: (Yang & Deb, 2009)

1. Cada ave cuco puede poner un solo huevo a la vez y lo deja en un nido (solución) escogido de forma aleatoria.
2. Los nidos que tengan los huevos de mejor calidad (soluciones) son los que serán transferidos den paso a las nuevas generaciones de aves cuco.
3. El pájaro de un nido puede descubrir un huevo de cuco (que no le pertenece) con una probabilidad P ∈ [0, 1]. El ave puede optar por arrojar el huevo del ave cuco fuera del nido o en todo caso abandonar el nido (en forma simple, es equivalente a reemplazar una proporción P de los peores nidos).

A continuación, se presenta el seudocódigo de un algoritmo Cuckoo Search:

Tabla 8. Seudocódigo del algoritmo Cuckoo Search

```
Procedimiento CuckooSearch (Instancia del problema)
1. Función objetivo f(x), x = (x1, …. xd)^T
2. Generar población inicial de NP nidos xi (i = 1, 2, …, n)
3. Mientras (t < Máximo de generaciones) o (No se cumpla condición de parada)
   3.1. Obtener un huevo de cuco "i" aleatoriamente con vuelo de Levy
   3.2. Calcular su fitness Fi
   3.3. Seleccionar un nido "j" de los NP nidos aleatoriamente
   3.4. Calcular su fitness Fj
   3.5. Si (Fi > Fj) entonces
        3.5.1. Reemplazar "j" con la nueva solución "i"
   3.6. Fin Si
   3.7. Abandonar una fracción (P) de los peores nidos
   3.8. Construir nuevas soluciones con vuelo de Levy
   3.9. Mantener las mejores soluciones (nidos) para la siguiente generación.
  3.10. Ordenar de mejor a peor solución (nido) y almacenar la mejor actual
4. Fin Mientras
5. Retornar mejor nido (solución) encontrado
Fin CuckooSearch
```

En la línea 3.1 del seudocódigo se obtiene un cuco (nueva solución) a través de un vuelo de Levy, el cual es un movimiento aleatorio de un estado en base a una función de probabilidad. Para simular este movimiento, existen varios algoritmos que usan funciones de probabilidad con diferentes parámetros (Kamaruzaman, Zain, Yusuf, & Udin, 2013). La implementación usada en este proyecto se especifica en la sección 6.3.3 del presente documento.

Este algoritmo ha sido aplicado en varios problemas de optimización grandes, dado que es fácilmente de paralelizar se utilizó correctamente para problemas de instancias grandes (Du & Swamy, 2016). Para el presente proyecto, se aplicará dicho algoritmo teniendo como referencia sus buenos resultados en problemas tanto de rutas como de empaquetamiento (Yang & Deb, 2009).

2.10 Algoritmo genético

Un algoritmo genético es un procedimiento metaheurístico de búsqueda basados en la idea de la evolución y la selección natural. Estos algoritmos evolucionan las poblaciones de soluciones que se combinan por parejas para generar descendencia. Los elementos de la población de la solución también se someten a procesos de mutación que crean individuos con nuevas características. Los individuos más aptos en cada generación son los que más probablemente sobreviven y pasan sus características a los individuos de la próxima generación (Martı, 2003).

Usualmente, se utilizan varios operadores y estrategias para generar la población inicial. Asimismo, cuenta con los siguientes *operadores genéticos*:

- Selección: Se seleccionan a los mejores individuos (con mejor fitness) que serán elegidos en base a una probabilidad.
- Cruce o apareamiento: De un grupo de parejas seleccionadas aplicar una operación de cruce para generar nuevos hijos.
- Mutación: De un grupo de hijos (soluciones) ir mutándolo (cambiándolo) en base a una probabilidad. Este nuevo grupo de hijos será parte de la nueva generación.

De forma general, el proceso de evolución termina después de varias generaciones sin mejora del mejor individuo. Si bien las implementaciones originales de algoritmos genéticos se basan casi exclusivamente en opciones probabilísticas de cruce, reproducción y mutación, en la actualidad existen variaciones en las que incorporan desarrollos del campo de la optimización y heurística, como el uso de algoritmos aleatorios voraces para generar la población inicial y la búsqueda local para mejorar las características de la descendencia (Martı, 2003).

En la Tabla 9 se presenta el seudocódigo de un algoritmo genético:

Tabla 9. Seudocódigo del algoritmo genético

```
Procedimiento AlgoritmoGenetico (Instancia del problema)
1. Definir la función de fitness
2. Generar población inicial
3. Mientras (t < Máximo de generaciones) o (No se cumpla
   condición de parada)
   3.1. Seleccionar padres de la población basados en su
        fitness
   3.2. Producir hijos a partir de los padres
        seleccionados
   3.3. Mutar los individuos hijos
   3.4. Extender la población añadiendo los hijos
   3.5. Reducir la población extendida
4. Fin Mientras
5. Retornar mejor solución (cromosoma) encontrada
Fin AlgoritmoGenetico
```

Capítulo 3. Estado del Arte

3.1 Introducción

Existe una gran cantidad de formas para resolver el problema de corte de material en su forma clásica (una dimensión). Es este capítulo se busca ver las diferentes formas con las que ha sido abordado el problema, investigando una tipología para definir el problema y sus variantes, así como los métodos que se aproximan a la mejor solución en su forma bidimensional, principalmente. Encontramos formulaciones de programación lineal y métodos aproximados con algoritmos heurísticos y metaheurísticos.

Para la revisión del estado del arte de este proyecto se ha utilizado el método tradicional.

3.2 Revisión y discusión

Investigaciones acerca del problema de corte de material

3.2.1 Aproximación de Gilmore y Gomory por programación lineal

Esta fue una de las primeras aproximaciones al problema de corte de material. En la investigación se plantea expresar el problema como uno de programación línea entera, considerando como variables del problema un conjunto válido de patrones de corte, la cantidad de piezas por patrón y el número de veces que un patrón es utilizado, tratando de minimizar esta cantidad como función objetivo, para el caso de una dimensión.

Una vez propuesto el modelo de programación lineal, establecen que para su solución no es necesario probar todas las combinaciones pues para un problema de muchas variables se hace intratable, sino que plantean que cada una de las columnas que mejoran parte de la matriz solución al problema puede ser encontrada resolviendo un problema auxiliar de la mochila asociado, para lo cual también proponen un algoritmo a dicho problema (Gilmore & Gomory, 1963). Esta relación tiene sentido si tomamos en cuenta que ambos problemas, el de corte de material y la mochila, son de empaquetamiento. Asimismo, en la misma investigación plantean el problema dual de maximización. El modelo original se encuentra en (Gilmore & Gomory, 1961).

3.2.2 Método de Wang

Este algoritmo fue desarrollado P.Y. Wang para el problema de corte en dos dimensiones considerando cortes de guillotina, piezas rectangulares y un solo material rectangular. La idea básica del problema es, en vez de enumerar todos los posibles cortes que se pueden realizar al material rectangular, se va construyendo los patrones de corte añadiendo sucesivamente al lado de otro otros rectángulos de forma horizontal y vertical, de modo tal que se escoja la construcción que genere menor pérdida interna y externa.

Se tiene que tomar en cuenta que la solución que se viene construyendo evite soluciones indeseables como que sobrepase el tamaño del material que se está recortando o que la pérdida de dicha construcción sea menor a un porcentaje del área de todo el material de modo que ya no es aceptable (este porcentaje es input al algoritmo). Este parámetro junto con las restricciones de factibilidad trata de reducir el número de combinaciones del problema. Como ilustra el autor en su publicación (Wang, 1983), en la Figura 6 vemos el tipo de construcción horizontal y vertical, respectivamente (el área sombreada es la pérdida generada).

El detalle del algoritmo original se encuentra se encuentra en (Wang, 1983). Además, en (Vasko, 1989) consideran restricciones adicionales para combinación de los rectángulos y logra ser hasta 25 veces más rápido en generar soluciones óptimas, pero aun así el método sigue siendo computacionalmente lento cuando las piezas son pequeñas.

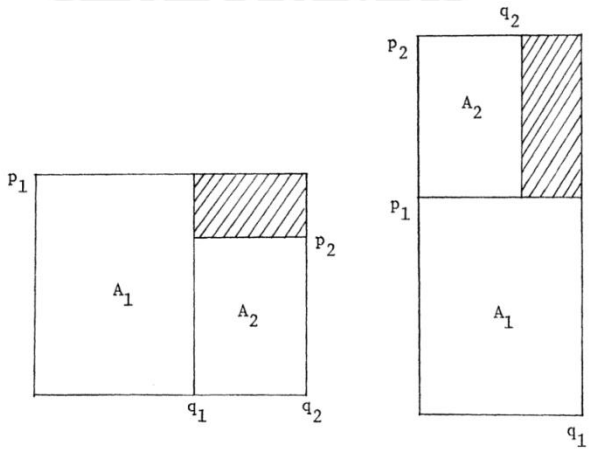


Figura 6. Construcción horizontal y vertical (Wang, 1983)

3.2.3 Un algoritmo GRASP para el problema de corte de material en dos dimensiones con cortes no-guillotinados

Un algoritmo GRASP se utiliza para resolver problemas de optimización combinatoria complejos. Es un proceso iterativo el cual tiene dos fases: construcción y mejora. En la fase de construcción, se va construyendo una solución adaptando un método voraz. Los elementos que serán parte de la solución no son elegidos por el criterio voraz sino de forma aleatoria, en base a una lista de posibles candidatos para cierto parámetro de relajación. Luego, en la fase de mejora se realiza una búsqueda local a partir de la solución de la fase anterior, con lo cual se espera una mejor solución (Resende & Ribeiro, 2016).

En este proyecto, se desarrolla una solución basada en el algoritmo GRASP. Para la fase de construcción, mantiene una lista de rectángulos inicialmente está sin cortes. Se escogen los rectángulos más pequeños, y se colocan las piezas (ordenadas por prioridad y dimensión) empezando de la menor distancia a la esquina de dicho rectángulo. Se trata de minimizar la demanda de piezas pequeñas con los rectángulos más pequeños, pues si se corta rectángulos grandes para piezas pequeñas, es probable que estos rectángulos nuevos queden inservibles para piezas más grandes que aún quedan por cortar, generando mayor pérdida de material. El resultado de estos cortes (rectángulos libres) se irán agregando a dicha lista original, como si fuera stock nuevo. Las piezas que están siendo asignadas se van retirando de la lista de piezas y termina cuando ya no quede ninguna.

Para la fase de mejora corrige algunas decisiones de la fase de construcción y proponen tres alternativas de mejora en un tiempo muy corto. La primera consiste en tomar una pieza adyacente a un bloque vacío y luego considerar reducirla o eliminarla. El espacio libre que deja se combina con los otros, formando un nuevo espacio libre y para este se aplica el algoritmo de la fase construcción (movimientos hacia la esquina). Para mayor detalle de las otras dos propuestas de mejora y la versión completa del algoritmo ver (Alvarez-Valdes, F. Parreño, & Tamarit, 2005).

3.2.4 Aplicación de un algoritmo genético para la optimización del corte de material

Los algoritmos genéticos son procedimientos para resolver problemas de optimización basados en la evolución de los seres vivos en la naturaleza y la selección natural (el que sobrevive es el más fuerte). Utilizan conceptos como cromosomas que hacen

analogía a una posible solución, las cuales son seleccionadas de una población. A estas se les aplican operaciones que simulan la evolución con el fin de obtener nuevos individuos (soluciones) cada vez mejores.

Para representar a los cromosomas, el autor se basa en una notación postfija con operadores H y V, el cual recibe dos rectángulos y produce el mínimo rectángulo capaz de cubrir los dos tomados de forma horizontal y vertical, respectivamente. Esta representación permite una construcción de la solución (cromosoma) en forma de árbol. Por ejemplo, para el cromosoma “1 5 H y 7 6 V 2 H 4 H V 9 8 V 3 V H” se puede ver reflejado en forma de árbol y representado por piezas en la Figura 7.

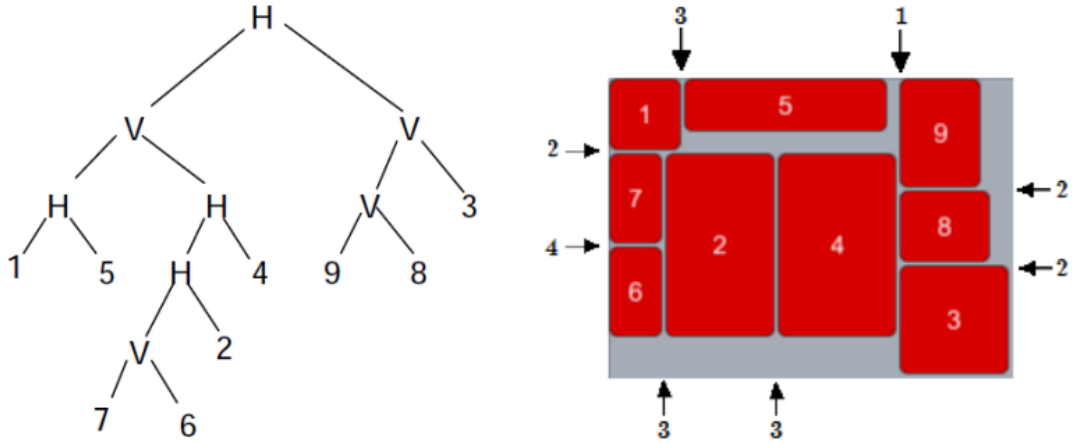


Figura 7. Representación del cromosoma y su estructura arbórea (Sánchez, 2016)

Al ser una representación no ambigua, permite determinar los cortes tipo guillotina a realizar para obtener una pieza siguiendo el camino de la raíz hasta la hoja. Por ejemplo, para obtener la pieza 3 se realizarían dos cortes, primero en 1 y luego en 2.

En este proyecto consideran la función de fitness o aptitud tomando en cuenta la minimización del material que se desecha y en la selección se utiliza el método de la ruleta, donde la probabilidad de selección de cada individuo es en base al valor de la función fitness con respecto a la de la población. Debido a que se debe considerar un orden en el cruce de los individuos para evitar soluciones no válidas, utiliza un tipo de cruce PMX (Partially-mapped crossover). Finalmente, para la mutación se basa en el intercambio de elementos y en el reemplazo mantiene a los mejores individuos (conocidos como élite) y los demás son reemplazados por su descendencia. Un mayor detalle de los métodos expuestos se encuentra en (Sánchez, 2016).

