UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

APLICACIÓN DE METAHEURÍSTICA *TABU SEARCH* PARA EL PROBLEMA DE CARGA DE CAMIÓN

POR

Mauricio Ignacio Ulloa Leyton

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Industrial

Profesora Guía

Rosa Medina Durán

Julio, 2022

Concepción, Chile

© 2022 Mauricio Ignacio Ulloa Leyton

© 2022 Mauricio Ignacio Ulloa Leyton

Se autoriza la reproducción total o parcial, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica del documento.

Sumario

El problema de carga de camión consiste en ubicar n cajas dentro de un camión, las cuales no pueden ser rotadas y deben ubicarse de forma paralela a las paredes del camión respetando la regla LIFO ("*Last in First out*"), es decir, ubicando las cajas de un mismo clientes juntas y, respetando el orden de entrega. También, se busca proteger la integridad de las cajas añadiendo restricciones de fragilidad al problema. El problema de carga de camión es una parte fundamental de la eficiencia de la operación de las cadenas de suministros. Una distribución deficiente, puede resultar en costos innecesarios y en experiencias negativas para los clientes, por esto, cada vez se hace más importante encontrar más y mejores soluciones para problemas de este tipo. En el presente trabajo se realiza un estudio de 2 algoritmos que resuelven el problema de carga de camión, diseñados a partir de la implementación de tres heurísticas constructivas que dan una solución rápida y factible para el problema, y dos algoritmos metaheurísticos que mejoran las soluciones entregadas por dichas heurísticas constructivas.

Con el objetivo de evaluar el desempeño de los algoritmos, se utilizan los datos y rutas usados en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Se consideró un camión con un ancho de 150 cm, un alto de 215 cm, largo infinito y 21 instancias diferentes. Los resultados muestran que algunos algoritmos y heurísticas superan las soluciones de investigaciones anteriores. En esta investigación se comparan tanto las heurísticas constructivas como las metaheurísticas, obteniendo mejores resultados con la heurística constructiva *Wall Building* modificado y con la metaheurística *Tabu Search* desarrollada en esta investigación.

// Summary

The truck loading problem consists of placing n boxes inside a truck, which cannot be rotated and must be placed parallel to the truck walls, respecting the LIFO law ("Last in First out"), that is, placing the boxes of the same customers together and in order of delivery. Also, it seeks to protect the integrity of the boxes by adding fragility restrictions to the problem. The truck loading problem is a critical part of efficient supply chain operations. Poor distribution can result in unnecessary costs and negative customer experiences, for this reason, it is becoming more and more important to find more and better solutions for problems of this type. In the present work a study of 2 algorithms that solve the truck loading problem is carried out, designed from the implementation of three constructive heuristics that give a quick and feasible solution to the problem, and two metaheuristic algorithms that improve the solutions delivered by such constructive heuristics.

In order to evaluate the performance of the algorithms, the data and routes used in Anabalón et al. (2021) [1] were solved. A truck with a width of 150 cm and a height of 215 cm and 21 different instances were considered. The results showed that some of the algorithms and heuristics surpassed the solutions of previous research. In this research, both constructive and metaheuristic algorithms are compared, obtaining better results with the modified Wall Building constructive heuristic and with the Tabu Search metaheuristic developed in this research.

Contenido

Sumario 2
Summary 3
Índice de Figuras 6
Índice de Tablas 6
Índice de Algoritmos 7
Anexos 7
1. Introducción y objetivos 1
1.1 Introducción 1
1.2 Objetivos 2
1.2.1 Objetivo general 2
1.2.2 Objetivos Específicos 2
2. Descripción del Problema 4
2.1 Definición del problema 4
2.2 Problemas Relacionados 5
3. Revisión de literatura 6
3.1 Descripción del capítulo 6
3.2 Programación Lineal Entera 6
3.3 Algoritmos Heurísticos y metaheurísticos 8
3.3.2 Algoritmos Voraces 10
3.4 Trabajos relacionados 14
3.4.1 Memoria de Nicolás Anabalón 14
3.4.1.4 Selección de Soluciones 17
3.4.2 Touching Perimeter Algorithm 18
3.5 Metaheurística Tabu Search 20
4. Metodología 22
4.1 Heurísticas Constructivas 23

4.1.1 *Wall Building* ... 23
4.1.2 *Wall Building* modificado ... 23
4.1.3 *Touching Perimeter Algorithm* ... 27
4.2 Adaptación de Harmony Search ... 30
4.3 Adaptación de Tabú Search ... 30
4.3.1 Diseño del algoritmo ... 30
4.3.2 Funciones *movf* y *movv* ... 31
4.3.3 Explicación pseudocódigo ... 31
4.3.4 Proceso de intensificación ... 35
5. Resultados computacionales ... 37
5.1 Selección heurística constructiva ... 37
5.2 Calibración de parámetros para metaheurística Tabu Search ... 39
5.3 Comparación de los algoritmos ... 44
5.4 Discusión de los resultados ... 45
6. Conclusiones ... 48
7. Glosario ... 50
8. Referencias ... 51
9. Anexos ... 55

Índice de Figuras

Figura 1: Solución instancia 5 Wall Building 14
Figura 2: Solución instancia 5 Harmony Search 14
Figura 3: Pseudocódigo Touching Perimeter Algorithm Lodi et al. (1999) [36] 19
Figura 4: Pseudocódigo Tabu Search, Stepanenko, 2008 [39] 20
Figura 5: Ejes del problema (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) 22
Figura 6: Wall Building modificado instancia 3 38
Figura 7: Wall Building instancia 3 38
Figura 8: Touching Perimeter instancia 3 38
Figura 9: Largo vs Tiempo 43
Figura 10: Volumen vs Tiempo 43
Figura 11: Penalización vs Tiempo 44
Figura 12: Instancia 2 heurística Wall Building modificado 46
Figura 13: Instancia 2 heurística Wall Building modificado y metaheurística Tabu Search 46

Índice de Tablas

Tabla 1: Heurísticas constructivas primeras 9 instancias 38
Tabla 2: Combinación de parámetros y resultados 40
Tabla 3: parámetro $cajasmovidas$ y resultados 41
Tabla 4: parámetro $Iteracionesintensificación$ y resultados 41
Tabla 5: parámetro $Procesointensificación$ y resultados 41
Tabla 6: Resultados combinación $vecindarios\boldsymbol{tabu}$ = 5, $tenencia\boldsymbol{tabu}$ = 7, $Iteracionesintensificación$ = 10 y $procesointensificación$ = 5 42
Tabla 7: Resultados 21 instancias heurística constructiva y metaheurísticas 600 segundos de cómputo 45

Índice de Algoritmos

Algoritmo 1: Wall Building modificado 26
Algoritmo 2: Touching Perimeter Algorithm........ 29
Algoritmo 3: metaheurística Tabu Search........ 34
Algoritmo 4: Proceso de intensificación Tabu Search 36

Anexos

Anexo 1: Experimentos vecindarios 55
Anexo 2: Experimento 4 vecindarios y 6 tenencia........ 55
Anexo 3: Experimento 4 vecindarios y 7 tenencia........ 55
Anexo 4: Experimento 5 vecindarios y 5 tenencia........ 56
Anexo 5: Experimento 5 vecindarios y 6 tenencia........ 56
Anexo 6: Experimento 5 vecindarios y 7 tenencia........ 56
Anexo 7: Experimento 6 vecindarios y 5 tenencia........ 57
Anexo 8: Experimento 6 vecindarios y 6 tenencia........ 57
Anexo 9: Experimento 6 vecindarios y 7 tenencia........ 57
Anexo 10: cajas_movidas = 6 58
Anexo 11: cajas_movidas = 4 58
Anexo 12: cajas_movidas = 8 58
Anexo 13: cajas_movidas = 10 59
Anexo 14: cajas_movidas = 12 59
Anexo 15: Iteraciones_intensificación = 5........ 59
Anexo 16: Iteraciones_intensificación = 10........ 60
Anexo 17: Iteraciones_intensificación = 15 60
Anexo 18: Proceso_intensificación = 5........ 60
Anexo 19: Proceso_intensificación = 10........ 60
Anexo 20: Proceso_intensificación = 15........ 61

A mis padres, por su esfuerzo, amor y dedicación

A mi hermana, por sus valiosas enseñanzas

A la Jo, por su compañía e importante apoyo en esta etapa

A mis amigos, por su incondicionalidad, constante apoyo y alegría

A mis profesores, por darme herramientas y experiencias para enfrentar cada desafío

A todos, gracias totales

1. Introducción y objetivos

1.1 Introducción

El problema de carga de camión, junto con el problema de ruteo son muy importantes dentro de la logística, está presente en casi todas las áreas productivas de la industria, esto debido a la necesidad de hacer llegar los productos a los clientes, cada vez es más necesario y más aún hoy que cobra especial importancia en el contexto sanitario en que nos encontramos, haciendo que sea necesario implementar más y mejores soluciones a este tipo de problemáticas. Una vez resuelto el problema de ruteo, se debe buscar la mejor forma de ubicar las cajas dentro de un camión, de manera que se evite el daño sobre las cajas y se simplifique la entrega a los múltiples clientes.
Dentro de la literatura, existen muchas formas de abordar el problema, por lo que hay más de una forma de modelarlo, agregando diferentes restricciones para cada enfoque. Son los casos del *3-D Packing* (Costa & Captivo, 2014) , el *Truck Loading Problem* (Respen & Zufferey, 2016) y el *Container Loading Problem* (Yamashita & Morabito, 2015). Para solucionar dichos problemas y otros similares, se suelen usar algoritmos genéticos y métodos exactos (Respen & Zufferey, 2016)**.** Para resolver esta problemática, en esta investigación se pretende usar *Tabu search (TS).*

Tabu search es una metaheurística de solución única usada para resolver problemas de optimización combinatoria que se diferencia de otros por el uso de memoria adaptativa y de estrategias especiales de resolución de problemas. El término *Tabu Search* ha sido introducido por Fred Glover en 1986 al mismo tiempo que acuñaba el término metaheurística. Esta ha tenido un destacado éxito para resolver problemas duros de optimización, especialmente con problemas del mundo real (Batista & Glover, 2006).

Tabu Search comienza con una solución inicial, y en cada iteración se explora al vecindario de la solución actual con el objetivo de buscar una posible mejor solución. El “vecindario” corresponde a las soluciones alcanzables desde la solución s haciendo una modificación particular en esta, el proceso de buscar mejores soluciones en el vecindario se realiza hasta un cierto criterio de término definido por el usuario. *Tabu Search* es un método de búsqueda local, cuya característica principal es que el paso anterior está prohibido para un cierto número de iteraciones (Respen & Zufferey, 2016)**,** esto se hace con él objetivo de escapar de mínimos locales, para esto se crea una lista tabú que almacenará todos los elementos o movimientos tabú que no podrán realizarse por una cierta

cantidad de iteraciones definidas en tenencia tabú, por ello la lista tabú tendrá un tamaño igual al parámetro tenencia tabú.

Debido a que cada problema es modelado y evaluado de forma distinta en cada investigación, se hace interesante comparar dentro del mismo trabajo y con las mismas condiciones la metaheurística de búsqueda local con la propuesta realizada en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) evaluando los resultados de ambos trabajos, verificando la utilidad y aporte de dichas metodologías para el problema de carga de cajas.

También, en este trabajo se añaden nuevas heurísticas constructivas que permiten tener una disposición de cajas inicial distinta a la presentada en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021).

1.2 Objetivos

1.2.1 Objetivo general

Proponer una nueva metaheurística y heurísticas constructivas para el problema de carga de camión y comparar sus resultados con soluciones propuestas para el mismo modelo.

1.2.2 Objetivos Específicos

- Realizar una revisión de literatura donde se recopilen métodos de resolución del problema de empaquetamiento de cajas en 2D y 3D para conocer estrategias de resolución del problema y como estas pueden aportar características a la nueva solución.
- Estudiar trabajos o estudios previos relacionados a la carga de cajas en camión, como lo visto en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) para realizar comparaciones válidas.
- Diseño e implementación de nuevas heurísticas constructivas de resolución para el problema de empaquetamiento de cajas.
- Diseñar adaptación de *Tabú Search* para el problema de carga de camión.
- Implementar adaptación de *Tabú Search* en lenguaje de programación Python.
- Comparar métodos de resolución del problema con los resultados en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) usando las mismas instancias.

La hipótesis de esta investigación es: "se pueden encontrar soluciones válidas para el *Truck Loading Problem* mediante la metaheurística *Tabu Search*"

En particular en el presente estudio se busca analizar el uso de la metaheurística *Tabu Search* para el *Truck Loading Problem,* comparando sus resultados por los obtenidos por Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), de esta manera, puede ser posible validar el uso de la metaheurística *Harmony Search* utilizada en dicho estudio. También, se busca encontrar mejores soluciones probando diferentes configuraciones para el algoritmo, tres diferentes soluciones iniciales. Finalmente, después de seleccionar la mejor heurística constructiva, se realiza una comparación entre las dos metaheurísticas de mejora.

El presente documento se estructura en 6 capítulos, sin considerar glosario, referencias ni anexos. En el capítulo 2 se presenta una descripción del problema abordado, junto con descripciones de problemas similares y como se han solucionado. En el capítulo 3, se muestran los distintos métodos de resolución para este problema y para problemas similares. También, se referencian los trabajos anteriores que fueron utilizados en esta investigación. En el capítulo 4, se describe la implementación de las heurísticas constructivas y metaheurísticas propuestas. En el capítulo 5, se muestran los resultados obtenidos durante las pruebas computacionales. Y, finalmente, en el capítulo 6 se presentan las conclusiones del trabajo.

2. Descripción del Problema

2.1 Definición del problema

La planificación de las cargas es una de las actividades clave en logística operacional, un rendimiento ineficiente en esta área puede resultar en costos innecesarios para la empresa y en un servicio al cliente deficiente (Bortfeldt & Wäscher, Constraints in container loading – A state-of-the-art review, 2013).

El problema de carga de camión o *Truck Loading Problem (TLP)*, es una subclase de *Container loading problem (CLP)*, que a su vez es un problema del tipo *cutting and packing problem (C&P)* (Wäscher, HauBner , & Schumann, 2007).

Los problemas tipo *C&P* son aquellos que tienen una lógica estructural similar y en la literatura, se encuentran nombres:

- Problemas de *cutting stock* y *trim loss,*
- Problemas de *bin packing, dual bin packing, strip packing, vector packing,* y *knapsack packing,*
- Problemas de *vehicule loading, pallet loading*, *container loading y car loading.*
- Problemas de *assortment, depletion, design, dividing, layout, nesting,* y *partitioning.*
- Incluso problemas de *capital budgeting, change making, line balancing, memory allocation,* y *multi-processor scheduling.* (Dyckhoff, 1990)

Los problemas tipo *Container loading* pueden ser vistos como problemas de asignación geométrica, en donde pequeños objetos 3D (denominados *cargo*) deben ser asignados a grandes objetos rectangulares (denominados *containers*) de forma que cierta función objetivo es optimizada y sujeta a dos tipos de restricciones básicas; todos los objetos pequeños deben quedar dentro del *container* y no pueden estar superpuestos uno sobre el otro (Bortfeldt & Wäscher, Constraints in container loading – A state-of-the-art review, 2013).

Se entiende por "un objeto grande" que podría ser un contenedor propiamente tal, sin embargo, también puede ser un camión o un pallet que deba cargarse hasta cierta altura, se define *TLP* como

un tipo de *CLP* (Bortfeldt & Wäscher, Constraints in container loading – A state-of-the-art review, 2013).

2.2 Problemas Relacionados

En la literatura especializada, es posible encontrar diferentes problemas relacionados al *Truck Loading Problem* (Wäscher, HauBner , & Schumann, 2007) *y Container Loading Problem.* Algunos ejemplos son:

- Problema de la mochila o *Knapsack Loading,* consiste en el modelo de un problema similar a llenar una mochila con un peso determinado que no puede ser sobrepasado, esta debe llenarse con la cantidad total o parte de los objetos a incluir, cada uno con valores y pesos específicos. Fue desarrollado en primer lugar por Lorie and Savage (Lorie & Savage, 1955), donde se buscaba incluir las mejores opciones de inversión para maximizar los beneficios dado un presupuesto previamente definido.
- Problema de embalaje de contenedores tridimensionales o *3D-bin packing problem,* este problema es definido como un grupo de *n* cajas con forma rectangular, cada una con largo, ancho y altura definidos que deben ser ubicadas en un contenedor, con cierto largo, ancho y alto. Las cajas no pueden ser rotadas, además, deben estar contenidas en el contenedor (Martello, Pisinger, & Vigo, The Three-Dimensional Bin Packing Problem, 1997) muy parecido al problema de esta investigación.
- *3D- Strip packing problem,* en este tipo de problema, todos los ítems deben ser cargados dentro de un solo contenedor de largo variable, de forma que el uso del largo del contenedor sea minimizado, en Bortfeldt y Mack (Bortfeldt & Mack, 2007) una heurística proveniente de un *CLP* es aplicada al problema.

3. Revisión de literatura

3.1 Descripción del capítulo

La literatura relacionada al *TLP* puede ser clasificada según los métodos de resolución, en métodos exactos, que incluyen programación lineal entera. En algoritmos heurísticos y metaheurísticos, que incluyen algoritmos voraces, genéticos, *Tabu Search,* entre otros. Dado que, no hay muchas investigaciones que se centren en el *TLP,* se investigaron mayormente los métodos de resolución de los problemas relacionados.

3.2 Programación Lineal Entera

En el año 1995 se publicó una investigación de Chen et al. (Chen, Lee, & Shen, 1995), en esta se desarrolló un modelo analítico para enfrentar de forma matemática el problema. El *CLP* se resolvió con un modelo entero mixto de programación. En el modelo se considera el uso de múltiples contenedores, tamaños de las cajas, su orientación y la superposición de las cajas. Esta investigación se validó numéricamente y fue de gran importancia, pues dio paso a más trabajos de esta índole, donde se podía agregar más restricciones, o bien desarrollar soluciones más eficientes para problemas de mayor escala.

En el año 1997 fue desarrollada una solución exacta para el *2D Bin Packing Problem* por Martello y Vigo (Martello & Vigo, Exact Solution of the Two-Dimensional Finite Bin Packing Problem, 1997). A partir de la solución se determinó el peor escenario, lo que permitió observar nuevas soluciones; que se mejoran mediante un algoritmo *Branch-and-bound* para acercarse al óptimo.

Gracias al trabajo realizado por Martello y Vigo (Martello & Vigo, Exact Solution of the Two-Dimensional Finite Bin Packing Problem, 1997), en el mismo año fue posible realizar la primera investigación en torno a soluciones exactas para el problema *3D Bin Packing.* Martello, Pisinger y Vigo (Martello, Pisinger, & Vigo, The Three-Dimensional Bin Packing Problem, 1997). Al adaptar la solución de 2 dimensiones a 3 dimensiones presentaron un algoritmo exacto para llenar un

contenedor, generando nuevas soluciones iniciales, trabajándolas con un algoritmo *Branch-and-bound.*

Otra investigación de algoritmo exacto es la elaborada por Fekete et al. (Fekete, Schepers, & van der Veen, An exact Algorithm for Higher-Dimensional Orthogonal Packing, 2007) el año 2007. En este trabajo se desarrolló un algoritmo de árbol de búsqueda de dos niveles para resolver problemas de empaquetamiento ortogonal u *Orthogonal packing problem (OPP).*

En el año 2012, Junqueira et al. (Junqueira, Morabito, & Sato Yamashita, 2012), buscando una solución exacta, diseñaron 3 modelos de programación lineal entera mixta para el problema de carga de contenedor, que sirve para contenedores, camiones, vagones de ferrocarril o pallets. Tomando en cuenta la estabilidad vertical y horizontal de la carga, junto con la resistencia de carga del contenedor.

Estos modelos pueden aplicarse para las variantes *3D-packing* y *bin packing (BPP).* Luego de la experimentación, se mostró que los modelos eran consistentes al mismo tiempo que representaban bien la situación real. Sin embargo, solo eran capaces de resolver en un tiempo aceptable problemas de tamaño moderado, dando paso a que se hicieran más investigaciones al respecto.

En el año 2021, do Nascimiento et al. (do Nascimiento, de Queiroz, & Junqueira, 2021) desarrollaron una aproximación exacta que propone solucionar el modelo de programación lineal y los modelos de restricción iterativamente para el problema de carga de contenedor. Se busca maximizar el volumen de la carga, para esto se propone una relajación del problema basada en empaquetado en planos.

En síntesis, la solución consta de 3 pasos; en primer lugar, se selecciona con el modelo un subconjunto U que maximice el volumen de carga y que solo respete el volumen del contenedor. El subconjunto U representará la cota superior del problema. Sin embargo, puede que, U no sea una solución factible para el problema, dado que podría no respetar la forma geométrica del contenedor. En este caso, se elige otro subconjunto.

En segundo lugar, se resuelve otro modelo de programación lineal, donde la restricción de no superponer ítems es relajada. El objetivo es verificar más rápidamente si U no es factible como solución. Se evalúa a la solución U, sin evaluarla directamente, ahorrando tiempo y así actualizando la cota superior eficientemente. Si se encuentra que, U no es factible, se realiza un corte para evitar elegir el mismo subconjunto nuevamente, regresando al primer paso.

Por otro lado, si se determina que, U es "parcialmente factible", se resuelve un tercer modelo de programación con restricciones para verificar que U sea factible. Así se encuentra una solución óptima. Si se descubre que no es factible se regresa al primer paso. (do Nascimiento, de Queiroz, & Junqueira, 2021)

Este modelo es comparado con el desarrollado por Junqueira et al. En 2012 (Junqueira, Morabito, & Sato Yamashita, 2012) testeando instancias de la literatura. Sin embargo, ese modelo podría requerir mucha memoria y esfuerzo adicional cuando se formulan restricciones complejas, por lo que aún no es validado completamente. Aun así, quedo demostrado que esta versión funciona mejor que las anteriores.

En el año 2020 Nova et al. (Novas, Ramello, & Rodríquez, 2020) trabajaron el problema de carga de camión para la más grande embotelladora de bebidas no alcohólicas en Argentina. Para resolverlo, se pretendió automatizar el proceso de carga a través de la aplicación de un modelo de programación disyuntiva generalizada. El modelo toma en cuenta 2 tipos de camiones, con 3 y 5 ejes y para cada tipo de camión se aplica una formulación distinta, en la primera, el balance de cargas es ignorado, mientras que, en el segundo se agrega dichas restricciones.

3.3 Algoritmos Heurísticos y Metaheurísticos

En la investigación de operaciones, se entiende por algoritmo heurístico al método donde se aplica un procedimiento lógico para encontrar soluciones para un problema de manera eficiente, tomando en cuenta el conocimiento que se tenga de él. De éste, se espera encontrar soluciones de alta calidad en un tiempo razonable, aunque usualmente sólo puede llegarse a aproximaciones del óptimo (Morillo, Moreno, & Díaz, 2014).

3.3.1 Algoritmos Metaheurísticos

El término *meta* significa "más allá" o "a un nivel superior". Sin embargo, la interpretación depende siempre de las concepciones actuales de lo que se considera una forma inteligente o que "vaya más

allá". Las metaheurísticas son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos generales con un alto rendimiento (Santana, y otros, 2004).

Antes de que métodos exactos fueran presentados en el año 1989, se presenta una heurística de aproximación para el *Container Loading Problem*, esta se caracteriza por llenar el espacio de carga capa por capa. Luego, es comparada con valores publicados por la administración general de servicios, Washington, DC.

En el año 2002, Pisinger (Pisinger, 2002) presentó un algoritmo heurístico basado en una aproximación *wall-building* para el *Container Loading Problem,* este descompone el problema en capas que luego son divididas en un cierto número de tiras. Luego, cada tira se formula y resuelve acercándose al óptimo con un algoritmo *Branch-and-bound* como un problema de la mochila, cuya capacidad es igual al ancho o largo del contenedor, también, cada nodo se explora sólo uno de los subconjuntos de los nodos. Este método permite llenar más del 95% del volumen total para instancias de gran tamaño. Este algoritmo difirió en su momento de otros algoritmos tipo *wall-building* en varios aspectos, como usar el esquema de enumeración de *m-cut* para elegir ubicaciones de los productos, además, el algoritmo tiene la facultad de retroceder para alcanzar combinaciones que funcionen bien juntas.