Software y productos relacionados

Están disponibles una gran variedad de paquetes que resuelven el problema de forma similar. En esta sección presentamos un programa y las características que ofrece, que son compartidas entre la mayoría de productos.

3.2.5 Real Cut 2D (Optimal Programs)

Este programa permite introducir las dimensiones las piezas rectangulares a obtener, definidas por ancho y alto, y la cantidad requerida. También permite introducir las dimensiones del material a partir del cual cortar, siendo solamente rectangulares definidas por ancho y alto y la cantidad disponible. Permite ajustar el tipo de corte (tipo guillotina o no-guillotina) y además si está permitida la rotación de las piezas (solo de 90°). Muestra el resultado de los cortes en los materiales tanto en forma gráfica los cortes a realizar con la posibilidad de exportarlo en formato de texto. Además, permite ajustar el grado de optimización, ya que si uno no está satisfecho con la solución mostrada puede incrementar dicho parámetro y volver a ejecutarlo (Real Cut 2D, 2012).

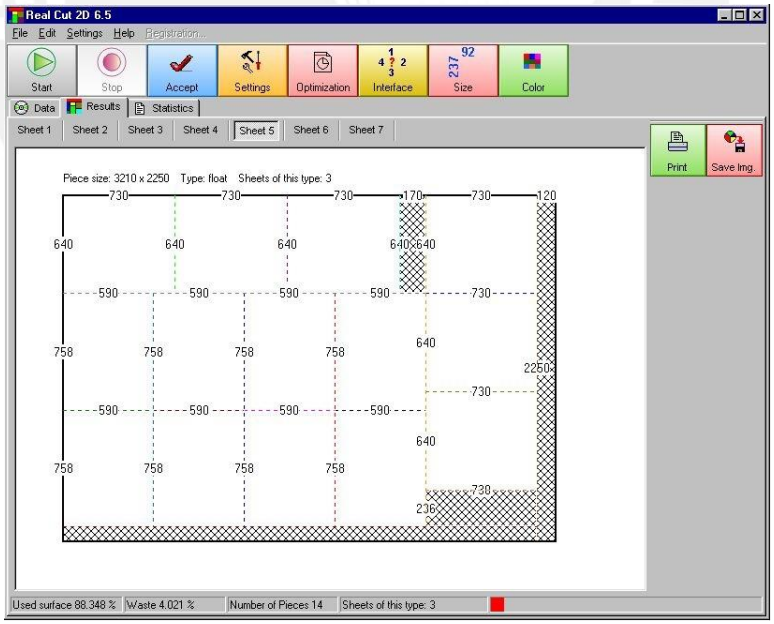


Figura 8. Ventana de resultados del programa Real Cut 2D (Real Cut 2D, 2012)

3.3 Conclusiones

Las soluciones encontradas en el estado del arte se enfocan en resolver el problema en su forma clásica y con restricciones como el corte de guillotina y material tanto de stock como piezas de forma rectangular. La mayoría de investigaciones tiene presente

la opción del corte de guillotina y de forma rectangular para resolver el problema, sin embargo, carecen de la restricción de defectos en los materiales, que se presenta en varias industrias como la de cerámicos. Un cuadro comparativo de dichas investigaciones se presenta en la Tabla 10.

Asimismo, de los productos de software que se encontraron, la mayoría son muy similares, pues resuelven el problema utilizando métodos heurísticos y ofrecen un conjunto de configuración muy limitado, pues no toman en cuenta otras características más que el corte de guillotina y en algunos casos la rotación de las piezas.

En conclusión, en la mayoría de los casos se ha abordado el problema de la optimización del corte sin tomar restricciones adicionales que se presentan en la realidad, por lo que el desarrollo un algoritmo metaheurístico que considere la característica de material defectuoso sería una contribución al estado del arte del problema de corte de material.

Tabla 10. Cuadro de comparación de estudios relacionados

Estudios	Técnica	Dimensiones	Corte guillotinado	Considera regiones defectuosas
Aproximación de Gilmore y Gomory	Programación Lineal	1	Sí	No
Método de Wang	Heurística	2	Sí	No
Un algoritmo GRASP para el problema de corte de material en dos dimensiones con cortes no-guillotinados	Metaheurística (Algoritmo GRASP)	2	No	No
Aplicación de un algoritmo genético para la optimización del corte de material	Metaheurístico (Algoritmo Genético)	2	Sí	No
Propuesta para el problema	Metaheurístico (Cuckoo Search)	2	Sí	Sí

Capítulo 4. Definición del problema

4.1 Introducción

Este este capítulo abordaremos el **resultado esperado 2 (R 2)**, el cual es definir una función objetivo que mida la pérdida generada por el corte de un cerámico. Para ello, primero se definirá formalmente el problema de este proyecto como un problema de optimización, indicando la función objetivo y las restricciones a las que está sujeta.

Finalmente, se explicará cómo se lleva a cabo la estrategia de solución del problema, que consiste primero plantear una solución considerando stock grande (como la unión de todos los stocks) para luego proceder a su división en stock múltiple y evitando los defectos (representado por regiones rectangulares) encontrados en ellos.

4.2 Definición del problema de optimización

4.2.1 Función objetivo: Minimizar el residuo generado debido al recorte de material

Para llegar a minimizar el residuo generado debido al recorte de material, como objetivo principal, se busca minimizar el porcentaje del material que es considerado como defecto. Para este proyecto, se considera como residuo la región del stock que no fue utilizado, así como las piezas que estén sobre una región con defecto. Algo interesante es que la función objetivo a proponer favorece a que las piezas no se encuentren sobre regiones defectuosas y, si es que están, serán las piezas más pequeñas, pues de lo contrario generarían mayor residuo.

Se puede formalizar lo expresado en la siguiente función objetivo:

$$\text{F. O.} = \text{Mín}\left(1 - \frac{\sum_{\text{i}}^{\text{N}} \text{áreaPiezas}_{\text{stock}_\text{i}}}{\sum_{\text{i}}^{\text{N}} \text{área}_{\text{stock}_\text{i}}}\right) \times 100$$

Donde:

- N: El número de stocks utilizados para atender el pedido. El valor mínimo que puede tomar es 1 (todo el pedido se atiende con un stock), y el valor máximo es igual a la cantidad de piezas solicitadas (una pieza por stock).
- $stock_i$: Es el i-ésimo stock, con un ancho, alto y que tiene asociado una lista de defectos representados por piezas definidos.
- $areaPiezas_{stock_i}$: Es la sumatoria del área de todas las piezas en cada $stock_i$, que sería igual a todas las piezas del pedido. Se calcula internamente multiplicado el ancho y alto de cada pieza.

- $area_{stock_i}$: Es la sumatoria del área ocupada de cada $stock_i$. Se calcula internamente multiplicando el ancho y alto de cada stock.

4.2.2 Restricciones

1. **La cantidad de stocks disponibles debe ser mayor o igual a la cantidad de piezas solicitadas**

   Restricción que indica que la cantidad de stocks disponibles debe ser lo suficientemente grande para poder abarcar cualquier solución al problema, en el mejor y peor de los casos. Asimismo, la cantidad de piezas, para que se considere un pedido, deberá ser mínimo 1.

$$1 \leq \text{CantidadPiezas} \leq \text{CantidadStocksDisponibles}$$

2. **El rectángulo que cubre las piezas debe cubrir a la pieza a recortar**

   Restricción que indica que existe un rectángulo deberá ser capaz de cubrir a cada una de las piezas (no deben estar salidas de dicho rectángulo). Se prefiere que dicho rectángulo sea mínimo, pues al inicio se considera el stock como la unión de todos los stocks posibles (considerado infinito), tanto en ancho y largo, antes de dividirlos en stock múltiple.

$$\text{ancho}_{\text{RMP}(p_1,p_2,\dots p_n)} \leq \text{pieza}_i$$

$$\text{largo}_{\text{RMP}(p_1,p_2,\dots p_n)} \leq \text{pieza}_i$$

   Donde:

   - $pieza_i$: Es el cerámico a recortar solicitado, definido por ancho, alto y dos coordenadas de su esquina superior izquierda.
   - $\text{RMP}(p_1, p_2, \dots p_n)$: Es el rectángulo capaz de cubrir a todas las piezas solicitadas, antes de hacer la separación en stocks.
   - $pieza_i$: Es el cerámico a recortar solicitado, definido por ancho, alto y dos coordenadas de su esquina superior izquierda.
   - $ancho$: Función que devuelve el ancho de un rectángulo (pieza u stock).
   - $alto$: Función que devuelve el alto de un rectángulo (pieza u stock).

3. **Las piezas no se pueden solapar**

   Restricción que se busca es que no exista ningún solapamiento de piezas en la disposición.

$$\nexists\, p_i, p_j \mid x_i \leq x_j + \text{ancho}_i \cap y_i \geq y_j + \text{largo}_i$$

4. **Las piezas no pueden cubrir regiones con defectos**
   Lo que se busca es que no exista ningún solapamiento de piezas solicitadas con las regiones con defectos, representadas como piezas.

$$\nexists\, p_i, p_j \mid p_j \in RD \;\; x_i \leq x_j + \text{ancho}_i \;\cap\; y_i \geq y_j + \text{largo}_i$$

   Donde:

   - RD: Es el conjunto de regiones con defectos, representada por piezas $p_1, p_2, \dots p_n$.

4.3 Estructuras de datos

En esta sección se describirán las estructuras de datos utilizadas para representar la solución del problema, las cuales serán utilizados por ambos algoritmos: Genético y Cuckoo Search. Las estructuras de datos propias de cada algoritmo serán presentadas en las secciones de diseño de los mismos.

4.3.1 Lista de cadenas

Esta estructura permitirá representar una solución considerando un stock grande, la cual consistirá en una cadena de caracteres numéricos y letras para los cuales se ha utilizado, por mayor facilidad, una lista de cadenas para que puedan soportar tanto números, letras y cadenas vacías.

Tabla 11. Definición de la solución

```
List<String> solución;
```

A continuación, se presenta un ejemplo de la solución al problema, el cual tiene tamaño N (ocupado por piezas y operadores).

Tabla 12. Ejemplo de la solución

Posición	0	1	2	3	4	5	6	7	8	9	…	N - 1
Valor	1R	5N	H	7N	6N	V	2N	H	4R	H	…	H

Donde el valor de la lista, dependiendo de su tipo, representa lo siguiente:

- Valor tipo numérico: Representan al identificador de una pieza y su orientación, es decir, el valor “1R” se refiere a la pieza 1 con rotación de 90° mientras que, el valor “5N” se refiere a la pieza 5 sin rotación de 90°.

- Valor tipo carácter: Representan a un operador, representado por los caracteres H o V.

4.3.2 Clase rectángulo

Esta clase permitirá definir una pieza o conjunto de piezas (bloque) a partir de dos coordenadas x, y, que se interpretan respecto a un origen (0, 0) y el ancho y alto, representado por las variables w y h. Asimismo, existe una variable rotación la cual indicará si la pieza está rotada 90° respecto a cómo se registró inicialmente.

Tabla 13. Definición de la clase Rectángulo

```
class Rectangulo {
    int id;
    float x, y;
    float w, h;
}
```

4.3.3 Lista de rectángulos

Esta estructura permitirá representar la demanda de las piezas, los cuales estarán representados por una lista de las piezas (usando la clase Rectángulo, ver sección 4.3.2).

Tabla 14. Definición de la lista de piezas

```
List<Rectangulo> listaPiezas;
List<Rectangulo> listaStocks;
```

A continuación, se presenta un ejemplo de la lista de N piezas demandadas.

Tabla 15. Ejemplo de una lista de piezas

Posición	0	1	2	3	4	…	N-1
id	1	5	7	6	2	…	N
x	0	0	0	0	0	…	0
y	0	0	0	0	0	…	0
w	40	40	10	40	10	…	40
h	8	25	10	8	10	…	25

Donde las variables representan lo siguiente:

- Id: El identificador de las piezas: $[\text{pieza}_1,\ \text{pieza}_2, \ldots\ \text{pieza}_N]$.

- x, y: Coordenadas de las piezas respecto al origen (0, 0). Inicialmente, todas estará con posición (0, 0).
- w, h: El ancho y alto de las piezas.

4.3.4 Clase stock

Esta clase permitirá representar un material stock, el cual tiene un ancho y un alto, con la particularidad que, a diferencia de una pieza, el stock tiene asociado una lista de defectos, los cuales serán representados por una lista de piezas.

Tabla 16. Definición de la clase stock

```
class Stock {
    int w, h;
    List<Rectangulo> listaDefectos;
}
```

Adicionalmente, antes del cálculo del fitness, se ejecuta un algoritmo en el que se divide la solución en stock múltiple (ver sección 4.4.3.1), por lo que cada Stock tendrá asociado una lista de piezas, inicialmente vacía.

4.3.5 Lista de stocks

Esta estructura permitirá representa el material de stock que se tiene disponible, representado como una lista de stocks (ver sección 4.3.4).

A continuación, se presenta un ejemplo de una lista de N stocks disponibles, cada uno con un máximo de M defectos.

Tabla 17. Ejemplo de una lista de stocks

Posición	0	1	2	3	4	…	N-1
Id	1	2	3	4	5	…	N
w	100	300	200	100	200	…	300
h	200	300	300	200	300	…	300
$Defecto_1$	$Pieza_1$	$Pieza_1$	$Pieza_1$	$Pieza_1$	$Pieza_1$	…	$Pieza_1$
$Defecto_2$	$Pieza_2$	$Pieza_2$	$Pieza_2$		$Pieza_2$	…	$Pieza_2$
…	…	…	…	…		…	…
$Defecto_M$		$Pieza_M$			$Pieza_M$		

Donde las variables representan lo siguiente:

- Id: El identificador del stock: $[\text{stock}_1,\ \text{stock}_2, \ldots\ \text{stock}_N]$.
- w, h: El ancho y alto del stock.
- $\text{Defecto}_{1\text{-}M}$: Representa una región con defecto, el cual será representado por la misma estructura que una pieza (ver la Clase rectángulo en la sección 4.3.2), indicando la posición relativa del stock donde pertenecen.

4.3.6 Clase nodo

Esta clase permitirá definir un árbol binario en el cual pueda representar la notación postfija utilizada en la representación del algoritmo. Contiene dos atributos de la misma clase que representan los hijos izquierdo y derecho. Además, posee una variable rectángulo (ver sección 4.3.2) que indicará el área ocupada y la posición de una pieza (si es una hoja) o un bloque (si es la unión de varias piezas). Esto último se definirá a través de la variable tipo de integración, que puede ser "H" o "V".