Más tarde y a raíz de lo realizado por Pisinger (Pisinger, 2002), Borfeldt y Mack en 2007 (Bortfeldt & Mack, 2007) investigan dos aproximaciones para adaptarlas desde el *CLP* (Pisinger, 2002) al 3*D Strip Packing Problem*, la primera, mira directamente a un contenedor de largo prácticamente infinito, mientras la otra, resuelve una instancia *SPP* al calcular una serie de instancias tipo *CLP* con contenedores con largo descendente. Las mejores partes de las mejores soluciones son reutilizadas sistemáticamente con la segunda aproximación, la cual probó ser más exitosa. Al compararse a la segunda aproximación con las de la literatura de ese tiempo, se demostró un gran rendimiento independientemente del volumen utilizado.

En el año 2014, Araya y Riff (Araya & Riff, 2014), estudian un *Beam Search* o Búsqueda Haz para el *single container loading problem*, que es un tipo de *Branch-and-Bound* en el cual sólo se expanden los nodos más prometedores en cada nivel del árbol de búsqueda. Esta estrategia mantiene un conjunto de distintas ramas y no sólo una, como la mayoría de este tipo de heurísticas de aproximación. Este método también presenta un mecanismo simple para remover ramas que sean similares, para mantener un nivel mínimo de diversidad entre los restantes, esto último fue crucial

para mejorar la búsqueda. Esta publicación, alcanzó los mejores resultados conocidos para el problema en su momento, los cuales fueron comparados con 16 conocidos algoritmos.

Anabalón et. al (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) en el año 2021, entregan una innovadora propuesta de solución para el *Truck Loading Problem,* en esta se realiza, en primer lugar, una heurística constructiva basada en una aproximación tipo *Wall Building,* que provee de una solución inicial muy rápida para el problema, y luego, un algoritmo de búsqueda armónica o *Harmony Search* es aplicado para mejorar la solución. Este algoritmo, es relativamente nuevo y está basado en el proceso de improvisación de los músicos. Cabe destacar, que la metaheurística es guiada a través de una función de evaluación basada en la optimización de las entregas multicliente. En este trabajo, se toman en cuenta restricciones físicas, de apilado y de entregas múltiples.

3.3.2 Algoritmos Voraces

Un algoritmo voraz es un algoritmo simple usado en problemas de optimización. Este tipo de algoritmos, en cada paso, seleccionan siempre el óptimo local, buscando encontrar una solución general óptima sin tener una visión global del problema.

En el año 2005 (Lim & Zhang, 2005) Lim y Zhang, abordaron el *Multiple container loading problem (MCLP)* proponiendo una priorización dinámica para las cajas con formas extrañas. La caja con mayor prioridad se ubica en las partes más bajas del contenedor, o bien empacada en los primeros contenedores. Las soluciones encontradas en una iteración son analizadas, y las prioridades son actualizadas para la siguiente iteración. Durante una iteración del algoritmo, se aplica un algoritmo voraz integrado con un árbol de búsqueda para cargar el contenedor. Este método logró superar a otros métodos previos para el *MLCP*, dando resultados excelentes para el problema trabajado.

Zhu et al. (Zhu, Oon, & Lim, 2012) en el año 2012, notaron que gran parte de las técnicas para resolver el *Single Container Loading Problem* tienen estructuras relacionadas al uso de bloques de cajas. En esta investigación se determinan 6 elementos comunes entre los algoritmos tipo *block building,* luego, estas 6 estrategias son combinadas con una heurística *greedy* para el S*CLP*.

Alonso et al. (Alonso, Alvarez-Valdes, Parreño, & Tamarit, 2016) en el año 2016 realizaron una solución para el *Truck Loading Problem*, acá, además de usar las restricciones de no sobreponer productos entre ellos y no exceder el tamaño de los pallets en los camiones, se agregaron varias otras, relacionadas al peso total de la carga, el peso máximo aguantado por cada esquina del camión, así como la distribución de la carga dentro del camión. El problema, además, puede ser separado en 2 fases, carga de pallets y carga de camión, sin embargo, en esta investigación se desarrolló una aproximación combinada, construyendo y ubicando los pallets al mismo tiempo que se ubican en el camión. La idea es mantener un flujo constante información entre los 2 subproblemas. Para esto se diseñó un algoritmo *GRASP (Greedy Randomized Adaptative Search Procedure),* en el cual la solución inicial es construida con componentes aleatorios, donde luego mediante una fase de mejora se obtienen mejores resultados.

3.3.3 Algoritmos Genéticos

Algoritmos basados en el funcionamiento de la teoría de evolución de Darwin. Este procedimiento parte codificando apropiadamente las soluciones del espacio de búsqueda, así como una manera correcta de evaluar la función objetivo para dichas soluciones. Cada solución es un individuo y cada individuo posee su cromosoma, que a su vez está compuesto de un cierto número de genes a los que les corresponden ciertos alelos. A estos individuos, se le aplican 2 operaciones básicas: mutación y cruce. La mutación hace que un individuo cambie al azar un gen. El cruce de dos padres genera a un hijo que toma los genes de ambos padres. De esta forma, la población evoluciona con estrategias de selección de individuos (Santana, y otros, 2004).

En el año 1997, Ghering (Gehring, 1997) presenta un algoritmo genético para el *Container Loading Problem.* En esta aproximación, primero se genera un conjunto disyuntivo (sin elementos en común) de torres de cajas y luego, se ubican estas torres de cajas en el piso del contenedor según un criterio de optimización. Esta solución es validada al ser comparada con otros procedimientos de la época. Cabe destacar, que este AG sirve para *CLPs* cuyos requerimientos de estabilidad sean simples.

3.3.4 *Tabu Search*

La búsqueda tabú es un algoritmo de solución única que usa el concepto de memoria y lo estructura de forma simple para dirigir la búsqueda tomando en cuenta los pasos realizados de ésta, es decir, se busca usar la experiencia para actuar en consecuencia. Visto así, se puede decir que existe aprendizaje e inteligencia. Una característica clave de la búsqueda tabú, es el uso de memoria adaptativa, la cual puede explotarse con estrategias especiales. Esta memoria es usada en dos componentes importantes de la búsqueda tabú, las cuales son intensificación y diversificación. (Glover, Taillard, & Taillard, 1993)

En Bortfeldt et al. (2003) un algoritmo paralelo *Tabu Search* es presentado para el *container loading problem* con carga débilmente heterogénea. El procedimiento consiste en realizar búsquedas múltiples tabú, con grados de sincronización y cooperación donde se intercambian las mejores soluciones al final de cada fase anteriormente definida de búsqueda.

Para el año 2006, Gendreau et al. (Gendreau, Iori, Laporte, & Martello, 2006) investigan una solución para el problema combinado de ruteo y *3D loading problem,* tomando también en cuenta restricciones presentes en el área de transporte de carga. Dentro de esta investigación las cajas pueden ser rotadas en términos de ancho y largo, mientras que su posición respecto a la altura es fija. Además, cada ítem posee un valor de fragilidad que es 1 en caso de que el ítem sea frágil, o 0 en caso de no serlo. También, se añade un área de contacto mínimo a las cajas, al igual que se dejan las cajas que se sacarán primero en la parte de delante. Finalmente, las cajas pertenecientes a un mismo cliente deben ser ubicadas en el mismo lugar.

Con el objetivo de resolver ambos problemas, se aplica una búsqueda tabú para el problema de ruteo, que a su vez aplica iterativamente una rutina para resolver el *Single-Vehicle Loading Problem.* La rutina mencionada hace uso de un algoritmo tabú simple que utiliza dos heurísticas tipo *greedy* para la exploración del vecindario. Si no se encuentra ninguna solución factible, el algoritmo devuelve un resultado no factible. Y, si se superó el peso o volumen máximo, una solución del doble de largo que el máximo es entregada.

En el año 2009, Crainic et al. (A Two-Level Tabu Search for the Three-dimensional Bin Packing Problem, 2009) desplegaron una heurística para el *3D Bin Packing Problem.* En este problema, las cajas deben ser ubicadas ortogonalmente en la menor cantidad de contenedores posible. Acá se

presentó una búsqueda tabú de 2 niveles. El primer nivel tiene el propósito de reducir el número de contenedores. La segunda optimiza la carga de paquetes en cada contenedor. Este proceso, está basado en la representación del empaquetamiento en gráficos de intervalos, propuesto por Fekete y Schepers (Fekete & Schepers, A Combinatorial Characterization of Higher-Dimensional Orthogonal Packing, 2003) en 2004. Además, en esta investigación se agrega un método general para incrementar el tamaño de los vecindarios asociados, y por ende la calidad de la búsqueda, sin aumentar la complejidad del algoritmo.

Ailian et al. (Ailian, Baisong, Jun, & Liangfeng, 2010) en 2010 presentaron una adaptación del algoritmo de búsqueda tabú para el *Military Airlift Loading Problem o* problema de carga del puente aéreo militar maximizando masa o volumen, en esta solución, se adaptan la función de evaluación del algoritmo al problema y se crea una función de penalización basada en la diferencia entre los límites del problema y las soluciones entregadas en las iteraciones.

(Wisniewski, Ritt, & Buriol, 2011) En 2011 Wisniewski et al. resolvieron el problema de ruteo y carga de vehículo capacitado sin *overlap,* con orientación, fragilidad (1-0), manteniendo un mínimo de área de soporte de 75% para cada caja, las cajas se ordenan según orden de entrega (los últimos serán los primeros) o LIFO. Usan la misma aproximación de 2 pasos de Gendreau et al. (Gendreau, Iori, Laporte, & Martello, 2006), pero al considerar el ruteo y el empaquetamiento por separado, se soluciona cada uno con su algoritmo. *Tabu Search* se usa en el problema de ruteo y no en la carga del vehículo.

(Liu, Yue, Dong, & Keech, 2011) En Liu et al. 2011 trabajaron un *CLP* en el que se considera estabilidad pues las cajas deben estar cubiertas por debajo al 100%, puede rotarse, cajas iguales son cargadas juntas para ahorrar tiempo de carga y descarga. En la solución que es una aproximación híbrida de búsqueda tabú, primero se aplica una heurística de carga que incluye un método para llenar los espacios vacíos para generar soluciones cercanas al óptimo considerando la estabilidad. Luego, de hacer funcionar dicha heurística se aplica la búsqueda tabú como un método para mejorar la solución dada.

(Viegas, Vieira, Henriques, & Sousa, 2015) Viegas et al. 2015 Algoritmo búsqueda tabú y algoritmos *best-fit drecreasing* para el *3D Bin Packing Problem* en la industria del metal, productos se separan por cliente. El algoritmo de búsqueda tabú se aplica luego de una heurística constructiva

tipo *First-fit Decreasing*. El algoritmo propuesto es comparado con algoritmo ACO o de colonia de hormigas.

3.4 Trabajos relacionados

3.4.1 Memoria de Nicolás Anabalón

En el trabajo realizado por Anabalón en 2021 (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), se resuelve el problema de carga de un camión con cargas heterogéneas, respetando restricciones que facilitan el despacho de las cajas a distintos lugares y clientes, utilizando un enfoque de dimensión abierta. Para la resolución se utiliza una heurística constructiva tipo *Wall Building* y un algoritmo de mejora basado en la metaheurística *Harmony Search* (Yang, 2009). Se logró resolver e implementar soluciones factibles para el problema.

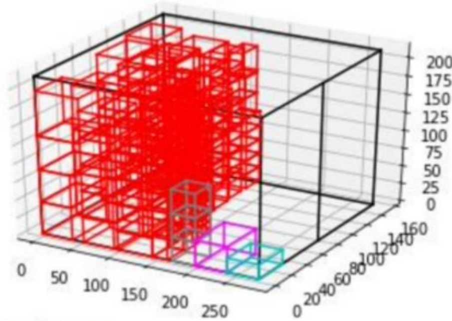


Figura 1: Solución instancia 5 *Wall Building*

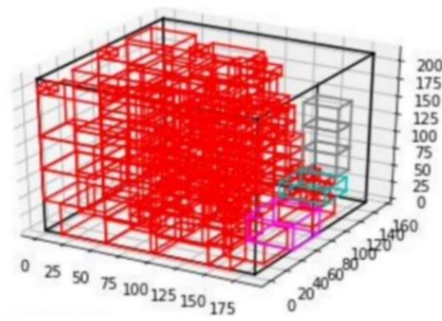


Figura 2: Solución instancia 5 *Harmony Search*

En el presente trabajo, se aplican las mismas restricciones de factibilidad y, por ende, es posible utilizar ambos algoritmos para encontrar nuevas formas de resolver el problema. De ahí se utiliza una modificación de la heurística tipo *Wall Building* a la que se llamará *WBm (Wall Building* modificado).

Dentro de las características que se tomaron del trabajo (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), se encuentran las restricciones físicas, que incluyen la restricción de la superposición de las cajas, la restricción que limita las cajas de manera que se encuentren en su totalidad dentro del camión y la restricción de que cada caja ubicada debe tener el 50% de su área inferior en contacto con otra superficie. Por otro lado, también se encuentran las restricciones de

apilado, una caja de mayor peso p_i no puede quedar encima de una caja de menor peso, de ocurrir esto, si una caja de mayor peso queda arriba de otra de menor peso y, además el contacto entre dichas cajas sea menor al 50%, entonces se cambia la posición de la caja de arriba a una posición factible. En tercer lugar, se incluyen restricciones de entregas múltiples, donde las cajas deben quedar ordenadas considerando la ruta del camión y las cajas de un mismo cliente deben quedar próximas entre sí.

3.4.1.1 Restricciones físicas

El problema consiste en ubicar n cajas dentro de un camión, las cuales no pueden ser rotadas y deben ubicarse de forma paralela a las paredes del camión. Las características para cada caja i son conocidas y denotadas por: largo l_i, ancho w_i, alto h_i, peso p_i y cliente k_i. Por otro lado, existen N clientes ordenados de acuerdo con la ruta de entrega. Se define Δ_j al conjunto de cajas del cliente j, donde se encuentran las cajas ordenadas por peso descendente.

El *Truck Loading Problem* trabajado en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) posee 2 importantes restricciones físicas. En primer lugar, se necesita que las cajas se encuentren totalmente contenidas dentro del camión, para esto debe cumplirse la siguiente restricción:

$$0 \leq x_i < x_i + l_i \leq L$$

$$0 \leq y_i < y_i + w_i \leq W$$

$$0 \leq z_i < z_i + h_i \leq H$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

Donde (x_i, y_i, z_i) es la posición de la caja i. Cada una de las ecuaciones representan a un eje del camión, la primera ecuación representa al eje x, en este caso se establece a que la posición en x de cualquier caja más el largo de esta caja debe ser menor al largo total del camión. La segunda ecuación establece que la posición en el eje y de cualquier caja sumada con su ancho no debe superar al ancho del camión. La tercera ecuación evita que la posición en z de cualquier caja sumado al alto de la caja no supere al largo del camión.

Otra de las restricciones que se tomó en cuenta se usa para evitar la superposición de las cajas, es decir, que 2 cajas no utilicen un mismo lugar o espacio físico dentro del camión, estas restricciones pueden verse aquí, ejemplificadas para el eje X:

$$x_i \leq x_u \leq x_u + l_u \leq x_i + l_i$$
$$x_u \leq x_i \leq x_i + l_i \leq x_u + l_u$$
$$x_i \leq x_u < x_i + l_i \leq x_u + l_u$$
$$x_u \leq x_i < x_u + l_u \leq x_i + l_i$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

Si la primera caja ya sea i o u se encuentra en una posición del eje x anterior a la segunda caja y la segunda caja se ubica entre la posición de la primera caja y la suma de esta posición con el largo de la primera caja, entonces es una posición no factible, pues la segunda caja se estaría ubicando en el lugar donde hay una caja ya ubicada. Finalmente, si una de estas condiciones no se cumple se considera que estas cajas están sobrepuestas y se utiliza a la función *mover(i) (4.3.2)* para ubicarla en una posición factible.

3.4.1.2 Restricciones de Apilado

Para mantener una factibilidad en la forma en que se apilan las cajas, es decir, que se mantenga el equilibrio y las cajas no se caigan en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) se utilizan 2 importantes condiciones para las cajas. En primer lugar, una caja i con un peso p_i no puede ubicarse encima de una caja j con un peso p_j menor. En segundo lugar, al momento de ubicar una caja i, el 50% área de su cara inferior $l_i \cdot w_i$ debe estar en contacto con la caja inferior j, es decir, $l_i \cdot w_i \cdot 0{,}5 < l_j \cdot w_j$.

3.4.1.3 Restricciones de Entregas Múltiples

Consisten en buscar que las cajas de un mismo cliente se encuentren cercanas entre sí y, además, en que se las cajas se dispongan de tal manera que se respete el orden de entregas, usando la técnica de gestión de la carga LIFO (“*Last In, First Out*”), cuyas siglas en español significan “último en entrar, primero en salir”, técnica que también fue utilizada en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) al ubicar primero las cajas del cliente que recibirá en último lugar su pedido.

Tal y como se realizó en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) en este trabajo se busca que la distancia entre las cajas de un mismo cliente sea mínima, para esto cuando se ubican las cajas se ordenan previamente por peso decreciente. Además, mediante funciones se evalúa la cercanía de las cajas y se intentan juntar lo más posible, estas mismas funciones son utilizadas en este trabajo. Funcionan al ubicar una caja, si esta caja no está cerca de una caja del mismo cliente, se ubica cercana a alguna caja del cliente anterior y si no, el subsiguiente.

3.4.1.4 Selección de Soluciones

Al igual que en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), en este trabajo se buscar minimizar en primer lugar el largo de las soluciones, en segundo lugar, se busca que las soluciones cumplan con las restricciones de entregas múltiples de las cajas junto con minimizar el volumen de las soluciones. Tanto es así, que el algoritmo de selección tomado de este trabajo selecciona la solución de menor largo, pero, si hay dos soluciones con el mismo largo, entonces se privilegia la solución con menor penalización o volumen.

Por otro lado, para que las cajas de un mismo cliente se encuentren cercanas entre sí, se considera una penalización de posición y volumen de las cajas al igual que en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Para el cálculo de la penalización, se considera que por cada cliente $j \in \{2, \dots, N\}$, se define λ_j un limite en el eje X de la posición de sus cajas y Δ_{j-1}, que indica el grupo de cajas pertenecientes al cliente j de acuerdo con la siguiente ecuación:

$$\lambda_j = max\{x_i + l_i : i \in \Delta_{j-1}\} \ \forall_j = \{2, \dots, N\}$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

Parafraseando a (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Por ejemplo, para el cliente 1 no existe penalización. Para el cliente 2 se penaliza cada caja que este antes de una caja del cliente 1, y así sucesivamente. De este modo, las cajas del cliente 1 quedarán cercanas al origen, ya que son las últimas en entregarse. Análogamente, se considera λ_j, el límite hacia adelante que deben respetar las cajas, para todos los clientes j menos el último.

$$\dot{\lambda}_j = min\{x_i : i \in \Delta_{j+1}\} \ \forall_j = \{1, \dots, N-1\}$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

Dadas estas definiciones, la penalización por orden de entrega se define como:

$$Pen = \frac{\sum_{j=2}^{N} \sum_{i \in \Delta_j} Max\{0, \lambda_j - x_i\}}{2L} + \frac{\sum_{j=1}^{N-1} \sum_{i \in \Delta_j} Max\{0, x_i + l_i - \dot{\lambda}_j\}}{2L}$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

El cálculo del volumen se realiza al tomando la posición máxima y restándola con la posición mínima de las cajas en cada eje cartesiano y multiplicando estos resultados entre sí. El volumen total se obtiene al sumar el volumen calculado con el procedimiento anterior para las cajas de cada cliente. A continuación, se observa en la ecuación.

$$V_j = \left(max_{i \in \Delta_j}\{x_i + l_i\} - min_{i \in \Delta_j}\{x_i\}\right) \cdot$$

$$\left(max_{i \in \Delta_j}\{y_i + w_i\} - min_{i \in \Delta_j}\{y_i\}\right) \cdot$$

$$(max_{i \in \Delta_j}\{z_i + h_i\} - min_{i \in \Delta_j}\{z_i\})$$

$$V = \sum_{j=1}^{N} V_j$$

(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

Para este trabajo, se toma prioritariamente en cuenta el largo, luego en caso de encontrar una solución con igual largo, se decide cual se seleccionará mediante la penalización por orden de entrega o por el volumen calculado.

3.4.2 Touching Perimeter Algorithm

Método que, en primer lugar, ordena las cajas por área de mayor a menor, luego, empaca caja por caja ya sea en un container, o en un camión como es el caso de esta investigación. El primer ítem siempre se pone en el punto más a la izquierda, atrás y abajo posible del contenedor. Cada elemento subsiguiente se ubica en una "posición normal" o donde se genere una esquina de 90 grados,

también, la parte de abajo del elemento debe tocar o al fondo del contenedor, o bien, la parte de arriba de otro ítem. Con su lado izquierdo, el lado izquierdo del contenedor, o bien, el lado derecho de otra caja (Lodi, Martello, & Vigo, 1999).

En 1999, Lodi et al. (Lodi, Martello, & Vigo, 1999) desarrollan varios algoritmos para solucionar el *2D bin Packing Problem,* construyendo 6 distintas soluciones iniciales aplicando una búsqueda tabú para mejorar las soluciones. Una de las soluciones iniciales utiliza *Touching Perimeter Algorithm (TPA),* entregando en promedio buenos resultados en comparación con las otras. De acá, se toma la idea de utilizar *TPA* como un posible algoritmo constructivo para el problema. Cabe destacar, que en la investigación citada resuelve el problema *2D Bin Packing Problem,* por lo tanto, el algoritmo, tal como está, no serviría para resolver el *TLP* de este trabajo.

En la figura 3, se muestra el pseudocódigo utilizado en Lodi et al. (1999) (Lodi, Martello, & Vigo, 1999) para el problema *2D bin Packing Problem.* Para el caso del *TLP* no se utiliza la primera parte, donde se incluye el cálculo del número de bins utilizados para *2D Bin Packing Problem.*

```
algorithm TP_RF:
  sort the items by nonincreasing w_j h_j values, and
  horizonally orient them;
  comment: Phase 1;
  compute a lower bound L on the optimal solution value,
  and open L empty bins;
  comment: Phase 2;
  for j := 1 to n do
    score := 0;
    for each normal packing position in an open bin do
      let score_1 and score_2 be the scores associated with
      the two orientations;
      score := max{score, score_1, score_2}
    end for;
    if score > 0 then
      pack item j in the bin, position and orientation
      corresponding to score
    else
      open a new bin, and horizontally pack item j into it
  end for
```

Figura 3: Pseudocódigo *Touching Perimeter Algorithm* Lodi et al. (1999) [36]

En el año 2003, Iori et al. (Iori, Martello, & Monaci, 2003) desarrollan un modelo híbrido de solución para el *2D Strip Packing Problem*, desarrollando tres soluciones iniciales diferentes y tres algoritmos de mejora trabajando con un algoritmo genético, uno de búsqueda tabú y uno híbrido

construido entre los dos ya mencionados. Una de las soluciones iniciales es una adaptación del *Touching Perimeter Algorithm.* La aplicación de una de las soluciones en este trabajo fue desarrollada por (Pönitz, 2020) en Python. En el presente trabajo, se utiliza este *Touching Perimeter Algorithm* para resolver de forma constructiva el *TLP* presentado en esta investigación. El código fuente se encuentra en este repositorio: https://github.com/tasptz/py-touchingperimeter, que al ser este diseñado originalmente para resolver un problema en dos dimensiones se adapta en el presente trabajo tratando cada muralla formada por cajas como un problema distinto hasta haber ubicado cada caja.

3.5 Metaheurística *Tabu Search*

Se comienza con una solución inicial, sobre la cual se generan vecindarios o movimientos posibles. Originalmente, en *Tabu Search* se verifican todos los movimientos posibles desde la solución actual, sin embargo, debido a limitaciones propias del problema y buscando eficiencia computacional se genera sólo un número limitado de vecindarios.