Tabla 18. Definición de la clase Nodo

```
class Nodo
{
    Nodo izquierdo;
    Nodo derecho;
    Rectangulo rect;
    String tipoIntegracion;
}
```

4.4 Representación de la solución

4.4.1 Operadores de integración utilizados

La representación de la solución en un problema influye en la complejidad e implementación de la solución. Para este trabajo, utilizaremos una representación postfija a través de una lista de caracteres, planteada por los autores (León, Miranda, Rodríguez y Segura, 2007). Para definir dicha estructura, utilizaremos dos operadores:

- **Operador H:** Recibe dos piezas como argumento y produce el mínimo rectángulo capaz de cubrir dichas piezas, dispuestos horizontalmente, uno al lado del otro. Las dimensiones de dicho rectángulo mínimo posible (L_h y W_h) se obtienen de la siguiente forma:

$$L_h = \text{largo}_{p_1} + \text{largo}_{p_2}$$
$$W_h = \max(\text{ancho}_{p_1}, \text{ancho}_{p_2})$$

- **Operador V:** Recibe dos piezas como argumento y produce el mínimo rectángulo capaz de cubrir dichas piezas, dispuestos verticalmente, uno al lado del otro. Las dimensiones de dicho rectángulo mínimo posible (L_h y W_h) se obtienen de la siguiente forma:

$$L_h = \max(\text{largo}_{p_1}, \text{largo}_{p_2})$$
$$W_h = \text{ancho}_{p_1} + \text{ancho}_{p_2}$$

Donde:

Li: Longitud de la pieza i.

Para ejemplificar los operadores, podemos verlos en las siguientes figuras (Figura 9 y Figura 10):

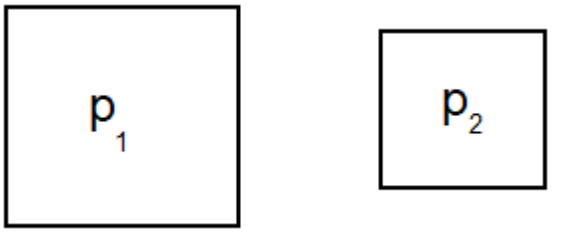


Figura 9. Piezas

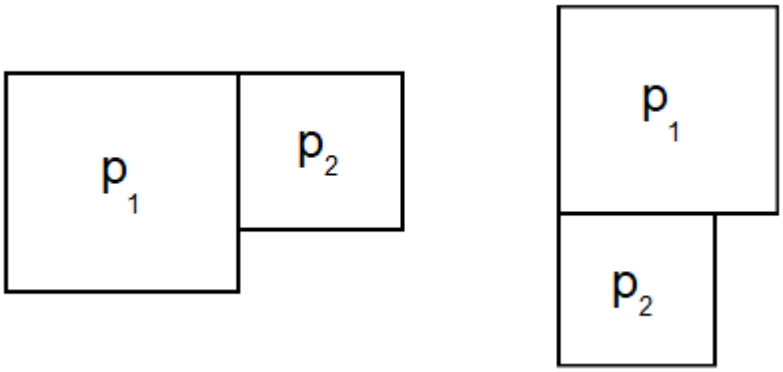


Figura 10. Piezas luego de aplicarse la operación p1p2H y p1p2V

4.4.2 Interpretación de una solución

Esta estructura nos servirá tanto para la definición de la solución, conocida como "huevo de un nido" en el algoritmo Cuckoo Search y como "cromosoma de un individuo" en el algoritmo Genético.

Utilizando los operadores de la sección anterior, podemos decidir la solución de un caso real. Por ejemplo, sea la solución “1 5 H 7 6 V 2 H 4 H V 9 8 V 3 V H”. Entonces, se comienza tomando los elementos de izquierda a derecha, y colocándolos en una pila hasta la aparición de un operador V o H, que se encargará de integrar dichos elementos (es decir, piezas).

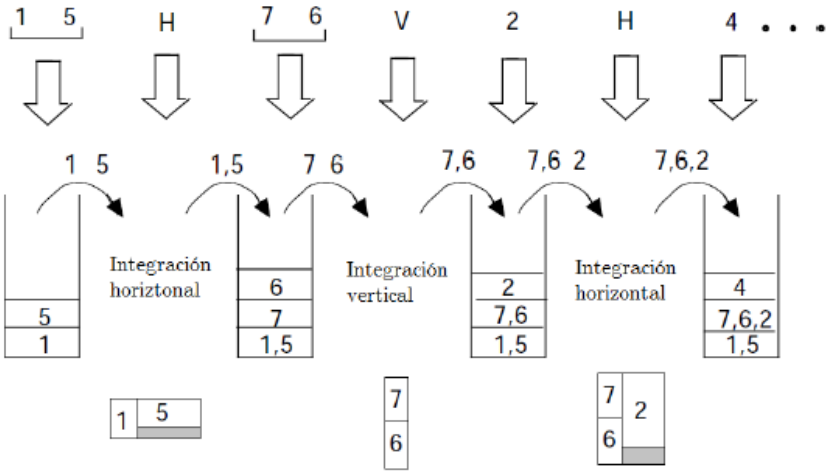


Figura 11. Decodificación de una solución

Si seguimos los pasos hasta el final, obtendremos la siguiente disposición (ver Figura 12) donde las flechas indican el orden en que los cortes de guillotina deben ser realizados.

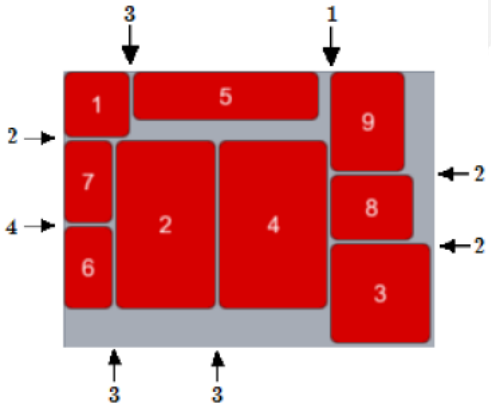


Figura 12. Disposición gráfica de las piezas

Si se desea obtener los cortes necesarios para obtener una pieza determinada, se deberán realizar los cortes que se tracen al seguir el camino desde la raíz del árbol hasta el nodo hoja en el que se encuentre dicha pieza (ver Figura 13).

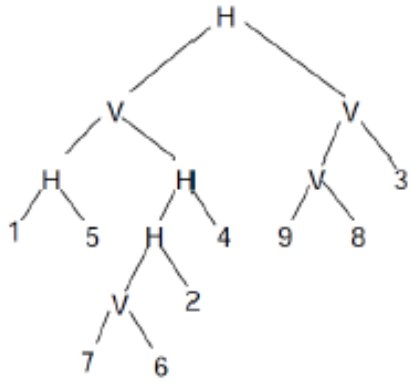


Figura 13. Representación en forma de árbol

Nota: Tener en cuenta que esta solución base asume un stock muy grande donde inmediatamente después se ejecuta un algoritmo exacto para lidiar con el stock múltiple y los defectos.

4.4.3 Representación en stock múltiple

A partir de una solución, que es como si se hubiera resuelto el problema considerando un stock grande (como la unión de todo el material disponible), se procederá a descomponer los bloques en varios stocks. Esta estrategia es válida ya que cada bloque cumplirá con la restricción de corte guillotinado y además para la división se considera colocar primero los bloques más grandes, de modo que ayudaría a minimizar el residuo por cada stock.

Dado la forma de su construcción, cada rama del árbol representa un bloque de piezas o una pieza (ver Figura 14 y Figura 15). Cada uno de dichos bloques es una solución válida y será colocada en un material en stock, tomando como consideración el área de dichos bloques y del material en stock.

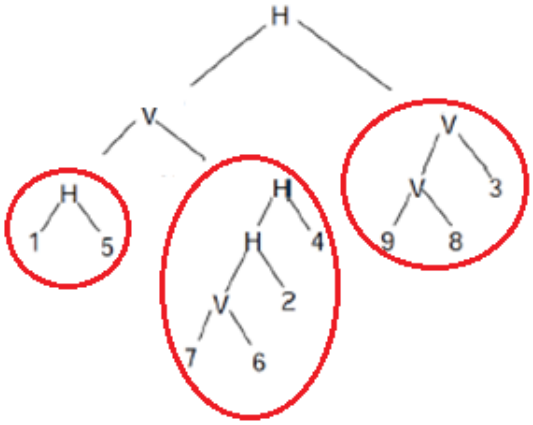


Figura 14. Representación en árbol en bloques

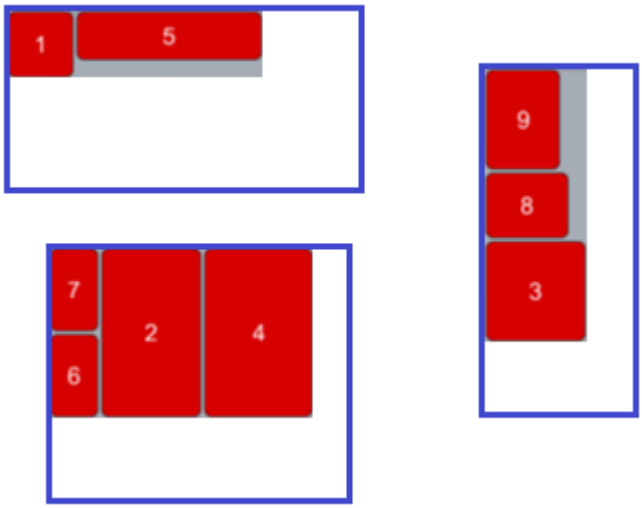


Figura 15. Representación gráfica en bloques

4.4.3.1 Algoritmo de stock múltiple

A continuación, se plantea el algoritmo exacto que fue utilizado para dividirlo en varios stocks. La idea central del algoritmo es colocar los bloques de piezas más grandes en los stocks de mayor dimensión, generando todos los bloques posibles (subárboles) y teniendo en cuenta no colocar dos veces una misma pieza.

Tabla 19. Seudocódigo de algoritmo de distribución en stocks

Inicio AlgoritmoDistribucionStocks(arbol, listaStocks)

1. listaNodos = CrearListaBloques(arbol) // obtiene todos los posibles subárboles, por lo que la lista será igual al número de nodos del árbol
2. OrdernarDescendientePorArea(listaNodos)
3. OrdernarDescendientePorArea(listaStocks)
4. nroStock = 0
5. **Mientras** NoEstaVacia(listaNodos) **hacer**
   a. nodo = listaNodos[0]
   b. **Si** EsCubierto(Nodo, listaStocks[nroStock]) **entonces**
      i. listaStocks[i] = nodo
      ii. EliminarSubArboles(listaNodos, nodo)
      iii. nroStock = nroStock + 1
      Fin Si
   c. RemoverNodo(listaNodos, nodo)

Fin Mientras

Fin AlgoritmoDistribucionStocks

Nota: Nodo representa a un conjunto de piezas o incluso a solo una pieza (es un árbol).

Se explican las siguientes líneas:

- Línea 1: Crea una lista de todos los árboles posibles de un árbol binario.
- Líneas 2-3: Ordena la lista de árboles y stocks por área ocupada.
- Líneas 4-5: Establece el índice el stock que se está considerando para recorrer la lista de stocks en un bucle.
    - Línea 5-a: El primer candidato será el stock en la posición 0 pues es el más grande (recordar que ordenó la lista en las líneas 2-3).
    - Línea 5-b: Si el stock es un candidato (puede cubrir el bloque de piezas), entonces entra en la condición
        - Línea 5-b-i: Coloco el bloque de piezas en el stock candidato.
        - Línea 5-b-ii: Elimino los subárboles hijos de dicho nodo. Debido a que está ordenado por área, se asegura que ningún subárbol hijo se considerará antes que el árbol padre y se evitará volver a colocar la misma pieza en otro stock.
        - Línea 5-b-iii: Se incrementar el índice de stock.
    - Línea 5-c: Se debe remover el nodo de la lista (es decir, el nodo en la posición 0), pues o bien porque ya se colocó en un stock o es demasiado grande y no se pudo colocar.

4.4.4 Algoritmo de defectos

La idea principal de este algoritmo es, una vez distribuido las piezas en stocks, recorrer cada uno de los stocks para verificar que ninguna haya caído en regiones con defecto. Si se diera el caso, lo que hace es colocar la pieza afectada en otro stock, con el fin de satisfacer el pedido.

Tabla 20. Seudocódigo de algoritmo de distribución en stocks

```
Inicio AlgoritmoDefectos(listaStocks)
1. Para cada stock de listaStocks hacer
    a. CalcularPosicionesAbsolutas(stock.Arbol, 0, 0)
    b. listaPiezas_Stock = ConvertirALista(stock.Arbol)
    c. Para cada pieza de listaPiezas_Stock hacer
        i. listaDefectos = ObtenerListaDefectos(stock)
        ii. Si SeEncuentraEnLista(pieza, listaDefectos) entonces
            1. InsertarPiezaEnUnStock(pieza, listaStocks)
           Fin Si
    Fin Para
  Fin Para
Fin AlgoritmoDefectos
```

Se explican las siguientes líneas:

- Línea 1: Se recorre cada uno de los stocks para verificarlos.
  - Línea 1-a: Se calcula las posiciones absolutas de las piezas en un stock, pues hasta ese momento se tenían de forma relativa (respecto a otro bloque). Es recorrer nuevamente el árbol binario, en cualquier orden.
  - Línea 1-b: Se convierte las piezas en un stock en una lista.
  - Línea 1-c: Se corre cada una de las piezas del stock considerado en un bucle
    - Línea 1-c-i: Obtiene la lista de defectos asociados a un stock.
    - Línea 1-c-ii: Se verifica que una pieza no se intersecte con un stock (ambos representados como piezas), comparando sus posiciones absolutas.
      - Línea 1-c-ii-1: Si una pieza se cae en una región con defecto, se inserta en el stock inmediato disponible.

Capítulo 5. Diseño del algoritmo genético

5.1 Introducción

Este este capítulo abordaremos los **resultados esperados 5 y 6 (R 5 y R 6)**, el cual consiste en la definición de las estructuras de datos y el diseño del algoritmo genético para resolver el problema.

5.2 Estructuras de datos

En esta sección se presentarán las estructuras de datos adicionales a las de la solución y que son propias del diseño del algoritmo genético.