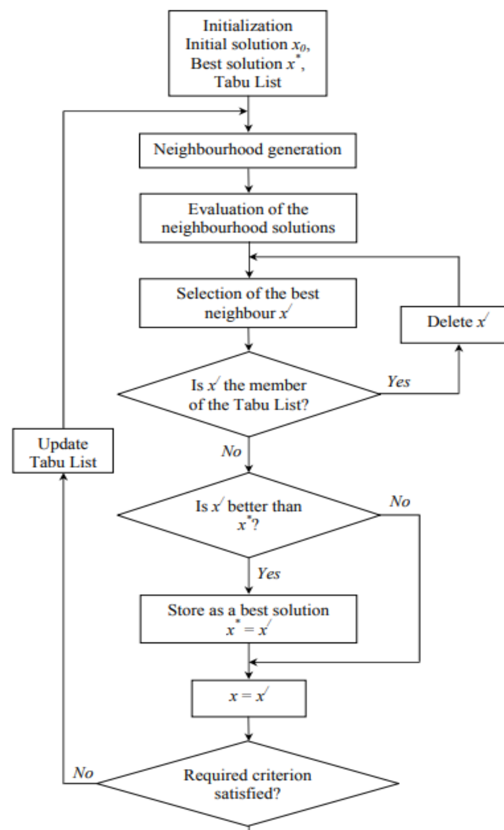


Figura 4: Pseudocódigo *Tabu Search*, Stepanenko, 2008 [39]

En la figura 4 es posible observar el proceso general de un algoritmo *Tabu Search*, para el desarrollo de la metaheurística diseñada en esta investigación se utilizó como ejemplo un esquema como este, tomando los pasos de generación de vecindarios, evaluación de dichos vecindarios, selección el mejor vecindario y descartándolo en caso de encontrarse que dicho movimiento es tabú, así como guardar las mejores soluciones y consultando si es que son elementos tabú o que ya ocurrieron hace un cierto número de iteraciones.

4. Metodología

En este trabajo, se toma como un vecindario, una disposición de cajas diferente generada a partir de la solución trabajada en la última iteración, esta disposición es registrada en una matriz con las posiciones de la esquina inferior trasera de cada caja a cargar en el camión de reparto, considerando un espacio de tres dimensiones (x, y, z) (Hamdi-Dhaoui, Labadie, & Yalaoui, 2012).

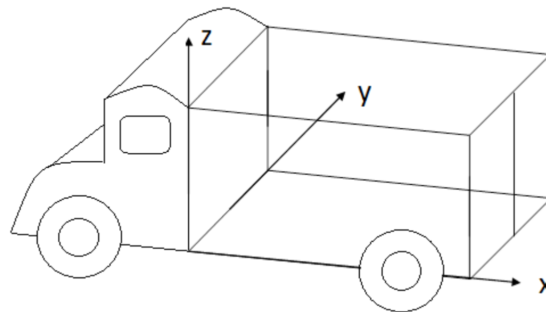


Figura 5: Ejes del problema *(Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)*

Con el fin de modelar el problema se usa el mismo sistema coordenado que en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) donde se toma en cuenta un camión de largo L, ancho W y alto H. El punto $(0,0,0)$ será la esquina inferior izquierda trasera de la zona de carga del camión, dejando al eje de coordenadas tal como se observa en la figura 5.

Las soluciones para el problema deben mostrar la posición de cada caja, representada por la coordenada de la esquina inferior izquierda de ésta, que respeten las restricciones físicas de factibilidad, cómo la contención, sobreposición y apoyo; y también, las restricciones de apilado, estabilidad y de entregas múltiples.

Utilizando el enfoque ***ODP/W*** (Wäscher, HauBner , & Schumann, 2007) usado en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Se busca minimizar el largo ocupado por las cajas. Por ende, el algoritmo de selección busca, en primer lugar, minimizar el largo utilizado, sin embargo, en caso de que dos soluciones presenten el mismo largo, se preferirá la solución en que las cajas de cada cliente se encuentren más cercanas entre sí, o bien, la que presente un menor volumen o penalización 3.4.1.4.

4.1 Heurísticas Constructivas

Para construir la solución inicial, se estudian 3 opciones distintas; por un lado, se puede usar un enfoque determinista de *Wall building* igual al utilizado en Anabalón et al. (2021); también puede utilizarse una modificación de esa heurística que busca disminuir el largo inicial de la solución, o bien, una solución tipo *Touching Perimeter.* Debido a que las cajas están ordenadas por peso, y orden de entrega; estos métodos entregan soluciones factibles para la problemática, pues respeta las restricciones físicas de empilado y estabilidad, manteniendo a las cajas de un mismo cliente cercanas entre sí, respetando las restricciones tipo LIFO (*Last In First Out).* Finalmente, para los experimentos se utilizará el algoritmo que presente los mejores resultados, este se utilizará para comparar las 2 metaheurísticas estudiadas en esta investigación.

4.1.1 *Wall Building*

Inicialmente, se considera el camión vacío, se ordenan las cajas de cada cliente en orden peso decreciente. Mientras las cajas sean del mismo cliente, la siguiente caja en la lista se coloca en la posición más atrás, a la izquierda y abajo posible. Si corresponde a un nuevo cliente, se inicia una nueva fila, en la base del camión y las cajas se ponen arriba mientras no se sobrepase la altura del camión. Si se sobrepasa la altura, la siguiente caja se ubica a la derecha de la columna anterior, inicializando una nueva columna, mientras no se sobrepase el ancho del camión. Si se sobrepasa el ancho, se inicializa una nueva fila de cajas. Esto se repite, mientras existan cajas sin cargar en el camión. (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021)

4.1.2 *Wall Building* modificado

Similar al procedimiento anterior, pero en lugar de comenzar a ubicar las cajas de un nuevo cliente en una nueva fila, se continúa en la misma fila en que se ubicó la última caja del cliente anterior, ahorrando bastante espacio que muchas veces no estaba siendo ocupado por las cajas y que sólo mediante la metaheurística se lograba llenar. Además, a diferencia de la heurística *Wall Building,* se evalúa que cada caja cumpla con las restricciones de área de contacto.

En la línea 1 se encuentra un *for* que recorre todas las cajas, entre las líneas 2 y 4 se ubica la primera caja y se guarda su ancho y largo en **maxx** y **maxy.** El *else* de la línea 5 a 70 es donde se comienzan a procesar todas las cajas que no son la primera caja. En el *if* de la línea 6 a 23 se pregunta si la caja *i* no pertenece al mismo cliente de la caja anterior, de ser así y si queda espacio a lo ancho esta caja tomará el mismo valor de posición de ***x*** que la caja del cliente anterior, ubicándose en el eje ***y*** en la posición justo después de la caja del cliente anterior. Si no queda espacio en lo ancho, la caja se ubicará en la posición más externa de la caja más prominente de la fila anterior. En cualquiera de los casos, al tratarse de distintos clientes, las cajas deben comenzar a ubicarse en la base del camión o posición 0 de ***z*** con el fin de mantener las soluciones factibles.

En la línea 23 a 64, se muestra el caso de cuando la caja actual pertenece al mismo cliente que la caja anterior, desde la línea 24 a 29 se pregunta si queda espacio para ubicar cajas en la misma posición de ***x*** de la caja anterior, de ser así se ubica en la misma posición de ***x*** que la caja anterior. De no ser así, se ubica la caja en una nueva posición de ***x*** comenzando una nueva fila de cajas en la posición más externa de la caja más prominente de la última fila. Por otro lado, en la línea 34 se verifica el caso de que no quede espacio ni en el eje ***y*** ni en el eje ***z***, de ser así, la caja se ubica en una nueva posición de ***y*** de valor 0. Luego, en la línea 39 se verifica el caso de que no quede espacio en el eje ***y***, pero si en el eje ***z***, que de ser así se ubica la caja encima de la caja anterior, manteniendo el mismo valor de posición en ***x*** e ***y***. En la línea 43 se verifica el caso en que quede espacio en el eje ***y***, pero se haya completado la columna o no haya espacio en el eje ***z*** para poner más cajas, de ser así se ubica a la caja en la base del camión justo al lado de la última columna de cajas formada, manteniendo el valor de posición en el eje ***x*** y cambiando el valor de ***y*** y ***z.*** Luego, en la línea 46 se verifica el caso en que quede espacio arriba, de ser así se verifica que la cara inferior de la caja que se está ubicando arriba se encuentre en contacto de al menos un 50% con la cara superior de la caja de abajo, si esto no se cumple, la caja se ubica en la base del camión. En la línea 55, si no queda espacio en el eje ***z*** la caja se ubica en la base del camión, de no ser así en la línea 58 se verifica que la caja que se va a ubicar con la caja de abajo tengan un 50% de contacto entre superficies, ubicando la caja recorrida encima de las otras cajas. Luego, en la línea 60, se verifica el caso en que no se cumpla la restricción del contacto entre superficies, dejando la caja en la base del camión.

Finalmente, en la línea 65 se actualiza la variable *fin* que ayuda a ubicar las cajas en una posición de ***x***, esta es actualizada cada vez que se ubica una caja pues siempre la siguiente fila de cajas se ubica en el lado más externo de la caja más prominente de la fila actual.

Algorithm 1 Wall Building modificado

Require : $fin = 0$

1: **for** $\mathbf{i} = 1, 2, \ldots, n$ **do**
2:     **if** $i = 0$ **then**
3:         $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) = (0, 0, 0)$
4:         $\mathbf{maxx} \leftarrow \mathbf{l}_i$, $\mathbf{maxy} \leftarrow \mathbf{w}_i$
5:     **else**
6:         **if** $\mathbf{k}_i =! \mathbf{k}_{i-1}$ **then**
7:             **if** $\mathbf{y}_{i-1} + \mathbf{w}_{i-1} + \mathbf{w}_i <= \mathbf{W}$ **then**
8:                 $\mathbf{x}_i = \mathbf{x}_{i-1}$
9:                 $\mathbf{maxx} \leftarrow \mathbf{l}_i$
10:             **else**
11:                 $\mathbf{x}_i = fin$
12:                 $\mathbf{maxx} = [\mathbf{l}_i]$
13:             **end if**
14:             **if** $\mathbf{y}_{i-1} + \mathbf{w}_{i-1} + \mathbf{w}_i > \mathbf{W}$ **then**
15:                 $\mathbf{y}_i = 0$
16:                 $\mathbf{maxy} = [\mathbf{w}_i]$
17:             **else**
18:                 $\mathbf{y}_i = \mathbf{y}_{i-1} + \mathbf{maxx}(\mathbf{maxy}_{\mathbf{nc}})$
19:                 $\mathbf{maxy} \leftarrow \mathbf{w}_i$
20:             **end if**
21:             $\mathbf{z}_i = 0$
22:             $\mathbf{maxy} = [0]$
23:         **else**
24:             **if** $\mathbf{y}_{i-1} + \mathbf{w}_{i-1} + \mathbf{w}_i <= \mathbf{W}$ **then**
25:                 $\mathbf{x}_i = \mathbf{x}_{i-1}$
26:                 $\mathbf{maxx} \leftarrow \mathbf{l}_\mathbf{i}$
27:             **else if** $\mathbf{z}_{i-1} + \mathbf{h}_{i-1} + \mathbf{h}_i <= \mathbf{H}$ **then**
28:                 $\mathbf{x}_i = \mathbf{x}_{i-1}$
29:                 $\mathbf{maxx} \leftarrow \mathbf{l}_\mathbf{i}$
30:             **else**
31:                 $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{max}(\mathbf{maxx})$
32:                 $\mathbf{maxx} = [\mathbf{l}_i]$
33:             **end if**

```
34:             if y_{i-1} + w_{i-1} + w_i > W then
35:                 if z_{i-1} + h_{i-1} + h_i > H then
36:                     y_i = 0
37:                     maxy_nc = maxy
38:                     maxy = [w_i]
39:                 else
40:                     y_i = y_{i-1}
41:                     maxy ← w_i
42:                 end if
43:             else if z_{i-1} + h_{i-1} + h_i > H then
44:                 y_i = y_{i-1} + max(maxy)
45:                 maxy = [w_i]
46:             else
47:                 if l_i * w_i * 0,5 < l_{i-1} * w_{i-1} then
48:                     y_i = y_{i-1}
49:                     maxy ← w_i
50:                     maxy_nc = maxy
51:                 else
52:                     y_i = y_{i-1} + max(w_i)
53:                 end if
54:             end if
55:             if z_{i-1} + h_{i-1} + h_i > H then
56:                 z_i = 0
57:             else
58:                 if l_i * w_i * 0,5 < l_{i-1} * w_{i-1} then
59:                     z_i = z_{i-1} + h_{i-1}
60:                 else
61:                     z_i = 0
62:                 end if
63:             end if
64:         end if
65:         if x_i + l_i > fin then
66:             maxx ← l_i
67:             fin = x_i + maxx(maxx)
68:             maxx = maxx − l_i
69:         end if
70:     end if
71: end for
```

Algoritmo 1: Wall Building modificado

4.1.3 *Touching Perimeter Algorithm*

Este algoritmo es una adaptación de un algoritmo aplicado en un *2D Strip Packing Problem* (Pönitz, 2020), en él se busca llenar un espacio de 2 dimensiones con cajas de sólo 2 dimensiones ocupando el menos espacio posible. El algoritmo ordena las cajas de mayor a menor en área, luego ubica en orden cada caja buscando que la caja actual se ubique en el espacio de 2 dimensiones donde más cubierta por otras cajas o contornos del camión quede, es decir donde su perímetro tenga más contacto con otros elementos. Para el caso de este trabajo se utiliza el algoritmo de la misma forma que en dicho trabajo, solo que aplicando el algoritmo en *2D* para la formación de cada plano (y-z) de cajas en el camión.

Este código utiliza las funciones desarrolladas por Pönitz en 2020 (Pönitz, 2020), quien construyó una adaptación de este algoritmo para el *2D Strip Packing Problem* presentado en (Hamdi-Dhaoui, Labadie, & Yalaoui, 2012). Consiste en construir una solución que ordena los rectángulos en un espacio rectangular, utilizando la lógica del *Touching Perimeter Algorithm,* para poder aplicar esta solución en el problema de esta investigación que consta de 3 dimensiones, se utiliza el procedimiento descrito abajo. Antes de comenzar con la explicación del pseudocódigo se hace necesario explicar que 3 clases con funciones se sacan del código mencionado, las cuales son $Rect$, $Caja$ y $empaquetador$. La primera clase, adapta el camión a una versión en 2 dimensiones (mirando el plano ***yz***), $Caja$ adapta las cajas en 3 dimensiones a cajas en 2 dimensiones para poder ser usadas en el algoritmo, finalmente, $empaquetador$, como bien dice su nombre, empaqueta las cajas, la función *empacar* de *empaquetador* retorna *c1, c2, c3* y *c4*, que tomarán los valores **True**, ***y***, ***z*** y ***x*** en caso de ser posible ubicar la caja actual en algún lugar del plano trabajado, de no ser así esta función tomará el valor **False**, **False**, **False** y **False**. Para poder ubicar las cajas en el eje ***x***, se utiliza el procedimiento del pseudocódigo de abajo.

En primer lugar, se inicializan los valores del tamaño del camión, lista $\boldsymbol{maxx}$, lista $cajas$, variable ***fin***, c1, c2, c3 y c4 en la línea 1 que servirán para poner en lugares factibles los diferentes planos de cajas. En la línea 3 se define el espacio de 2 dimensiones en la cual se irán ubicando las cajas hasta llenarlo, este espacio corresponde al plano ***yz*** que es de tamaño $W\ x\ H$. La lista $cajas$ es llenada en el ciclo del punto 4 con las cajas en el formato clase $Caja$, que luego se irán ubicando en el camión.

En el *for* de las líneas 7 a 37 se ubican todas las cajas en el camión. En las líneas de 8 a 13 se ubica la primera caja, en el punto 9 se saca la caja $caja_0$ de la lista $cajas$ mientras en la línea 10 se aplica

la función $empacar$ de $empaquetador$, dejando la caja ubicada en la posición (0, 0, 0), luego, en la línea 12 se agrega el largo de la caja a la lista $\boldsymbol{maxx}$**,** la lista auxiliar que ayuda a ubicar las cajas en posiciones factibles en el eje **x**.

Luego, desde la línea 14 a la 30 se procesan todas las cajas que no son $caja_0$. Primero, de la línea 14 a 20 se consulta si la caja procesada no pertenece al mismo cliente que la última caja ubicada, de ser así se repite el proceso de ubicar tal y como para la primera caja, con la diferencia de que en la línea 17, se utilizan los valores dados por la función $empacar$ para obtener los valores de $\boldsymbol{y}$ y z en que se ubicará la caja, siempre que se comience con un nuevo plano la primera caja será ubicada en la posición *(fin, 0, 0)* dado que el algoritmo (Pönitz, 2020) busca maximizar el perímetro de contacto de cada caja y en este caso la única forma de hacerlo sería ubicarla en *(fin, 0, 0).*

En la línea 20 se evalúa si la caja pertenece al mismo cliente que la caja anterior, luego en la línea 21 se evalúa si la caja cabe dentro del plano ***yz*** y si se cumplen las restricciones de factibilidad, si esto se cumple se ubica y se sigue con la siguiente caja, si no se cumple, en la línea 24 se comienza una nueva fila o plano ***yz*** donde se ubicará la caja actual y se seguirán ubicando el resto de las cajas. Finalmente, desde el punto 31 al 35, se actualiza el valor de fin, que determinará la ubicación de cada plano de objetos.

Algorithm 2 Touching Perimeter Algorithm

```
1: maxx = [], cajas = [], fin = 0, c1 = 0, c2 = 0, c3 = 0, c4 = 0
2: camion = rect(0, 0, W, H)
3: for i = 1, 2, ..., n do
4:     cajas ← caja_i
5: end for
6: empacador = empaquetador(camion, rotacion = False)
7: for i = 1, 2, ..., n do
8:     if i = 0 then
9:         caja_0 ← cajas
10:        c1, c2, c3, c4 = empacador.empacar(i)
11:        (x_0, y_0, z_0) = (0, 0, 0)
12:        maxx ← l_i
13:    else
14:        if k_i =!k_{i−1} then
15:            empacador = empaquetador(camion, rotacion = False)
16:            caja_0 ← cajas
17:            c1, c2, c3, c4 = empacador.empacar(c)
18:            (x_i, y_i, z_i) = (fin, c2, c3)
19:            maxx ← l_i
20:        else
21:            if c1 then
22:                (x_i, y_i, z_i) = (x_{i−1}, c2, c3)
23:                maxx ← l_i
24:            else if c1 = False then
25:                empacador = empaquetador(camion, rotacion = False)
26:                maxx = [0]
27:                c1, c2, c3, c4 = empacador.empacar(c)
28:                (x_i, y_i, z_i) = (fin, 0, 0)
29:            end if
30:        end if
31:        if x_i + l_i > fin then
32:            maxx ← l_i
33:            fin = x_i + max(maxx)
34:            maxx = maxx − l_i
35:        end if
36:    end if
37: end for
38: maxx = [0]
```

Algoritmo 2: Touching Perimeter Algorithm

4.2 Adaptación de *Harmony Search*

Se utiliza una modificación de la metaheurística *Harmony Search* para realizar una comparación con la metaheurística *Tabu Search*, esta se toma del trabajo desarrollado por Anabalón et al. (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). La metaheurística se basa en la improvisación musical, que mejora en cada práctica acercándose a la melodía perfecta. Funciona de forma similar a *Tabu Search* en el sentido de que mejora una solución inicial factible para el problema construida a partir de una heurística constructiva.

4.3 Adaptación de *Tabú Search*

4.3.1 Diseño del algoritmo

Se aplica una versión modificada de la metaheurística *Tabu Search,* como se observa en el algoritmo 3. La función de este algoritmo es mejorar las soluciones entregadas por la heurística constructiva seleccionada, por lo tanto, para el funcionamiento de este algoritmo se requiere obtener una solución factible. En esta investigación se prueban 3 distintas soluciones iniciales para el uso de la metaheurística.

La primera iteración comienza con la solución inicial seleccionada y genera tantos vecindarios como los definidos en el parámetro $cant_{vecindarios}$. Cada vecindario se crea a partir del movimiento de un cierto número de cajas seleccionadas aleatoriamente definido en el parámetro $cajas_{movidas}$, eligiendo el vecindario que presente menor largo, guardando el número de las cajas cambiadas de lugar dentro de $Listas_{tabu}$, para que no puedan ser movidas en las siguientes iteraciones. Luego, se sigue trabajando con la solución obtenida en la última iteración y se pasa a la siguiente iteración, verificando en cada iteración que no se repitan las cajas ya movidas por cierto número de iteraciones determinadas por $tenencia_{tabu}$. En caso de llegar a un valor de solución menor al mejor valor histórico, se cambia a la solución actual por el nuevo mejor valor histórico sin consultar si dichas cajas han sido movidas en iteraciones recientes. Finalmente, gracias al algoritmo de selección, se consideran las soluciones que presenten un menor largo, así como mejoras en términos de proximidad de las cajas de un mismo cliente, el volumen de la solución y el orden de las cajas según la ruta. A continuación, una explicación detallada del algoritmo.

Cada un cierto número de iteraciones, número definido en el parámetro, se realiza un proceso de intensificación donde se toman las cajas que más movimientos han sufrido, se generan nuevas vecindades a partir del movimiento de estas cajas y en caso de ser mejor que la solución actual, se reemplaza por la mejor vecindad generada en este proceso. La cantidad de cajas seleccionadas en este proceso es la misma que la definida en el parámetro $cajas_{movidas}$.

4.3.2 Funciones *movf* y *movv*

Para la construcción del algoritmo 3, se utilizan 2 funciones utilizadas en el trabajo de Anabalon et al. (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021).

La función *movf* tiene como entrada el número de una caja, la información de todas las cajas y una solución. Esta función mueve la caja ingresada dentro de la solución a un lugar aleatorio buscando dejarla lo más cerca posible de una caja del mismo cliente.

Por otro lado, la función *movv* que tiene como entrada una solución y la información de todas las cajas, disminuye el largo de la solución moviendo las cajas hacia atrás a lugares vacíos siendo una gran ayuda para mejorar cada solución.

4.3.3 Explicación pseudocódigo

Se requiere una solución inicial. En cada iteración del *while* ubicado en la línea 1 a 64 incluyendo al proceso de intensificación se generará una nueva solución, en la línea 2 se crearán la lista $Tabu$ que servirá para almacenar el orden de cajas que presentará cada vecindario y $Lista_{listas}$ que servirá para guardar los movimientos de cajas que cada vecindario sufrió.

En el *for* de la línea 3 a 15 se comenzarán a crear los vecindarios, en la línea 4 se crea una lista $Lista_{temporal}$ donde se guardarán todos los movimientos de cada vecindario para luego guardarlos en $Lista_{listas}$. En la línea 5, para cada vecindario se creará una variable llamada $solucion_{vecindario}$ donde se almacenará una copia de la solución actual para realizar cambios de ubicaciones de cajas de dicho vecindario. En la línea 6 a 12, hay un *for* donde en cada iteración se realizará un cambio en la ubicación de caja, para esto en la línea 8 se selecciona una caja al azar y para que no se repitan los

movimientos o cajas movidas, se utiliza un *while* que sirve para no elegir 2 veces la misma caja. Luego, en la línea 10 se registra la caja en $Lista_{temporal}$, mientras en la línea 11 se efectúa el cambio de ubicación en solución vecina. Al salir del *for*, en la línea 16 se guarda $solucion_{vecindario}$ dentro de la lista $Tabu$. También, se almacena $Lista_{temporal}$ de cada vecindario en $Lista_{listas}$.

Al terminar con el proceso de creación de vecindarios se comienza con el proceso de evaluación. En la línea 16 se crea la lista $Largos_{vecindarios}$, en ella se guardarán los largos calculados de cada solución mediante la función *FO*. Luego, en la línea 20 se guardará el número del vecindario que presente el menor largo y en la línea 23 se evaluará luego de un ajuste con la función *movv*, acá se calcula nuevamente el largo de la solución dado que gracias a la utilización de la función *movv* podría dar un valor menor al almacenado en $Largos_{vecindarios}$. Si esta solución es menor a la mejor solución histórica encontrada hasta ahora, se reemplaza a la solución actual con la del mejor vecindario en la línea 24, se agregan las cajas movidas en la nueva solución a $Listas_{tabu}$ en la línea 25 actualizando los valores tabúes. En la línea 26 a 28, si es que $Listas_{Tabu}$ está llena, se borran los elementos tabúes que sean más antiguos, y se agregan las cajas movidas a $Mem_{largoplazo}$ en la línea 30. Si no ocurre que el vecindario con solución más corta sea mejor que el valor histórico, en la línea 33 se comienza un bucle *while* que verifica que los movimientos de cajas del vecindario con mejores resultados no sean movimientos tabúes, buscando en $Lista_{listas}$ en el *for de las* líneas 35 a 44 que los movimientos del mejor vecindario no sean tabúes, esto se determina con el contador encontrado en la línea 37 que suma uno cada vez que una caja movida del mejor vecindario se encuentre dentro de $Listas_{tabu}$. Si el contador alcanza un número igual al definido en $cajas_{movidas}$ se le cambiará el valor de largo almacenado en $Largos_{vecindarios}$ a 999999 en la línea 39 con el objetivo de dejar a ese vecindario fuera del análisis, luego, al comenzar una nueva iteración de este *while* se prosigue con el siguiente vecindario con menor largo hasta encontrar uno que cumpla con la restricción. En la línea 45 se verifica el caso en que ya todos los vecindarios tengan guardado el valor 999999 en $Largos_{vecindarios}$, que de ser así se mantiene la solución como había quedado en la iteración anterior, pasando a la siguiente iteración.