5.2.1 Clase cromosoma

Esta es la que representará una solución del problema considerando un stock grande, donde la solución es una lista de cadenas representada por la variable listaGenes (la misma utilizada en la representación de la solución), un valor de fitness y una variable árbol de tipo Nodo (el mismo que se utilizado en la representación de la solución, ver sección 4.3.6).

```
class Cromosoma
{
    List<String> listaGenes;
    double fitness;
    Nodo arbol;
}
```

5.2.2 Clase población

Esta clase permitirá representar un conjunto de individuos, representados por una lista de cromosomas, así como otras variables como la generación en la que se encuentra, la probabilidad de mutación y el tamaño de la población, que son parámetros del problema.

```
class Poblacion
{
    int tamanhoPoblacion;
    double probabilidadMutacion;
    int generacion;
    List<Cromosoma> poblacion = new List<Cromosoma>();
}
```

5.3 Diseño del algoritmo

A continuación, se muestra el pseudocódigo general del algoritmo genético:

Tabla 21. Seudocódigo del algoritmo genético adaptado al problema

Inicio AlgoritmoGenético(tamañoPoblacion, probabilidadMutacion, elitismo)
1. poblacion = ConstruirPoblaciónInicial(tamañoPoblacion)
2. nroGeneracion = 0
3. **Mientras** contador < MÁXIMO_GENERACIONES **entonces**
   a. nueva_generacion = Vacia
   b. Reemplazo (población, nueva_generacion, elitismo)
   c. **Mientras** tamaño (nueva_generacion) < tamañoPoblacion **hacer**:
      i. progenitor1 = Selección (población)
      ii. progenitor2 = Selección (población)
      iii. descendientes = crossover (progenitor1, progenitor2)
      iv. descendiente1 = descendientes[0]
      v. descendiente2 = descendientes[1]
      vi. agregar_individuo (nueva_generacion, descendiente1)
      vii. agregar_individuo (nueva_generacion, descendiente2)
      Fin Mientras
   d. MutarPoblacion (nueva_generacion, probabilidadMutacion)
   e. población = nueva_generacion
   f. contador = contador + 1
   Fin Mientras
4. **Retornar** mejor_individuo(población)

Fin AlgoritmoGenético

Se explican las siguientes líneas:

- Línea 1: Se construye una población inicial de manera aleatoria. Para mayor detalle ver la sección 5.3.2.
- Línea 2: Se inicializa una variable nroGeneracion en 0, que indica la cantidad de generaciones a calcular y en la que se encuentra.
- Línea 3: Se calculan nuevas generaciones hasta un máximo número de generaciones.
  - Línea 3.a: Se inicializa la nueva generación vacía.
  - Línea 3.b: Se aplica una función de elitismo para asegurarme que los mejores individuos de la generación actual se encuentren en la nueva generación. Para mayor detalle ver la sección 5.3.6 Método de reemplazo.
  - Línea 3.c: Se itera hasta que el tamaño de la nueva generación tenga la cantidad de individuos especificada por la variable tamañoPoblacion. El objetivo de este bucle es ir generando nuevos individuos.

        - Líneas 3.c.i-3.c.ii: De la población actual, se seleccionan dos individuos a través de un método de selección (ver sección 5.3.3 Método de selección), que en este caso es el de la ruleta.
        - Líneas 3.c.iii: Se cruzan los dos individuos seleccionados en el paso anterior a través del método Crossover (ver sección 5.3.4 Método de cruce) y devuelve una lista de dos individuos nuevos a partir de ellos.
        - Líneas 3.c.iv-3.c.v: Se asignan estos individuos a dos variables: descendiente1 y descendiente2.
        - Líneas 3.c.vi-3.c.vii: Se añaden estos nuevos individuos (descendiente1 y descendiente2) a la generación que se está calculando (variable nueva_generación).
    - Línea 3.d: Se aplica la función de Mutación a la nueva generación en base a una probabilidad, dada como input del algoritmo (ver 5.3.5 Método de mutación).
    - Línea 3.e: Se simula el paso a la siguiente generación a través de la asignación de la nueva generación a la variable población.
    - Línea 3.f: Se incrementa el contador de la cantidad de generaciones.
- Línea 4: Finalmente se devuelve el mejor individuo de la última generación.

5.3.1 Función de aptitud

En este caso se utilizará la misma función objetivo definida en la sección 4.2.1:

$$\text{Fitness} = \left(1 - \frac{\sum_{i}^{N} \text{áreaPiezas}_{\text{stock}_i}}{\sum_{i}^{N} \text{área}_{\text{stock}_i}}\right) \times 100$$

La cual se implementa de la siguiente manera:

```
Inicio CalcularFitness(listaStocksConPiezas)
    1. areaTotalPiezas = 0
    2. areaTotalStock = 0
    3. Para cada stock de listaStocksConPiezas hacer
            a. areaTotalStock = areaTotalStock + Área(stock)
            b. listaPiezas_Stock = ConvertirAListaPiezas(stock.Arbol)
            c. Para cada pieza de listaPiezas_Stock hacer
                i. areaTotalPiezas = areaTotalPiezas + Área(pieza)
               Fin Para
       Fin Para
    4. Retornar (1 - (areaTotalPiezas / areaTotalStocks)) * 100
Fin CalcularFitness
```

Se explican las siguientes líneas:

- Líneas 1-2: Se inicializan variables acumuladoras que indica el área total utilizada por las piezas y el área total de los stocks.
- Línea 3: Se inicia un bucle para donde se itera cada stock:
  - Línea 3-a: Se acumula el área de cada stock en la variable areaTotalStock.
  - Línea 3-b: Se obtienen la lista de piezas de cada stock.
  - Línea 3-c: Se recorre cada pieza de la lista de piezas en un bucle.
    - Línea 3-c-i: Se acumula el área de cada pieza en la variable areaTotalPiezas.
- Línea 4: Se calcula y retorna el valor de la función objetivo, expresado como porcentaje de área con defecto.

5.3.2 Población inicial

Se construirá una población aleatoria, pero considerando varias restricciones para construir soluciones válidas. Una de las principales es tomar el número de operadores de integración (H o V) que puede estar en una solución:

$$\mathrm{No} = \mathrm{Np} - 1$$

Donde: - No: Número de operadores
- Np: Número de piezas

Adicionalmente, dadas un número determinado de piezas, como es problema de corte guillotinado, se necesita como mínimo un operador para formar un bloque entre dos piezas. Siguiendo esta idea de construcción, se puede conocer la longitud del cromosoma, expresada por la fórmula:

$$\mathrm{Ng} = \mathrm{Np} + \mathrm{No} = 2\mathrm{Np} - 1$$

Donde: - Ng: Número de genes (longitud del cromosoma)
- Np: Número de piezas
- No: Número de operadores de integración (H o V)

Otra condición por verificar para evitar formar cromosomas no válidos se relaciona con la posición relativa entre las piezas y los operadores, es decir, se debe cumplir que tomando un operador N (tipo H o V), la cantidad de piezas a la izquierda de dicho

operador (representado por Np) y la cantidad de operadores a la izquierda de dicho operador (representado por No) deben cumplir la siguiente condición:

$$1 \leq \mathrm{No} \leq \mathrm{Np} - 1$$

A continuación, se muestra el pseudocódigo para construir un individuo, el cual recibe como valor de entrada el número de piezas (representando por la variable np). Este método será utilizado hasta obtener el tamaño de la población.

```
Inicio CrearIndividuoAleatorio(np)
 1. Lista(Cadena) cromosoma = vacia
 2. longitud_cromosoma = np * 2 - 1
 3. ng = longitud_cromosoma
 4. no = np – 1
 5. listaPiezas = vacia
 6. Para i=1 hasta np hacer
      a. ListaPiezas = agregar(i)
    Fin Para
 7. cromosoma.introducirPiezaAleatoria(listaPiezas)
 8. cromosoma.introducirPiezaAleatoria(listaPiezas)
 9. Para i = 3 hasta ng hacer
      a. actual_np = cantidadPiezas(cromosoma[i])
      b. actual_no = cantidadOperadores(cromosoma[i])
      c. Si (no == np + 1) entonces
           1. cromosoma.introducirPiezaAleatoria(listaPiezas)
      d. Sino
           1. Si ListaPiezas está vacía entonces
                a. cromosoma.introducirOperadorAleatorio()
           2. Sino
                b. aleatorio = SeleccionarAleatorio(“pieza”, “operador”)
                c. Si aleatorio = “pieza” entonces
                     i. cromosoma.introducirPiezaAleatoria(listaPiezas)
                d. Sino
                     i. cromosoma.introducirOperadorAleatorio()
                     Fin Si
              Fin si
        Fin si
    Fin Para
10. Retornar cromosoma
Fin CrearIndividuoAleatorio
```

Se explican las siguientes líneas:

- Línea 1: Se crea un cromosoma vacío, como una lista de cadenas.
- Línea 2: Se inicializa la longitud del cromosoma.
- Línea 3: Se inicializa el número de genes
- Línea 4: Se inicializa el número de operadores
- Línea 5: Se inicializa una lista de piezas vacía.

- Línea 6: Se itera sobre todas las piezas dadas.
    - Línea 6.a: Se agregar un número a la lista, que representará una pieza.
- Línea 7: Se introduce el primer gen al cromosoma de la lista, aleatoriamente.
- Línea 8: Se introduce el segundo gen al cromosoma de la lista, aleatoriamente.
- Línea 9: Se itera a partir de la posición i=3 hasta el tamaño del cromosoma:
    - Línea 9.a: Se cuenta la cantidad de piezas a la izquierda del cromosoma (desde el inicio hasta la posición i)
    - Línea 9.b: Se cuenta la cantidad de operadores a la izquierda del cromosoma (desde el inicio hasta la posición i)
    - Línea 9.c: Si el número de operadores es igual al número piezas más uno, estamos en el límite de la condición y solo podemos introducir una pieza.
        - Línea 9.c.1: Introducimos una pieza aleatoria al cromosoma de la lista de piezas, con una probabilidad de 50% de que sea rotada o no.
    - Línea 9.d: Si aún no estamos en el límite, entonces podemos agregar un gen, el cual puede ser operador o una pieza.
        - Línea 9.d.1: Si no hay más piezas para introducir al cromosoma.
            - Línea 9.d.1.a: Se introduce un operador aleatorio al cromosoma.
        - Línea 9.d.2: Si hay piezas restantes, entonces hay una probabilidad de introducir o bien una pieza o bien un operador.
            - Líneas 9.d.2.b - 9.d.2.d: Se introduce un gen al cromosoma, considerando una probabilidad de 50% de que sea pieza o de que sea operador, y otra probabilidad de 50% de que la pieza sea rotada o no.
- Línea 10: Se devuelve el cromosoma generado.

5.3.3 Método de selección

El método de selección utilizado es el de selección por rueda de la ruleta (roulette wheel selection). Este método de selección establece una probabilidad de selección que es equivalente a la proporción de aptitud que un individuo tiene con respecto a la aptitud de la población total. La siguiente fórmula determina la probabilidad de un individuo dado a ser elegido:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i}$$

Donde:

- p_i es la probabilidad de que el individuo sea seleccionado.
- f_i es el valor de aptitud del individuo i.

Tabla 22. Seudocódigo del método de selección

```
Inicio Selección (poblacion)
  2. Suma_Fitness_Poblacion = 0
  3. Para cada individuo de Poblacion hacer
       a. Suma_Fitness_Poblacion = individuo.fitness
     Fin Para
  4. R = aleatorio (0, Suma_Fitness_Poblacion)
  5. Suma_Fitness_Recorrer = 0
  6. Para cada individuo de Poblacion hacer
       a. Suma_Fitness_Recorrer = Suma_Fitness_Reccorrer + individuo.fitness
       b. Si Suma_Fitness_Recorrer > Suma_Fitness_Poblacion entonces
            i. Retornar individuo
          Fin Si
     Fin Para
Fin Selección (población)
```

Se explican las siguientes líneas:

- Líneas 1-2: Tiene como objetivo calcular el fitness total de toda la población, para ello se recorren todos los individuos y se guarda en la variable Suma_Fitness_Poblacion.
- Línea 3: Tiene como objetivo seleccionar un número aleatorio entre 0 y el total del fitness de la población, queriendo representar la probabilidad de que se seleccione el individuo en base a dicho valor.
- Línea 4: Se inicializa la una variable para calcular el fitness acumulado hasta el momento.
- Línea 5: Se recorre cada individuo de la población
  - Línea 5.a: Se acumula el valor del fitness de los individuos hasta el individuo considerado en la variable Suma_Fitness_Recorrer
  - Línea 5.b: Si la suma del fitness hasta el individuo considerado es mayor que la suma total del fitness de la población, entonces:
    - Línea 5.b.i: Devolver el individuo considerado y terminar la función

5.3.4 Método de cruce

El método de cruce involucra a dos progenitores seleccionados mediante la técnica de la rueda de la ruleta para formar dos descendientes.

Dado que los genes que indican números de pieza y los genes que indican operadores no comparten las mismas características y restricciones en el cromosoma. Se seleccionan todos los operadores por un lado y todos los números de pieza por otro para combinarlos con los de otro individuo y después unirlos para formar los siguientes individuos.

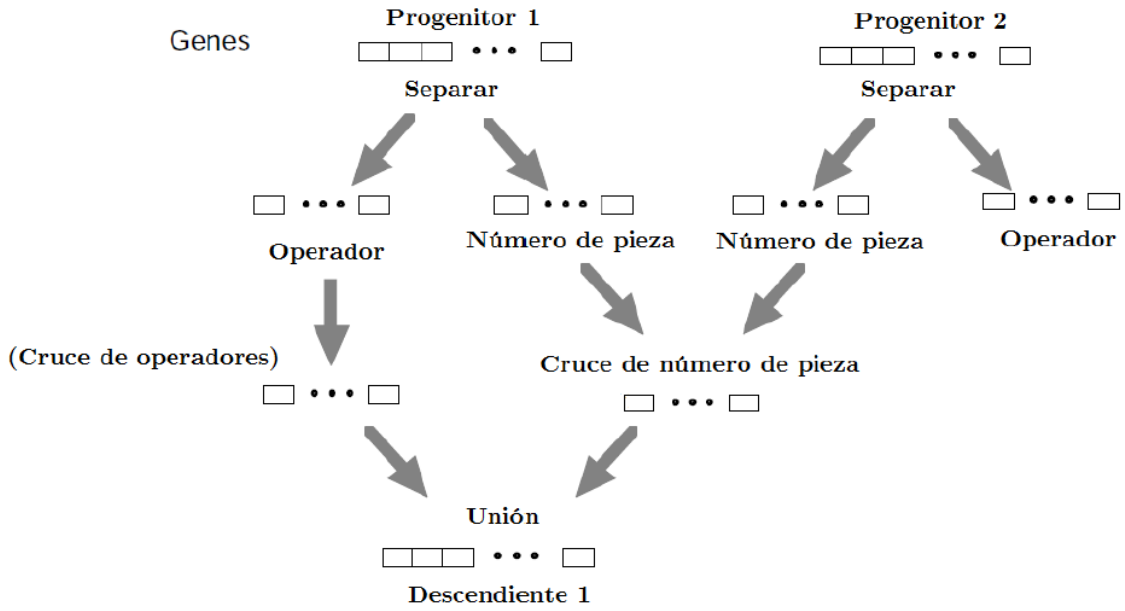


Figura 16. Estrategia de cruce de dos progenitores

En vista de que un intercambio de genes entre dos progenitores no es posible, dado que podrían dar lugar a soluciones no válidas de manera muy rápida, se tienen que utilizar técnicas de combinación cuando los cromosomas tiene más de un tipo de gen. Uno de los más efectivos es el Partially-mapped crossover, el cual explicaremos a continuación.