En el caso en que se elija un vecindario en la línea 51, se actualizan las cajas movidas en $Listas_{tabu}$ y $Mem_{largoplazo}$.

Algorithm 3 Tabu Search

Require: $\textbf{Listas}_{Tabu} = [], \textbf{cajas}_{movidas}, \textbf{Mem}_{largoplazo}, iteracion = 0$

1: **while True do**
2: $\textbf{Tabu} = [], \textbf{Lista}_{listas} = []$
3: **for** $k = 1, 2, \ldots, \textbf{vecindarios}_{tabu}$ **do**
4: $\textbf{Lista}_{temporal} = []$
5: $\textbf{Solucion}_{vecindario} = \textbf{Sol}_{Actual}$
6: **for** $j = 1, 2, \ldots, \textbf{cajas}_{movidas}$ **do**
7: **do**
8: $x = random(\textbf{N})$
9: **while** $(\textbf{x}\ in\ \textbf{Lista}_{temporal})$
10: $\textbf{Lista}_{temporal} \leftarrow \textbf{x}$
11: $movf(\textbf{Solucion}_{vecindario}, x)$
12: **end for**
13: $\textbf{Tabu} \leftarrow \textbf{Solucion}_{vecindario}$
14: $\textbf{Lista}_{Listas} \leftarrow \textbf{Lista}_{temporal}$
15: **end for**
16: $\textbf{Largos}_{vecindarios} = []$
17: **for** $iteration = 1, 2, \ldots, len(\textbf{Lista}_{listas})$ **do**
18: $\textbf{Largos}_{vecindarios} \leftarrow FO(\textbf{Solucion}_{vecindario})$
19: **end for**
20: $n = \textbf{Lista}_{listas}.index(min(\textbf{Largos}_{vecindarios}))$
21: $prueba = Tabu[n]$
22: $movv(prueba)$
23: **if** $FO(prueba) < min(Soluciones)$ **then**
24: $sol.Actual = Tabu[n]$
25: $\textbf{Listas}_{Tabu} = \textbf{Listas}_{Tabu} \leftarrow \textbf{Listas}_{listas}[n]$
26: **if** $len(\textbf{Listas}_{Tabu}) = \textbf{cajas}_{movidas} * \textbf{tenencia}_{tabu}$ **then**
27: $\textbf{Listas}_{Tabu} = \textbf{Listas}_{Tabu}[(\textbf{cajas}_{movidas} - 1) ::]$
28: **end if**
29: **for** $\textbf{i} = 1, 2, \ldots, \textbf{cajas}_{movidas}$ **do**
30: $\textbf{Mem}_{largoplazo}[\textbf{Lista}_{listas}[n][i] - 1] + +$
31: **end for**

```
32:     else
33:         while True do
34:             a = 0, b = 0
35:             for k = 1, 2, ..., Lista_listas[Largos_vecindarios.index(min(Largos_vecindarios))]
  do
36:                 if k in Listas_Tabu then
37:                     b = b + 1
38:                     if b = cajas_movidas then
39:                         Largos_vecindarios[n] = 999999
40:                         a = 1
41:                         break
42:                     end if
43:                 end if
44:             end for
45:             if min(Largos_vecindarios) = 999999 then
46:                 Listas_Tabu = Listas_Tabu[cajas_movidas − 1 ::]
47:                 break
48:             end if
49:             if a = 0 then
50:                 n = Largos_vecindarios.index(min(Largos_vecindarios))
51:                 sol.Actual = Tabu[n]
52:                 movv(sol.Actual)
53:                 Listas_Tabu = Listas_Tabu ← Listas_listas[n]
54:                 if (len(Listas_Tabu)) = cajas_movidas * tenencia_tabu then
55:                     Listas_Tabu = Listas_Tabu[cajas_movidas − 1 ::]
56:                 end if
57:                 for i = 1, 2, ..., cajas_movidas do
58:                     Mem_largoplazo[Lista_listas[n][i] − 1] + +
59:                 end for
60:                 break
61:             end if
62:         end while
63:     end if
64:     iteracion + +
65: end while
```

Algoritmo 3: metaheurística *Tabu Search*

4.3.4 Proceso de intensificación

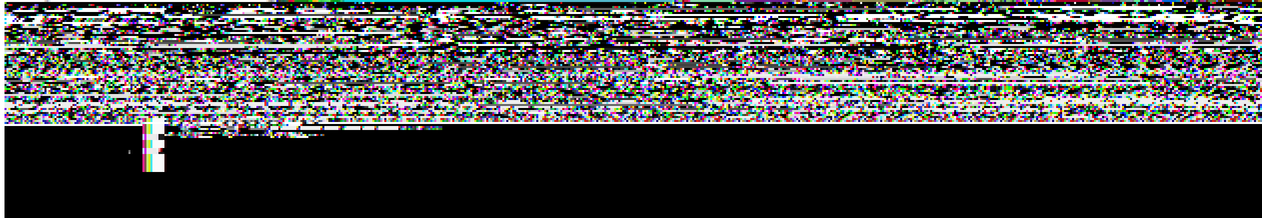
Se agrega a la metaheurística *Tabu Search* un proceso de intensificación que busca minimizar aún más el largo de las soluciones. Para lograr esto se mueven la cantidad de veces definido en parámetro $proceso_{intensificación}$ a lugares aleatorios las cajas que más veces han sido seleccionadas en los procesos de diversificación, esto debido a que se espera que sean estas cajas las que más podrían afectar los resultados de largo de las soluciones, el número de cajas a mover en cada intento está determinado por el parámetro $cajas_{movidas}$.

En primer lugar, se debe definir cuantas iteraciones ocurrirán entre procesos de intensificación en parámetro $Iteraciones_{intensificación}$. Se crea en la línea 2 una lista donde se guardará el número de las cajas que más veces han sido seleccionadas, el número de cajas será determinado por el parámetro $cajas_{movidas}$, en la línea 3 se copia la lista $Mem_{largoplazo}$, que es una lista de tamaño igual al número total de productos, donde cada elemento representará una caja y el número almacenado, las veces que se ha movido dicha caja. En el *for* de la línea 4 a 8 se extraerán los números de las cajas más sensibles, o que más cambios positivos generan en las soluciones y luego en el *for* de la línea 9 a 18 se realizan las pruebas, se hacen tantos intentos como los definidos en $proceso_{intensificación}$, en cada uno se mueven la cantidad de cajas definidas en $cajas_{movidas}$ que más cambios positivos generan en las soluciones, en la línea 15 se verifica en cada una de las pruebas si es que se supera a la solución actual, que siendo así se reemplaza por la nueva solución.

Después de este proceso de mejora en cada iteración, se asegura que la solución sea factible, aplicando las funciones overlap, gravedad y fragilidad y se toma registro cuando la solución mejora a la actual con el algoritmo de selección utilizado en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021).

Algorithm 4 Proceso de intensificación

1: **if iteracion** % **iteraciones**$_{mejora} = 0$ **then**
2: **mas.usadas** = []
3: $lista$ = **Mem**$_{largoplazo}$
4: **for x** in **cajas**$_{movidas}$ **do**
5: $a = lista.index($**max**$(lista))$
6: **mas.usadas** ← a
7: $lista[a] = 0$
8: **end for**
9: **for i** in **proceso**$_{intensificación}$ **do**
10: **Probar** = $sol.Actual$
11: **for j** in **cajas**$_{movidas}$ **do**
12: $movf$(**Probar**)
13: **end for**
14: $movv$(**Probar**)
15: **if** $FO(Probar) < FO(sol.Actual)$ **then**



Algoritmo 4: Proceso de intensificación *Tabu Search*

5. Resultados computacionales

En este capítulo son presentados los resultados obtenidos por el algoritmo para el problema planteado, el cual se implementó en Jupyter Notebook, utilizando Python 3.7. Se usó un servidor con un procesador Intel® Xeon® CPU E3-1270 v6, 3.80 GHz, 62 GB de ram y 30 nodos.

Los datos para el problema fueron obtenidos de una empresa distribuidora de productos en cajas. Se considera un camión con ancho de 150 cm y alto de 215 cm. Para calibrar los parámetros del algoritmo *Tabu Search* y para seleccionar la solución inicial se utilizaron las primeras 9 instancias. También, se realizó una parametrización del tiempo en que debería iterar la metaheurística de esta investigación. Finalmente, con el objetivo de encontrar que algoritmo se desempeña mejor, se comparó el resultado de las 21 instancias del problema entre los 2 algoritmos creados a partir de la combinación de la heurística constructiva seleccionada y ambas metaheurísticas estudiadas para el problema, entre ellas *Harmony Search* desarrollada por Anabalón et al. (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021).

5.1 Selección heurística constructiva

Con el objetivo de encontrar mejores soluciones que cumplan con las restricciones del problema, se resolvió el problema con 3 distintas heurísticas constructivas. En esta parte de los experimentos, se realizó una comparación del desempeño de los 3 algoritmos en las primeras 9 instancias para seleccionar la heurística a la que se le aplicarán las metaheurísticas de mejora.

			Wall Building				*Wall Building* modificado				*Touching Perimeter*			
Instancia	N	n	L(cm)	Vol (m^3)	Pen	Tiempo (s	L(cm)	Vol (m^3)	Pen	Tiempo (s)	L(cm)	Vol (m^3)	Pen	Tiempo (s)
1	3	83	185	4,03	0	0,001	145	5,71	7,38	0,74	220	6,30	0	0,769
2	3	141	378	11,36	0	0,006	366	12,10	0,887	11,22	385	11,03	0	1,309
3	6	681	716	19,75	0	0,013	619	19,65	1,524	5,82	790	22,10	0	6,333
4	7	573	760	19,45	0	0,006	620	19,79	2,475	5,03	784	20,36	0	5,292
5	4	132	280	0,58	0	0,001	180	5,36	2,177	1,12	277	5,48	0	1,25
6	6	222	432	9,15	0	0,003	344	10,23	1,862	1,89	498	10,69	0	1,992
7	5	300	360	8,57	0	0,003	298	9,85	2,77	2,61	390	9,71	0	2,75
8	4	435	958	27,80	0	0,004	910	29,58	0,774	3,74	969	27,21	0	3,937
9	4	103	273	6,19	0	0,001	205	6,19	2,685	0,90	320	8,27	0	0,95
Promedio			482,4	11,88	0	0,004	409,7	13,16	2,50	3,67	515	13,46	0	3