Dados dos progenitores, como los siguientes:

Progenitor 1	1	3	H	2	4	V	H	5	6	V	V
Progenitor 2	2	3	V	4	V	1	H	5	V	6	H

Figura 17. Progenitores a realizar el cruce

Se separan los genes que indican los números de pieza de ambos progenitores.

1	3	2	4	5	6
2	3	4	1	5	6

Figura 18.Lista de piezas compacta de ambos progenitores

Se realiza un intercambio entre dos posiciones aleatorias, por ejemplo, las posiciones 3 y 4 de ambos arreglos, del cual se obtiene el siguiente resultado.

1	3	4	1	5	6
2	3	2	4	5	6

Figura 19. Lista de piezas de ambos progenitores intercambiada

Se validan los resultados y si se detecta alguna solución inválida, se realiza un intercambio entre las piezas que no fueron intercambiadas. En este caso particular, vemos que la pieza 1 y pieza 2 se encontraban más de una vez en un mismo progenitor. Por tanto, se intercambian dichas regiones afectadas para tener la siguiente solución válida, de modo que obtenemos los nuevos descendientes (parte pieza):

2	3	4	1	5	6
1	3	2	1	5	6

Figura 20. Lista de piezas de ambos progenitores validada

Se puede realizar la misma operación de las piezas para los genes del tipo operador e incluso se pueden dejar en la misma posición, ya que al estar intercambiadas las piezas, al unirse con los operadores ya forman una nueva solución, el cual es el objetivo el algoritmo. Por lo tanto, a partir de la unión de la parte de pieza con la parte de operadores se obtienen los nuevos descendientes:

2	3	H	4	1	V	H	5	6	V	V
1	3	V	2	V	1	H	5	V	6	H

Figura 21. Nuevos descendientes

A continuación, se muestra el seudocódigo de método de cruce.

Tabla 23. Seudocódigo de método de cruce

```
Inicio Crossover(padre1, padre2)
 1. progen1_ListaPiezas = SepararPiezas(padre1, padre2)
 2. progen2_ListaPiezas = SepararPiezas(padre1, padre2)
 3. progen1_ListaOperadores = SepararOperadores(padre1, padre2)
 4. progen2_ListaOperadores = SepararOperadores(padre1, padre2)
 5. primerCorte = aleatorio(1, tamaño(padre1)-1)
 6. segundoCorte = aleatorio(primerCorte, tamaño(padre1)
 7. IntercambiarPiezas(padre1_ListaPiezas, padre2_ListaPiezas,
                       primerCorte, segundoCorte)
 8. CorregirPiezas(padre1_ListaPiezas, padre2_ListaPiezas,
                   1, primerCorte-1)
 9. CorregirPiezas(padre1_ListaPiezas, padre2_ListaPiezas,
                   segundoCorte+1, tamaño(progenitor1) - 1)
10. hijo1 = UnirOperadoresYPiezas(padre1_ListaPiezas, padre1_ListaOperadores)
11. hijo2 = UnirOperadoresYPiezas(padre2_ListaPiezas, padre2_ListaOperadores)
12. descendientes = Lista(hijo1, hijo2)
13. Retornar descendientes
Fin Crossover
```

Se explican las siguientes líneas:

- Línea 1-2: Se obtiene una lista compacta de piezas (sin operadores) de ambos cromosomas y guardando el orden existente entre las mismas.
- Línea 3-4: Se obtiene una lista de operadores de ambos cromosomas, guardando los espacios entre operadores correspondientes a las piezas y guardando el orden existente los operadores.
- Línea 5-6: Se seleccionan dos puntos de corte aleatorio de las piezas, que será el segmento del cromosoma a intercambiar.
- Línea 7: Se realiza el intercambio directo entre la lista de piezas, delimitado por el segmento definido entre los puntos de corte.
- Línea 8: Se corrigen las piezas intercambiadas de los dos hijos que se están formando en el exterior del rango definido por los puntos de corte, en este caso del segmento izquierdo.
- Línea 9: Se corrigen las piezas intercambiadas de los dos hijos que se están formando en el exterior del rango definido por los puntos de corte, en este caso del segmento derecho.
- Línea 10: Se forma el primer hijo, colocando las piezas intercambiadas del padre1 en los espacios vacíos dejados por los operadores del padre1.
- Línea 11: Se forma el segundo hijo, colocando las piezas intercambiadas del padre2 en los espacios vacíos dejados por los operadores del padre2.
- Línea 12: Se crear una lista que contiene a los dos hijos.

- Línea 13: Se devuelve la lista de nuevos descendientes.

5.3.5 Método de mutación

La mutación adoptada consiste en el intercambio de los elementos de dos genes elegidos aleatoriamente, sean genes que contengan un operador o un número de pieza.

Esta mutación se debe aplicar a todos los individuos excepto a los mejores individuos, una vez se ha formado la nueva generación, constituida íntegramente por la descendencia obtenida en los cruces.

- Sean p1 y p2 números de pieza.
- p1 es un operador, independientemente del tipo que sea p2.
- p1 es un número de pieza, p2 es un operador y después de ser intercambiados, entonces debe cumplirse:

$$\mathrm{No} \leq \mathrm{Np} - 3$$

Tabla 24. Seudocódigo del método de mutación

Inicio Mutacion (cromosoma)
1. gen1 = aleatorio (cromosoma)
2. gen2 = aleatorio (cromosoma)
3. **Si** tipo(gen1) = ”Pieza” y tipo(gen2)=”Pieza” **entonces**
   a. Intercambiar (gen1, gen2)
   b. **Retornar** cierto
4. **Sino si** tipo(gen1) = ”Operador” **entonces**
   a. Intercambiar (gen1, gen2)
   b. **Retornar** cierto
5. **Sino si** tipo(gen1) = ”Pieza” y tipo(gen2) = “Operador” **entonces**
   a. **Si** VerificarPosicionRelativa(gen1, gen2, cromosoma) **entonces**
      i. Intercambio (gen1, gen2)
      ii. **Retornar** cierto
   b. **Sino**
      iii. **Retornar** falso
   Fin Si

Fin Si
Fin Mutacion

Se explican las siguientes líneas:

- Líneas 1-2: Inicializan dos genes aleatorios de un cromosoma.
- Línea 3: Si ambos genes son piezas entonces
  - Línea 3.a-3b: Intercambiar los genes y retornar cierto.
- Línea 4: Si el primer gen es un operador, entonces

- Línea 4.a-4b: Intercambiar los genes y retornar cierto.

- Línea 5: Si el primer gen es una pieza y el segundo gen es un operador, entonces
  - Línea 5.a: Si se cumple la posición relativa entre ambos genes entonces:
    - Línea 5.a.1-4.a.2: Intercambiar los genes y retornar cierto.
  - Línea 5.b: Si no se cumple la posición relativa entre ambos genes entonces:
    - Línea 5.a.1-4.a.2: No intercambiar ningún gen y retornar falso.

Como vimos, en la Línea 5.a se hace uso de la función VerificarPosicionRelativa, donde la idea es verifica la restricción $\mathrm{No} \leq \mathrm{Np} - 3$, para que sea una solución válida. El seudocódigo de dicha función se muestra a continuación:

Tabla 25. Seudocódigo del método VerificarPosicionRelativa

```
Inicio VerificarPosicionRelativa (gen1, gen2, listaGenes)
 1. cantOperadores = 0;
 2. cantPiezas = 0;
 3. Para cada gen de listaGenes hacer:
     a. Si tipo(gen)=pieza entonces
          i. cantPiezas = cantPiezas + 1
     b. Sino
          i. cantOperadores = cantOperadores + 1
        Fin Si
     c. Si (pos(gen) >= pos(gen1) y pos(gen) <= pos(gen2))
          i. Si no se cumple que (cantOperadores <= cantPiezas - 3) entonces
               1. Retornar falso
             Fin Si
        Fin Si
    Fin Para
 4. Retornar cierto
Fin VerificarPosicionRelativa
```

Se explican las siguientes líneas:

- Líneas 1-2: Inicializa las variables de contadores de piezas y operadores
- Línea 3: Se itera para cada gen de la lista de genes
  - Línea 3.a: Si el tipo(gen) = “Pieza” entonces
    - Línea 3.a.i: Incrementar el contador de piezas.
  - Línea 3.b: Si el tipo(gen) <> “Pieza”
    - Línea 3.b.i: Incrementar el contador de operadores.
  - Línea 3.c: Si se cumple que el gen considerado se encuentra entre los genes gen1 y gen2
    - Línea 3.c.i: Si no se cumple la condición de la cantidad de operadores menor o igual que la cantidad de piezas menos 3, entonces no es una solución válida y retorna falso.

- Línea 4: Finalmente si cumple todas las condiciones, devuelve cierto.

Este método estará dentro del siguiente bucle para el cálculo de una generación, el cual básicamente iterará dependiente de la probabilidad de mutación y hasta que haya una mutación posible.

Tabla 26. Seudocódigo del método MutarPoblacion

```
Inicio MutarPoblacion (población, probabilidadMutacion)
  1. Para cada individuo de población hacer
       a. Si (aleatorio < probabilidadMutacion) entonces
            i. Mientras no sea posible Mutacion (individuo) hacer
                 1. Mutacion (individuo)
                 Fin Mientras
          Fin Si
     Fin Para
Fin MutarPoblacion
```

Se explican las siguientes líneas:

- Línea 1: Se ordena la población de mayor a menor fitness.
  - Línea 1.a: Si el individuo tiene probabilidad de mutación, entonces se muta.
    - Línea 1.a.i: Esta línea sigue iterando hasta que se pueda mutar al individuo, apoyándose en la función Mutacion.

5.3.6 Método de reemplazo

El método de reemplazo consiste en reemplazar por la descendencia generada por un número fijado del algoritmo, de manera de quedarme con los mejores individuos. Como recomendación, se puede seleccionar de uno a 5 de los mejores individuos de una generación. Este concepto se conoce como elitismo, es decir, que durante el transcurso de las generaciones deberán sobrevivir estos mejores individuos. Para este caso, dicho parámetro viene expresado por la variable numeroElitismo, es decir, esta cantidad de los mejores de la generación estarán en la siguiente.

```
Inicio Reemplazo (población, sobrevivientes, numeroElitismo)
 1. OrdenarMenorAMayorFitness(poblacion)
 2. Lista<Cromosoma> sobrevivientes = vacia
 3. Para i=0 hasta numeroElitismo-1 hacer:
      a. sobrevivientes(población[i])
    Fin Para
Fin Reemplazo
```

Se explican las siguientes líneas:

- Línea 1: Se ordena la población de menor a mayor fitness.

- Línea 2: Se crea una lista de sobrevivientes a la siguiente generación.
- Línea 3: Se itera para según la cantidad de elitismo, de 0 a numeroElitismo-1.
    - Línea 3.a: Se agrega el primer elemento i de la población a la lista de sobrevivientes, considerando que la población está ordenada por fitness.

Capítulo 6. Diseño del algoritmo Cuckoo Search

6.1 Introducción

Este este capítulo abordaremos los **resultados esperados 3 y 4 (12R 3 y R 4)**, el cual consiste en la definición de las estructuras de datos y el diseño del algoritmo Cuckoo Search para resolver el problema. El algoritmo se basa en las siguientes reglas principales:

1. Cada cuco pone un único huevo cada vez, y lo deja en un nido elegido al azar.
2. Los mejores nidos con mayor calidad del huevo pasarán a la siguiente generación.
3. El número de nidos disponibles en cada generación (n) está predefinido, y los huevos serán descubiertos y descartados con una probabilidad pa del total de los nidos será sustituida por nidos nuevos cada generación.

6.2 Estructuras de datos

6.2.1 Clase nido

Tiene una lista de cadenas, representación de la solución y una variable árbol, representación de la lista de cadenas (postfija) en forma de árbol binario. Asimismo, se guarda el fitness pues es un valor muy utilizado.

Esta clase se puede generalizar de modo que un Nido contenga una lista de huevos de Cuckoo, sin embargo, en este proyecto seguiremos la idea del autor del algoritmo (Yang & Deb, 2009), en donde considera que "*cada nido contiene un solo huevo de Cuckoo"*, por lo que el nido será la solución al problema.

```
class Nido
{
    List<String> listaStr;
    Nodo arbol;
    double fitness;
    List<Stock> listaStocksConPiezas;
}
```

Se utiliza una lista de cadenas para representar la solución. A continuación, se muestra un ejemplo de dicha estructura.

Posición	0	1	2	3	4	5	6	7	8	9	…	N - 1
Valor	1R	5N	H	7N	6N	V	2N	H	4R	H	…	H

donde:

- 1R: representa a la pieza 1 con rotación de 90°.
- 5N: representa a la pieza 5 sin rotación de 90°.

6.2.2 Lista de nidos

Esta estructura permitirá representar a un conjunto de individuos, representados como una lista de objetos de la clase Nido. La mejor solución de una generación se encontrará en dicha lista.

```
List<Nido> listaNidos;
```

6.3 Diseño del algoritmo

A continuación, se muestra el pseudocódigo general del algoritmo Cuckoo Search:

Tabla 27. Seudocódigo del algoritmo Cuckoo Search

Inicio AlgoritmoCuckooSearch(numMaxGeneraciones, p_a)

1. listaNidos = CalcularNidosIniciales(numNidos);
2. **Para** i = 0 **hasta** numMaxGeneraciones **hacer**
    a. nidoCuckoo_1 = SeleccionarNidoAleatorio(listaNidos)
    b. nidoCuckoo_1 = ActualizarViaVueloLevy(nidoCuckoo_1)
    c. nidoCuckoo_2 = SeleccionarNidoAleatorio(listaNidos)
    d. **Si** (Fitness(nidoCuckoo_1) > Fitness(nidoCuckoo_2)) **entonces**
        i. nidoCuckoo_2 = nidoCuckoo_1;
    Fin Si
    e. RemoverPeoresYGenerarNidos(listaNidos, p_a)
    f. OrdenarMenorAMayorFitness(listaNidos)

    Fin Para
3. **Retornar** listaNidos[0]

Fin AlgoritmoCuckooSearch

Se explican las siguientes líneas:

- Línea 1: Se calcula una lista de nidos iniciales de forma aleatoria.
- Línea 2: Se crea una variable i para iterar en cada generación.
    - Línea 2-a: Se selecciona un nido aleatorio de la lista de nidos.
    - Línea 2-b: Al nido seleccionado, se le hace una modificación a través de una función de que siga la distribución de Levy.

- Línea 2-c: Se selecciona un nido aleatorio de la lista de nidos.
- Línea 2-d: Si el nido modificado es mejor que el otro nido, entonces me quedo con el nido modificado.
- Línea 2-e: Se remueven una fracción (p_a) de los peores nidos porque han sido descubiertos.
- Línea 2-f: Se ordena la lista de nidos por fitness, de menor a mayor.

- Línea 3: Retornar el mejor nido, que es el que está la primera posición.

6.3.1 Función de aptitud

En este caso se utilizará la misma función objetivo definida en la sección 4.2.1:

$$\text{Fitness} = \left(1 - \frac{\sum_{i}^{N} \text{áreaPiezas}_{\text{stock}_i}}{\sum_{i}^{N} \text{área}_{\text{stock}_i}}\right) \times 100$$

La cual se implementa de la siguiente manera:

Inicio CalcularFitness(listaStocksConPiezas)
5. areaTotalPiezas = 0
6. areaTotalStock = 0
7. **Para cada** stock **de** listaStocksConPiezas **hacer**
    a. areaTotalStock = areaTotalStock + Área(stock)
    b. listaPiezas_Stock = ConvertirALista(stock.Arbol)
    c. **Para cada** pieza **de** listaPiezas_Stock **hacer**
        i. areaTotalPiezas = areaTotalPiezas + Área(pieza)
    Fin Para

    Fin Para
8. **Retornar** (1 - (areaTotalPiezas / areaTotalStocks)) * 100

Fin CalcularFitness

Se explican las siguientes líneas:

- Líneas 1-2: Se inicializan variables acumuladoras que indica el área total utilizada por las piezas y el área total de los stocks.
- Línea 3: Se inicia un bucle para donde se itera cada stock:
    - Línea 3-a: Se acumula el área de cada stock en la variable areaTotalStock.
    - Línea 3-b: Se obtienen la lista de piezas de cada stock.
    - Línea 3-c: Se recorre cada pieza de la lista de piezas en un bucle.
        - Línea 3-c-i: Se acumula el área de cada pieza en la variable areaTotalPiezas.
- Línea 4: Se calcula y retorna el valor de la función objetivo, expresado como porcentaje de área con defecto.

6.3.2 Calcular nidos iniciales

Los nidos iniciales se calculan de forma a similar a la construcción de la población en el algoritmo genético, es decir, se crean nidos de manera aleatoria hasta llegar al tamaño de la población, considerando las restricciones para construir soluciones válidas.

Para el cálculo de los nidos, se considera una optimización en la construcción de dichas soluciones, pues considerando que una parte importante del algoritmo Cuckoo Search es la construcción de nuevas soluciones en cada generación.

```
Inicio CrearNidoAleatorio(np)
  1. Lista(Cadena) nido = vacia
  2. longitud_nido = np * 2 - 1
  3. no = np – 1
  4. listaPiezas = vacia
  5. Para i=1 hasta np hacer
       a. ListaPiezas = agregar(i)
     Fin Para
  6. nido.introducirPiezaAleatoria(pieza)
  7. nido.introducirPiezaAleatoria(pieza)
  8. Para i = 3 hasta longitud_nido hacer
       a. actual_np = cantidadPiezas(nido[i])
       b. actual_no = cantidadOperadores(nido[i])
       c. Si (no == np + 1) entonces
            i. nido.introducirPiezaAleatoria(listaPiezas)
       d. Sino
            i. Si ListaPiezas está vacía entonces
                 1. IntroducirOperadorMayorArea(nido)
            ii. Sino
                 1. aleatorio = SeleccionarAleatorio("pieza", "operador")
                 2. Si aleatorio = "pieza" entonces
                      a. nido.introducirPiezaAleatoria(listaPiezas)
                 3. Sino
                      a. IntroducirOperadorMayorArea(nido)
                    Fin Si
               Fin si
          Fin si
     Fin Para
  9. Retornar cromosoma
Fin CrearIndividuoAleatorio
```

Se explican las siguientes líneas:

- Línea 1: Se crea un nido vacío, como una lista de cadenas.
- Línea 2: Se inicializa la longitud del nido.
- Línea 3: Se inicializa el número de operadores
- Línea 4: Se inicializa una lista de piezas vacía.

- Línea 5: Se itera sobre todas las piezas dadas.
    - Línea 5.a: Se agregar un número a la lista, que representará una pieza.
- Línea 6: Se introduce la primera pieza al nido, seleccionado aleatoriamente.
- Línea 7: Se introduce la segunda pieza al nido, seleccionado aleatoriamente.
- Línea 8: Se itera a partir de la posición i=3 hasta el tamaño del nido:
    - Línea 8.a: Se cuenta la cantidad de piezas a la izquierda del nido (desde el inicio hasta la posición i)
    - Línea 8.b: Se cuenta la cantidad de operadores a la izquierda del nido (desde el inicio hasta la posición i)
    - Línea 8.c: Si el número de operadores es igual al número piezas más uno, estamos en el límite de la condición y solo podemos introducir una pieza.
        - Línea 8.c.i: Introducimos una pieza aleatoria al nido de la lista de piezas, con una probabilidad de 50% de que sea rotada o no.
    - Línea 8.d: Si aún no estamos en el límite, entonces podemos agregar un elemento, el cual puede ser operador o una pieza.
        - Línea 8.d.i: Si no hay más piezas para introducir al nido.
            - Línea 8.d.i.1: Se introduce el operador que genere la mayor área usando la función IntroducirOperadorMayorArea.
        - Línea 8.d.ii: Si hay piezas restantes, entonces hay una probabilidad de introducir o bien una pieza o bien un operador.
            - Líneas 8.d.i.2 - 8.d.ii.3: Se introduce un elemento al nido, considerando una probabilidad de 50% de que sea pieza o de que sea operador (considerando el que genere mayor área usando la función IntroducirOperadorMayorArea), y otra probabilidad de 50% de que la pieza sea rotada o no.
- Línea 9: Se devuelve el nido generado.

A continuación, se muestra la función utilizada para realizar la optimización en las líneas 8.d.1.a y 8.d.i.2 del pseudocódigo anterior.

```
Inicio IntroducirOperadorMayorArea(nido)
    1. Area_H = nido.AreaConOperador("H")
    2. Area_V = nido.AreaConOperador("V")
    3. Si (Area_H >= Area_V)
        a. nido.introducirOperador("H")
    4. Sino
        a. nido.introducirOperador("V")
       Fin Si
```

```
Fin IntroducirOperadorMayorArea
```

- Línea 1: Se obtiene el área de integrar la última pieza con el operador “H”.
- Línea 2: Se obtiene el área de integrar la última pieza con el operador “V”.
- Línea 3: Si el área de integrar con el operador “H” es mayor entonces
  - Línea 3.a: Se introduce el operador “H” al nido (lista).
- Línea 4: De lo contrario, el área de integrar con el operador “V” es mayor y entonces
  - Línea 4.a: Se introduce el operador “V” al nido (lista).

6.3.3 Actualizar nido vía vuelo de Levy

Se hace una modificación de la solución actual, no tan aleatoria sino en base a una distribución de Levy. Para simular este movimiento se pueden hacer uso de diversos algoritmos con diferentes funciones de probabilidad (Kamaruzaman, Zain, Yusuf, & Udin, 2013). Para este proyecto, se decidió utilizar el algoritmo de Mantegna, el cual es utilizado por el autor del algoritmo Cuckoo Search (Yang, 2010).

```
Inicio Nido ActualizarViaVueloLevy(nido, listaPiezas)
 1. numStr = Tamaño(nido.ListaStr)
 2. beta = 3 / 2;
 3. numSigma = Gamma(1.0 + beta) * Seno(PI * beta / 2.0)
 4. denSigma = Gamma((1.0 + beta) / 2.0) * beta * Potencia(2.0, (beta - 1) / 2.0)
 5. sigma = Potencia (numSigma / denSigma, 1 / beta)
 6. cotaInf = 0
 7. cotaSup = numStr - 1;
 8. Para i = 0 hasta numStr hacer
      a. u = Aleatorio(0, numStr) * sigma
      b. v = Aleatorio(0, numStr)
      c. step = u / Potencia(ValorAbsoluto (v), 1 / beta)
      d. stepsize = 0.1 * step * (cotaSup - i)
      e. posStr1 = i
      f. posStr2 = i + stepsize * Aleatorio(0, numStr);
      g. posStr2 = Acotar(posStr2, cotaInf, cotaSup)
      h. IntercambiarElementos(nido.LlistaStr, posStr1, posStr2)
    Fin Para
 9. CalcularNido(nido, listaPiezas)
10. Retornar nido
  Fin ActualizarViaVueloLevy
```

Se explican las siguientes líneas:

- Líneas 1-7: Se calculan los exponentes y coeficientes de Levy, así como las cotas inferior y superior de cada movimiento.
- Línea 8: Se itera para cada elemento de la solución de un nido

- Línea 8-a-d: Se calcula el tamaño del paso.
- Línea 8-e-f: La posición pos1 será igual al elemento que se está recorriendo, mientras que la posición pos2 será igual a pos1 más el paso.
- Línea 8-g: Se acota la variable pos2 si el paso es muy grande y se sale del límite.
- Línea 8-h: Se realiza el camino aleatorio del nido con las posiciones pos1 y pos2.

- Línea 9: Se completa la información del nido (posiciones absolutas, fitness).
- Línea 10: Se devuelve el nido modificado

6.3.4 Remover peores y generar nidos

Esta función es la que le agregar el carácter de elitismo al algoritmo, en cual consiste en remover un porcentaje de los nidos en base a una probabilidad pa, simulando la idea de que fueron "descubiertos" por el ave dueña del nido. Se abandona el nido como consecuencia y se crea otro nuevo.

```
Inicio RemoverPeoresYGenerarNidos(listaNidos, p_a)
  1. cantNidos = TamañoLista(listaNidos)
  2. OrdenarMenorAMayorFitness(listaNidos)
  3. inicio = cantNidos * (1 - p_a))
  4. Para j=inicio hasta cantNidos hacer
       a. listaNidos[j] = CrearNido(numStr)
       b. CalcularFitness(listaNidos[j], listaPiezas)
     Fin Para
Fin RemoverPeoresYGenerarNidos
```

Se explican las siguientes líneas:

- Línea 1: Se calcula la cantidad de nidos.
- Línea 2: Se ordena la lista de mayor a menor por fitness.
- Línea 3: Se calcula el inicio de la lista nidos a partir de donde se deberá remover una fracción p_a de nidos.
- Línea 4: Se crear una variable índice j para recorrer los nidos
  - Línea 3-a: Se crear un nuevo nido en la posición j.
  - Línea 3-b: Se calcula información como distribución en stocks y defectos y su fitness.

Capítulo 7. Interfaz gráfica

7.1 Introducción

En este capítulo se presentará la interfaz que se utilizará para mostrar los resultados de los algoritmos.

Este este capítulo se abordan los **resultados esperados 7, 8, 9 y 11 (R 7, 12R 8, R 9 y R 11)**, los cuales están relacionados al desarrollo de una interfaz gráfica la cual permita la entrada de datos del problema, ejecución de los algoritmos y visualización de los resultados del problema.

7.2 Módulo pedidos

En esta sección de aborda específicamente el resultado esperado 7 (R 7), el cual consiste un módulo de carga para la pedidos y material en stock. Consta de dos campos donde se solicita la ruta de los archivos de entrada (pedidos y stock), además de botones que llevarán a los siguientes módulos de los algoritmos genético y Cuckoo Search.

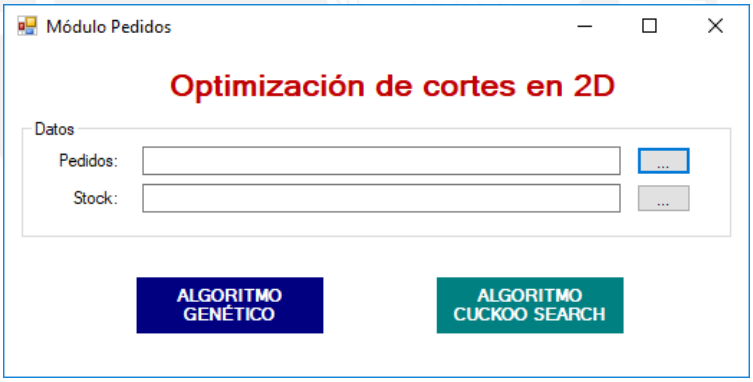
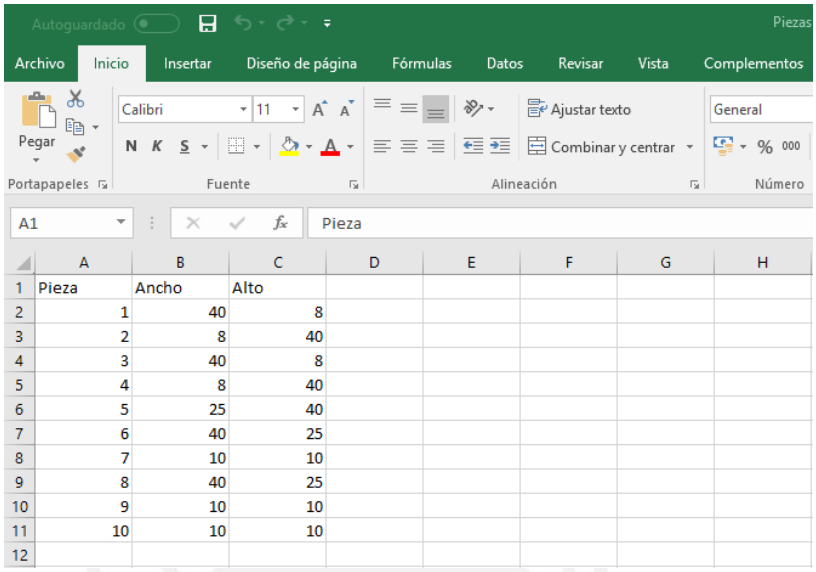


Figura 22. Interfaz del módulo de pedidos

7.2.1 Archivos de entrada

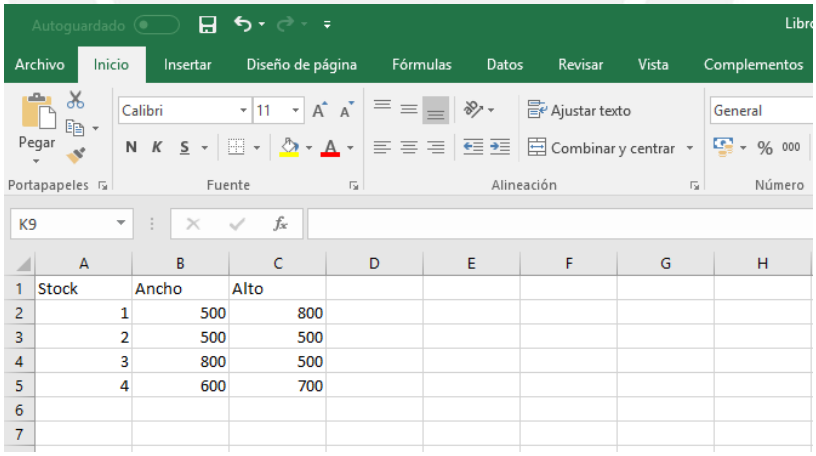
1. **Pedidos:** Este será un archivo de Excel el cual se indicarán las siguientes variables:
   - Pieza: El número de la pieza como identificador.
   - Ancho: El ancho de la pieza solicitada, en cm.
   - Alto: El alto de la pieza solicitada, en cm.



	A	B	C	D	E	F	G	H
1	Pieza	Ancho	Alto					
2	1	40	8					
3	2	8	40					
4	3	40	8					
5	4	8	40					
6	5	25	40					
7	6	40	25					
8	7	10	10					
9	8	40	25					
10	9	10	10					
11	10	10	10					
12								

Figura 23. Archivo de pedidos

2. **Stocks:** Este será un archivo de Excel el cual se indicarán las siguientes variables:
   - Stock: El número del material en stock como identificador.
   - Ancho: El ancho del material en stock, en cm.
   - Alto: El alto del material en stock, en cm.



	A	B	C	D	E	F	G	H
1	Stock	Ancho	Alto					
2	1	500	800					
3	2	500	500					
4	3	800	500					
5	4	600	700					
6								
7								

Figura 24. Archivo de stocks.

7.3 Módulo del Algoritmo Genético

En esta sección de aborda específicamente el resultado esperado 9 (R 9), el cual consiste un módulo para los parámetros del algoritmo genético. Consta de 4 campos:

- Probabilidad de mutación: Un número entre 0 y 1.
- Tamaño de la población: Un número entero.

- Cantidad de elitismo: Un número entero, indicando cuantos individuos sobreviven de cada generación.
- Cantidad de generaciones: La máxima cantidad de iteraciones del algoritmo.

Figura 25. Módulo algoritmo genético

7.4 Módulo del Algoritmo Cuckoo Search

En esta sección de aborda específicamente el resultado esperado 8 (R 8), el cual consiste un módulo para los parámetros del algoritmo Cuckoo Search. Consta de 3 campos:

- Cantidad de nidos: Un número entero.
- Porcentaje de nidos descubierto: Un número entre 0 y 1.
- Cantidad de generaciones: La máxima cantidad de iteraciones del algoritmo.

Figura 26. Módulo algoritmo Cuckoo Search

7.5 Interfaz de Resultados

En esta sección de aborda específicamente el resultado esperado 11 (R 11R 7), el cual consiste en una interfaz para mostrar los resultados del algoritmo.

Las regiones con un rectángulo negro indican los defectos, tomando en cuenta que si una pieza se encuentra sobre uno de ellos, se considera toda la región como desperdicio y se vuelve a colocar nuevamente la pieza, es por ello que comparten el mismo color.

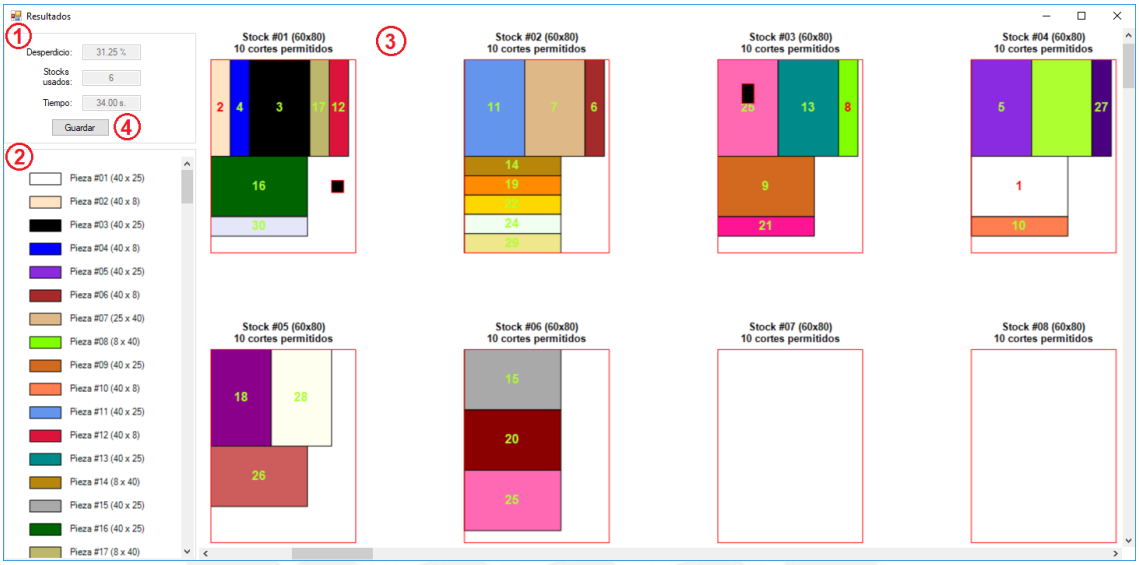


Figura 27. Interfaz de resultado

A continuación, se describen la información que contienen las secciones de la interfaz.

1. En esta sección se muestra los resultados de la solución expuesta, como el porcentaje de desperdicio, la cantidad de stocks utilizados y el tiempo
2. En esta sección se muestra una leyenda de cada una de las piezas, sus dimensiones y el color que se utilizado en los gráficos.
3. En esta sección se muestran cada uno de los stocks y las piezas que contienen, donde los stocks no utilizados se muestran sin ninguna pieza. Se muestra información de los stocks y el identificador de cada pieza.
4. Este botón sirve para exportar los resultados de la solución en un formato que pueda ser impreso y un reporte con instrucciones de cómo realizar los cortes de tipo guillotina, de tal manera que sea entendible por un operario. Por lo tanto, se generan dos tipos de archivos:

- Un archivo de imagen (.png) igual al gráfico de la interfaz (piezas y stocks).
- Un archivo de texto (.txt) indicando cómo realizar los cortes (ver Figura 28).

Instrucciones.txt

```
================================================================================================
                                    OPTIMIZACION DE CORTES EN 2D
================================================================================================
Stock #01 (60 x 80)
01. Realizar un corte guillotinado vertical paralelo a 48 unidades del borde izquierdo.
02. Realizar un corte guillotinado vertical paralelo a 40 unidades del borde izquierdo.
03. Realizar un corte guillotinado horizontal paralelo a 66 unidades del borde superior.
04. Realizar un corte guillotinado horizontal paralelo a 58 unidades del borde superior.
05. Realizar un corte guillotinado horizontal paralelo a 33 unidades del borde superior.
06. Realizar un corte guillotinado horizontal paralelo a 8 unidades del borde superior.
------------------------------------------------------------------------------------------------
Stock #02 (60 x 80)
01. Realizar un corte guillotinado vertical paralelo a 48 unidades del borde izquierdo.
02. Realizar un corte guillotinado vertical paralelo a 40 unidades del borde izquierdo.
03. Realizar un corte guillotinado horizontal paralelo a 49 unidades del borde superior.
04. Realizar un corte guillotinado horizontal paralelo a 41 unidades del borde superior.
05. Realizar un corte guillotinado horizontal paralelo a 33 unidades del borde superior.
06. Realizar un corte guillotinado horizontal paralelo a 8 unidades del borde superior.
------------------------------------------------------------------------------------------------
Stock #03 (60 x 80)
01. Realizar un corte guillotinado vertical paralelo a 50 unidades del borde izquierdo.
02. Realizar un corte guillotinado horizontal paralelo a 65 unidades del borde superior.
03. Realizar un corte guillotinado horizontal paralelo a 40 unidades del borde superior.
04. Realizar un corte guillotinado vertical paralelo a 25 unidades del borde izquierdo.
```

Figura 28. Archivo de texto con instrucciones de los cortes.

Capítulo 8. Calibración de variables y parámetros

8.1 Introducción

La siguiente sección muestra la justificación de los parámetros que se utilizarán en el algoritmo Cuckoo Search para el problema de corte de material abordado en este proyecto. Se consideran los siguientes parámetros:

- Tamaño de los nidos
- Cantidad de generaciones
- Porcentaje de descubrimiento de nidos

8.2 Tamaño de los nidos

Debido a que hay un gran componente estadístico, el tamaño de los nidos también debe ser relativamente grande. Para esta ocasión, se considera un tamaño de 150, mayor al que se utilizará en el genético. Esta cantidad se consideró en base al algoritmo original desarrollado por (Yang & Deb, 2009).

8.3 Cantidad de máxima de generaciones

La cantidad de generaciones, entendida como las iteraciones del algoritmo, depende mucho del problema a resolver. Para este proyecto, la cantidad de generaciones apropiada fue calculado de manera empírica, realizando experimentos con 100, 500, 1000 y 2000 iteraciones, donde se obtuvo que la 1000 es un número apropiado para el proyecto.

Iteraciones	Función Objetivo de la solución encontrada
100	55.0617
500	52.5068
1000	46.0740
2000	46.0740

Se observa que al incrementar las iteraciones de 1000 a 2000, la función objetivo de la solución no se incrementó. Se concluye que 1000 es un número aceptable de iteraciones para el algoritmo.

8.4 Probabilidad pa

Es otro de los componentes aleatorios del algoritmo, y es la que le introduce el carácter elitista al algoritmo Cuckoo Search. El parámetro “pa” consiste en la fracción de los peores nidos que se van a descartar. Para este proyecto, se escogerá un valor

de 0.4, el cual es un valor considerado en el algoritmo original en varias implementaciones del algoritmo (Valian, Mohanna, & Tavakoli, 2011).

Capítulo 9. Experimentación numérica

9.1 Introducción

En este capítulo se exponen los resultados de la experimentación numérica, el cual tiene como objetivo comparar el desempeño del algoritmo genético contra el desempeño del algoritmo de búsqueda tabú, tomando como métrica su valor de fitness o función objetivo.

Para llevar a cabo el objetivo de este capítulo, se ejecutaron ambos algoritmos con 40 muestras distintas. Dada el componente aleatorio de los algoritmos metaheurísticos, se decidió ejecutar por cada muestra 10 veces el algoritmo, quedándonos con el mejor resultado como representativo para cada muestra, es decir, el mínimo resultado de dichas ejecuciones. Si bien se trabajan con muestras distintas, se mantiene el mismo stock disponible y sus defectos, pues no afecta de manera relevante la construcción de la solución (se planteó asumiendo stock infinito y luego dividirlo en varios stocks) y la experimentación numérica tampoco se verá afectada.

Asimismo, al estar el resultado del fitness del algoritmo entre valores de 0 a 100, esto debido a que su cálculo es minimizar el porcentaje de desperdicio, se puede asegurar que la desviación estándar de los resultados no es alta y podrán realizarse las pruebas estadísticas.

9.2 Generación de las muestras

Para la creación de las muestras, se desarrolló un programa en el cual genera combinaciones aleatorias de los tamaños estándares de los listelos, tacos e insertos: 40x8 cm, 40x25 cm y 10x10 cm. Se realiza esta combinación para probar la eficacia de los algoritmos, considerando que es más complejo que el caso que se soliciten cerámicos del mismo tamaño.

Asimismo, se ha establecido que un tamaño para la cantidad de pedidos de 20. Se ha escogido esté número ya que se podrá comparar mejor la diferencia de los algoritmos para instancias del problema no triviales, sin dejar de ser un número demasiado grande con el objetivo de poder realizar un seguimiento manual a un caso específico.

Para la información del material disponible en stock con defectos, se utilizará solo un stock fijo para todas las muestras, pues igual se podrá observar el comportamiento del

algoritmo debido a que los pedidos de piezas siempre son distintos y facilitaría realizar un seguimiento manual del algoritmo para algunos casos.

9.3 Resultados

A continuación, se muestran los resultados obtenidos en las ejecuciones de ambos algoritmos. Se presenta el valor de la función objetivo de la mejor solución encontrada en cada muestra (cada muestra se ejecutó 10 veces).

Tabla 28. Ejecución de los algoritmos en las muestras

Muestra	Función objetivo de la solución del algoritmo Genético	Función objetivo de la solución del algoritmo Cuckoo Search
1	46.07	55.06
2	36.67	50.72
3	40.00	43.29
4	28.15	31.06
5	43.98	46.37
6	29.51	30.16
7	34.63	38.08
8	36.24	47.24
9	40.49	51.71
10	38.22	48.19
11	45.73	46.85
12	39.18	52.63
13	39.67	48.23
14	37.78	51.55
15	49.31	57.23
16	41.62	52.62
17	41.51	41.64
18	35.18	40.45
19	46.55	51.77
20	37.02	38.05
21	39.84	47.27
22	38.05	40.84
23	41.25	53.88
24	45.50	53.97
25	39.46	43.26
26	40.60	49.41
27	35.02	44.60
28	48.41	52.29
29	45.18	51.86
30	36.13	47.49
31	47.18	54.70
32	36.87	41.59
33	45.32	50.05
34	40.10	40.21

35	45.22	51.33
36	45.80	48.61
37	49.99	53.09
38	36.22	46.70
39	43.04	53.31
40	46.39	59.30

9.4 Prueba Kolmogorov-Smirnov

Una de las condiciones para realizar la Prueba Z (ver sección 9.6) es que los valores de la muestra sigan una distribución normal, por lo que es necesario demostrar que los resultados de la ejecución de los algoritmos presentan esta distribución. Para ello, se realiza la prueba Kolmogorov-Smirnov.

9.4.1 Prueba para el algoritmo genético

Se plantean las siguientes hipótesis:

- H0: Los valores de la muestra del algoritmo genético presentan una distribución normal.
- H1: Los valores de la muestra del algoritmo genético no presentan una distribución normal.

Se calculan los siguientes estadísticos:

Tabla 29. Estadísticos para la Prueba de Kolmogorov-Smirnov del algoritmo genético

Media	40.827	**Mínimo**	28.15
Desviación Estándar	5.13	**Máximo**	49.99
Varianza	26.32675487	**Datos**	40
Máxima Diferencia	0.102		
Significancia de la prueba	0.05		
Valor Crítico	0.215		

donde el valor crítico se calcula $\frac{1.36}{\sqrt{Cant.\ Datos}}$ para cantidad de datos mayor a 40.

Como la máxima diferencia es menor al valor crítico (0.102 < 0.215), se acepta la hipótesis nula (H0), es decir, los valores siguen una distribución normal.

9.4.2 Prueba para el algoritmo Cuckoo Search

Se plantean las siguientes hipótesis:

- H0: Los valores de la muestra del algoritmo Cuckoo Search presentan una distribución normal.

- H1: Los valores de la muestra del algoritmo Cuckoo Search no presentan una distribución normal.

Se calculan los siguientes estadísticos:

Tabla 30. Estadísticos para la Prueba de Kolmogorov-Smirnov del algoritmo Cukoo Search

Media	47.6665	**Mínimo**	30.16
Desviación Estándar	6.57	**Máximo**	59.3
Varianza	43.22	**Datos**	40
Máxima Diferencia	0.097		
Significancia de la prueba	0.05		
Valor Crítico	0.215		

donde el valor crítico se calcula $\frac{1.36}{\sqrt{Cant.\ Datos}}$ para cantidad de datos mayor a 40.

Como la máxima diferencia es menor al valor crítico (0.097 < 0.215), se acepta la hipótesis nula (H0), es decir, los valores siguen una distribución normal.

9.5 Prueba F de Fisher

Otra condición para realizar la Prueba Z (ver sección 9.6) es que las varianzas de ambos algoritmos sean significativamente homogéneas, por lo que es necesario demostrar este hecho. Para ello, se realiza la prueba F de Fisher.

9.5.1 Prueba para los algoritmos

Se plantean las siguientes hipótesis:

- H0: Las varianzas de los algoritmos son significativamente homogéneas
- H1: Las varianzas de los algoritmos son significativamente diferentes.

Se calculan los siguientes estadísticos:

Tabla 31. Estadísticos para la Prueba F de Fisher de los algoritmos

	Genético	**Cuckoo S.**
Media	40.827	47.6665
Varianza	26.3267549	43.22
Observaciones	40	40
Grados de libertad	39	39
F	0.609100981	
P(F<=f) una cola (derecha)	0.937045541	
Valor crítico para F (una cola)	1.704465067	

Como F = 0.6091 ∈ [0.25, 1.70] (región de aceptación) **se acepta** la hipótesis nula, por lo que se concluye que las varianzas **son significativamente homogéneas.**

9.6 Prueba Z

Como objetivo de esta experimentación numérica, se busca comparar la media de los resultados del algoritmo genético con la media de los resultados del algoritmo Cuckoo Search. En principio, la prueba t-Student es adecuada para la verificación de las medias entre 2 tratamientos, sin embargo, al ser el número de datos mayor a 30 (como en este proyecto), esta prueba se comporta como la distribución Z, por lo que la Prueba Z es la más apropiada para este proyecto. Dicha prueba presenta las siguientes condiciones:

- Verificar que los datos presenten una distribución normal. Esto fue demostrado en la sección 9.4 a través de la prueba de Kolmogorov-Smirnov.
- Verificar que los datos posean varianzas homogéneas. Esto fue demostrado en la sección 9.5 a través de la prueba F de Fisher.

Para realizar esta prueba, se deben seguir dos pasos. El primero consiste en determinar si la media de los resultados obtenidos con el algoritmo genético es igual o significativamente diferente a la media de los resultados obtenidos con el algoritmo Cuckoo Search (Prueba de dos colas).

Si de concluye que las medias son distintas, se continúa con el segundo paso, en cual consiste en determinar si la media de un algoritmo es menor a la media del otro algoritmo (Prueba de una cola).

Tabla 32. Estadísticos para la Prueba Z de los algoritmos

	Genético	**Cuckoo Search**
Media	40.8277886	47.6663899
Varianza (conocida)	26.3267549	43.2223156
Observaciones	40	40
Diferencia hipotética de las medias	0	
z	-5.1862281	
P(Z<=z) una cola	1.073E-07	
Valor crítico de z (una cola)	1.64485363	
Valor crítico de z (dos colas)	2.146E-07	
Valor crítico de z (dos colas)	1.95996398	

9.6.1 Prueba de dos colas

Se plantean las siguientes hipótesis:

- H0: Las varianzas de los algoritmos son significativamente homogéneas
- H1: Las varianzas de los algoritmos son significativamente diferentes.

Para aceptar como cierta la hipótesis nula, el valor de z debe estar entre -1.95996 y +1.95996. Como vimos en la tabla Tabla 32, el valor de z es -5.1862281, por lo que caería en una región crítica haciendo que se rechace la hipótesis nula, es decir, se concluye que la media de los algoritmos genético y Cuckoo Search son diferentes.

9.6.2 Prueba de una cola

Una vez demostrado que las medias son significativamente diferentes, se puede proceder a averiguar si la media de los resultados del algoritmo Cuckoo Search es menor que la media de los resultados del algoritmo genético o viceversa.

Se plantean las siguientes hipótesis:

- H0: La media del algoritmo Genético es mayor que la media del algoritmo Cuckoo Search.
- H1: La media del algoritmo Genético es menor que la media del algoritmo Cuckoo Search.

Para aceptar como cierta la hipótesis nula, el valor de z debe ser mayor a -1.64485. Como vimos en la tabla Tabla 32, el valor de z es -5.1862281, el cual es menor y caería en una región crítica haciendo que se rechace la hipótesis nula, es decir, se concluye que la media del algoritmo Genético es menor a la media del algoritmo Cuckoo Search.

Capítulo 10. Conclusiones y trabajos futuros

10.1 Conclusiones

Si bien es posible encontrar en la literatura algoritmos exactos que permitan resolver el problema de corte de material, la mayoría de estos no son viables de implementar para instancias grandes del problema. Según (Kallrath, Rebennack, Kallrath, & Kusche, 2014), quienes hicieron una experimentación en la vida real en la industria del papel, para una cantidad de piezas de 25 demorarían mucho tiempo, lo que hace imposible de usarlos.

Asimismo, a partir de los resultados de la experimentación numérica, se ha podido verificar que la media de la función objetivo de las soluciones generadas por el algoritmo genético es significativamente menor a la media de la función objetivo de las soluciones generadas por el del algoritmo Cuckoo Search. Se puede concluir que el algoritmo genético tiene mejor desempeño que el algoritmo Cuckoo Search para el conjunto de datos que se usó, lo que se puede explicar teniendo en cuenta que este último algoritmo tiene como fase de mejora un movimiento aleatorio (vuelo de Levy) que en la mayoría de los casos no mejora la solución actual, a diferencia de la fase de mejora del algoritmo genético, en el que las operaciones de cruce y mutación sí logran mejorarlas.

Finalmente, los algoritmos metaheurísticos genético y Cuckoo Search diseñados en el presente proyecto de fin de carrera para resolver el problema del corte, pueden ser fácilmente adaptables a otras variantes del problema del corte (modificando adecuadamente la estructura de la solución) e incluso pueden servir como base para resolver otro problema de optimización.

10.2 Trabajos futuros

Se proponen las siguientes investigaciones futuras a partir de este proyecto de fin de carrera:

- El cálculo de la población inicial para ambos algoritmos puede no ser aleatorio sino a través de otro algoritmo metaheurístico como el GRASP, colocando primero las piezas más grandes y las siguientes a escoger serán las que vayan ocupando menor tamaño acumulado, lo cual podría reducir el tiempo que se demora en llegar al óptimo.

- Modificar el algoritmo Cuckoo Search, incluyéndole una fase de mejora que incluso puede ser similar a la operación de mutación del algoritmo genético, pues las operaciones que realiza no obtienen una solución óptima.
- Incluir la fase de evitar los defectos como parte de la construcción de la solución. Si bien este objetivo se alcanzó en el cálculo del fitness de cada algoritmo, incluirlo en la construcción, a pesar de que podría tomar más tiempo debido a las posibles soluciones no válidas, podría llegar al óptimo en menor cantidad de generaciones para ambos algoritmos.
- Si bien el sistema fue diseñado con características propias de la industria de cerámicos como corte guillotinado, stock múltiple y defectos, se puede trabajar con otros materiales de forma rectangular como planchas de metal, lonas de tejido, madera, incluyendo restricciones como la cantidad de cortes, forma regular de la solución, teniendo como base el presente algoritmo.

Referencias

Adams, P. J. (1961). Geology and ceramics: a brief review of the nature, geological occurrence, processing, and principal industrial application of the rocks and minerals used in British ceramic manufacture. HM Stationery Off.

Alvarez-Valdes, R., F. Parreño, & Tamarit, J. M. (2005). A GRASP Algorithm for Constrained Two-Dimensional Non-Guillotine Cutting Problems. The Journal of the Operational Research Society, (4), 414.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. University computing, 15(2), 56–69.

CCL (2018). La cámara, la revista de la CCL. Cámara de Comercio de Lima. Recuperado el 10 de enero de 2018, a partir de https://www.camaralima.org.pe/repositorioaps/0/0/par/edicion808/edicion_808.pdf

Du, K.-L., & Swamy, M. N. S. (2016). Search and Optimization by Metaheuristics. Springer.

Dyckhoff, H. (1990). A typology of cutting and packing problems. European Journal of Operational Research, 44(2), 145–159.

Galán, E., & Aparicio, P. (2006). Materias primas para la industria cerámica. Seminarios de la Sociedad Española de Mineralogía, 2, 31–49.

Galarza, M., & Paulo, M. (2011). Desperdicio de materiales en obras de construccion civil: metodos de medicion y control.

Gen, M., & Cheng, R. (2000). Genetic algorithms and engineering optimization (Vol. 7). John Wiley & Sons.

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. Operations research, 9(6), 849–859.

Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—Part II. Operations research, 11(6), 863–888.

INEI (2015). Exportación de baldosas de cerámica barnizada o esmaltada creció en 762%. Recuperado el 10 de abril de 2017, a partir de https://www.inei.gob.pe/prensa/noticias/exportacion-de-baldosas-de-ceramica-barnizada-o-esmaltada-crecio-en-762-8688/

INEI (2017). Producto Bruto Interno según Actividad Económica. Recuperado el 5 de mayo de 2017, a partir de https://www.inei.gob.pe/estadisticas/indice-tematico/economia/

Kallrath, J., Rebennack, S., Kallrath, J., & Kusche, R. (2014). Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. European Journal of Operational Research, 238(1), 374–389.

Kamaruzaman, A. F., Zain, A. M., Yusuf, S. M., & Udin, A. (2013). Levy flight algorithm for optimization problems-a literature review. En Applied Mechanics and Materials (Vol. 421, pp. 496–501). Trans Tech Publ.

Kantu. (2016). Productos decorados de cerámica. Recuperado el 6 de mayo de 2017, a partir de http://www.ceramicaskantu.com/productos_ceramica.html

Ledolter, J., & Robert, V. (2010). Applied Statistics for Engineers. N/A Cover Type N/A.

Lodi, A., Martello, S., & Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing, 11(4), 345–357.

Martı, R. (2003). Procedimientos metaheurısticos en optimización combinatoria. Matemátiques, Universidad de Valencia, 1(1), 3–62.

Microsoft. (2017a). C# | Microsoft Docs. Recuperado a partir de https://docs.microsoft.com/en-us/dotnet/csharp/csharp

Microsoft. (2017b). Visual Studio IDE. Recuperado el 6 de mayo de 2017, a partir de https://www.visualstudio.com/vs/

Porras, A. M. L. (2000). Diseño estadístico de experimentos, análisis de la varianza y temas relacionados: tratamiento informático mediante SPSS.

Pravesjit, S., & Kantawong, K. (2017). An improvement of genetic algorithm for optimization problem. En Digital Arts, Media and Technology (ICDAMT), International Conference on (pp. 226–229). IEEE.

RAE. (2017). Cortar. Recuperado a partir de http://dle.rae.es/?id=B1wW3tP

Real Cut 2D. (2012). Optimal Programs. Recuperado el 17 de abril de 2017, a partir de http://www.optimalprograms.com/realcut2d.htm

Resende, M. G., & Ribeiro, C. C. (2016). Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer.

Sánchez, I. (2016). Aplicación de Algoritmos Genéticos para la optimización del corte de material.

Schwaber, K., & Sutherland, J. (2013). La guía de scrum: La guía definitiva de scrum, las reglas del juego.

SNI (2015). Industria de fabricación de baldosas cerámicas creció 4,9% en el 2014. Recuperado el 10 de abril de 2017, a partir de http://www.sni.org.pe/industria-de-fabricacion-de-baldosas-ceramicas-crecio-49-en-el-2014/

Talbi, E.-G. (2009). Metaheuristics: from design to implementation (Vol. 74). John Wiley & Sons.

UDEP. (2015). Análisis del Sector Construcción. Recuperado el 6 de abril de 2017, a partir de http://www.biblioteca.udep.edu.pe/bibvirudep/tesis/pdf/1_97_204_59_903.pdf

Valian, E., Mohanna, S., & Tavakoli, S. (2011). Improved cuckoo search algorithm for global optimization. International Journal of Communications and Information Technology, 1(1), 31–44.

Vasko, F. J. (1989). A computational improvement to Wang's two-dimensional cutting stock algorithm. Computers & industrial engineering, 16(1), 109–115.

Wang, P. Y. (1983). Two algorithms for constrained two-dimensional cutting stock problems. Operations Research, 31(3), 573–586.

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. European journal of operational research, 183(3), 1109–1130.

Yang, X.-S. (2010). Firefly algorithm, Levy flights and global optimization. Research and development in intelligent systems XXVI, 209–218.

Yang, X.-S., & Deb, S. (2009). Cuckoo search via Lévy flights. En Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on (pp. 210–214). IEEE.