Tabla 1: Heurísticas constructivas primeras 9 instancias

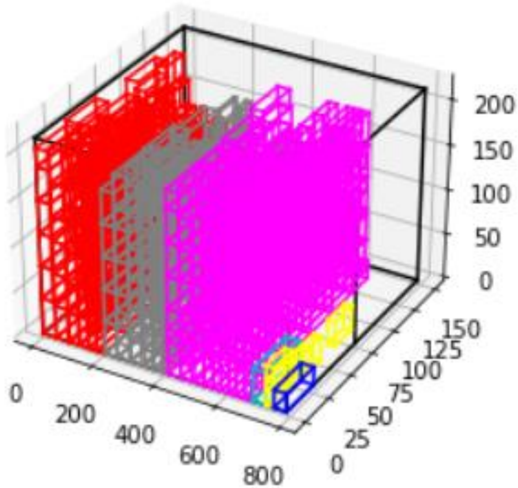


Figura 7: Wall Building instancia 3

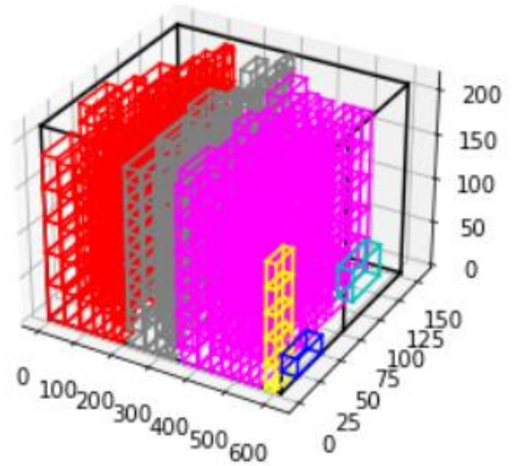


Figura 6: Wall Building modificado instancia 3

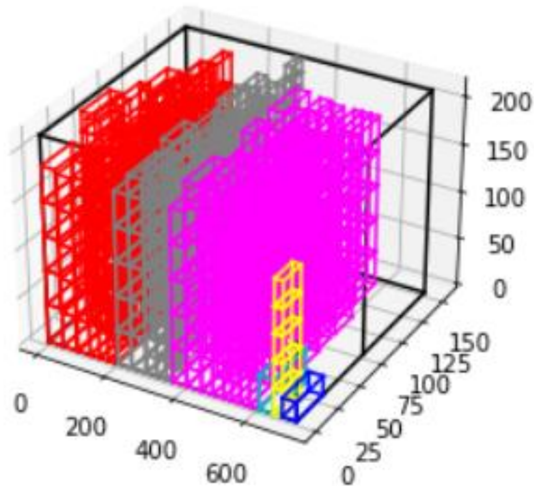


Figura 8: Touching Perimeter instancia 3

En la tabla 1 se presentan las soluciones para las primeras 9 instancias. Para visualizar los resultados se agregó el promedio de los resultados. También, las figuras 6, 7 y 8 muestran los resultados de las 3 heurísticas constructivas para la instancia 3 de manera de visualizar las características de cada una.

Se puede observar que el método que presenta mejores resultados en cuanto al largo de las soluciones es el algoritmo *Wall Building* modificado superando en un 15,1% a su par *Wall Building* y en un 20,4% a la heurística *Touching Perimeter*, sin embargo, es la única heurística que entrega un valor de penalización mayor a 0. Aun así, si bien es la única heurística que presenta penalización, esto no es un problema, dado que dicha penalización no influye necesariamente en la factibilidad entregada por la solución, debido a que al igual que en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) el valor de penalización se calcula con el objetivo de minimizar la cercanía que poseen las cajas de un mismo cliente, al penalizar cada caja de un cliente posterior que se encuentre antes de una caja del cliente anterior, haciendo así que al encontrarse ubicadas cajas de distintos clientes a la misma distancia del origen del camión puede ocurrir que cajas de, por ejemplo, un cliente 1 se encuentren más adelante por el largo propio de las cajas que las de un cliente 2, generando así penalización para las cajas del cliente 2.

Respecto al tiempo, la heurística que más tiempo utiliza es *Wall Building* modificado, que demora en promedio 3,67 segundos por instancia, la segunda que más tiempo utiliza es la heurística *Touching Perimeter* con 2,73 segundos en promedio por instancia, finalmente la más rápida es la heurística *Wall Building* (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) con 0,004 segundos de cómputo en promedio por instancia. Sin embargo, a pesar de que la heurística *Wall Building* modificado tome más tiempo a la larga tiene el potencial de ahorrar más tiempo tiempo a la metaheurística que se utilice al buscar mejores soluciones.

Como conclusión, en este experimento se determina que la heurística constructiva a utilizar para los experimentos finales será el algoritmo *Wall Building* modificado al obtener los valores de largo más bajos por ser un buen piso para comenzar a mejorar mediante el uso de alguna metaheurística.

5.2 Calibración de parámetros para metaheurística *Tabu Search*

Se busca ajustar los parámetros $tenencia_{tabu}$, $cant_{vecindarios}$, $cajas_{movidas}$, $Iteraciones_{intensificación}$ y $proceso_{intensificación}$ probando el algoritmo con las primeras nueve

instancias usadas en estudios anteriores, seleccionando la combinación de parámetros que presente menor largo en las nueve instancias mencionadas, sin dejar de lado los valores de volumen y penalización se escapen demasiado de un rango aceptable. **S**e consideraron 8 horas para este procedimiento y todos los algoritmos poseen la misma semilla de aleatoriedad, por tanto, son comparables.

$vecindarios_{tabu}$	**4**	**4**	**4**	**5**	**5**	**5**	**6**	**6**	**6**
$tenencia_{tabu}$	**5**	**6**	**7**	**5**	**6**	**7**	**5**	**6**	**7**
Largo (cm)	3533	3533	3533	3499	3499	3499	3531	3531	3531
Vol (m^3)	162,24	162,24	162,24	171,05	171,05	171,05	164,55	164,55	164,55
Penalización	256,67	256,66	256,58	319,45	319,45	319,45	303,52	303,52	303,52
Iteraciones (mejor resultado)	6666	6666	6666	6144	6144	5797	3637	3637	3637

Tabla 2: Combinación de parámetros y resultados

En la tabla 2 es posible notar la combinación de valores usados para los parámetros $vecindarios_{tabu}$ y $tenencia_{tabu}$ y sus resultados de largo, volumen, penalización e iteraciones para llegar al mejor resultado. En la primera fila, se puede ver el parámetro $vecindarios_{tabu}$, en la segunda, se puede ver el parámetro $tenencia_{tabu}$, en la tercera, la suma de los mejores valores encontrados por el algoritmo para cada instancia dentro de las 8 horas de cómputo, en la cuarta, la suma de los volúmenes de cada solución, en la quinta la suma de las penalizaciones y, la quinta, el número de iteraciones que tomó cada algoritmo para llegar a la mejor solución. Estos valores fueron escogidos al observarse empíricamente que lograban mejores resultados. Se determina que la combinación de $vecindarios_{tabu}$ = 5 y $tenencia_{tabu}$ = 7 sería la elegida, pues con ella se obtiene un menor largo en las 9 instancias y se logran mejores resultados en un menor tiempo.

Los resultados para cada cambio de parámetros pueden observarse en anexos 1 a 9.

$cajas_{movidas}$	**4**	**6**	**8**	**10**	**12**
Largo (cm)	3499	3560	3538	3537	3548
Vol (m^3)	171,05	132,22	154,60	141,23	175,12

Penalización	319,46	141,82	177,2	260,27	192,14

Tabla 3: parámetro $cajas_{movidas}$ y resultados

En la tabla 3, es posible observar el desempeño del algoritmo según cuantas cajas son seleccionadas para la creación de nuevos vecindarios, se puede notar que con el valor $cajas_{movidas}$ = 4 se obtienen mejores resultados en el largo de las soluciones por lo que será seleccionado para la aplicación de la metaheurística a las demás instancias.

Los resultados para cada cambio de parámetro $cajas_{movidas}$ se pueden observar en anexos 10 a 14.

$\boldsymbol{Iteraciones_{intensificación}}$	**5**	**10**	**15**
Largo (cm)	3536	3499	3505
Vol ($\boldsymbol{m^3}$)	151,38	171,05	201,35
Penalización	274,41	319,46	491,26

Tabla 4: parámetro $Iteraciones_{intensificación}$ y resultados

En la tabla 4, se puede notar el comportamiento del algoritmo cuando cambia el parámetro $Iteraciones_{intensificación}$ que indica cada cuantas iteraciones se aplicará el proceso de intensificación explicado en el ítem 4.3.4, en esta parametrización se deja a $Iteraciones_{intensificación}$ = 10, dado que es con el cual se obtienen mejores resultados de largo.

Los resultados para cada cambio de parámetro $Iteraciones_{intensificación}$ se pueden observar en anexos 15 a 17.

$\boldsymbol{Proceso_{intensificación}}$	**5**	**10**	**15**
Largo (cm)	3499	3523	3535
Vol ($\boldsymbol{m^3}$)	171,05	157,63	203,64
Penalización	319,46	288	203,64

Tabla 5: parámetro $Proceso_{intensificación}$ y resultados

En la tabla 5, se compara el desempeño de la metaheurística *Tabu Search* cuando el parámetro $Proceso_{intensificación}$ cambia de valor, obteniendo con $Proceso_{intensificación}$ = 5 los mejores resultados de largo de las soluciones, por lo tanto, se usará ese número para futuras pruebas.

Los resultados para cada cambio de parámetro $Proceso_{intensificación}$ se pueden observar en anexos 18 a 20.

Instancia	$L\ (cm)$	Volumen (m^3)	Pen	Tiempo (s)	Iteración
1	141	4,83	7,197	16439,69	1782
2	317	10,62	1,592	20514	1782
3	567	32,20	86,652	25691	113
4	614	32,16	66,034	9224,14	56
5	180	4,83	0,703	15864	1729
6	344	10,23	1,862	0	0
7	281	17,04	99,76	6330	99
8	850	52,95	52,97	25075	236
9	205	6,19	2,685	0	0

Tabla 6: Resultados combinación $vecindarios_{tabu}$ = 5, $tenencia_{tabu}$ = 7, $Iteraciones_{intensificación}$ = 10 y $proceso_{intensificación}$ = 5

La tabla 6 muestra detalladamente los valores encontrados con estos parámetros para cada instancia con los parámetros seleccionados. En la primera columna, se indica el número de la instancia. Las siguientes columnas, indican el largo calculado, el volumen ocupado del camión, la penalización que indica la distancia de la última a la primera caja del mismo cliente y, finalmente, el tiempo de cómputo en el cual se encuentra la mejor solución.

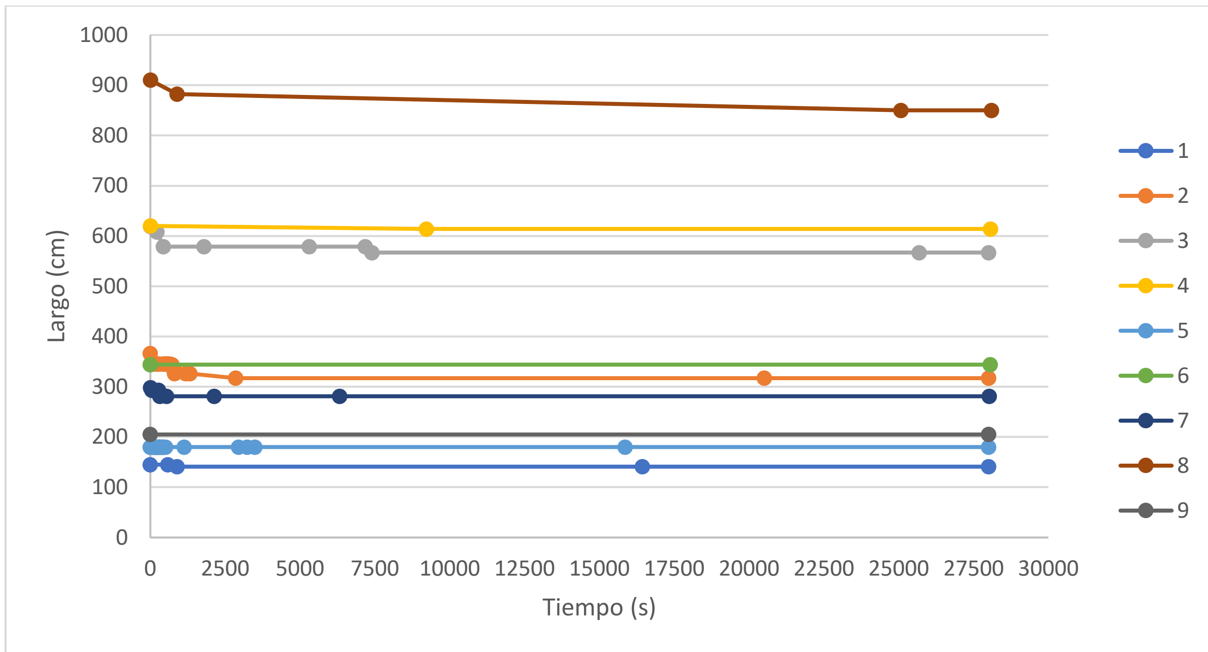


Figura 9: Largo vs Tiempo

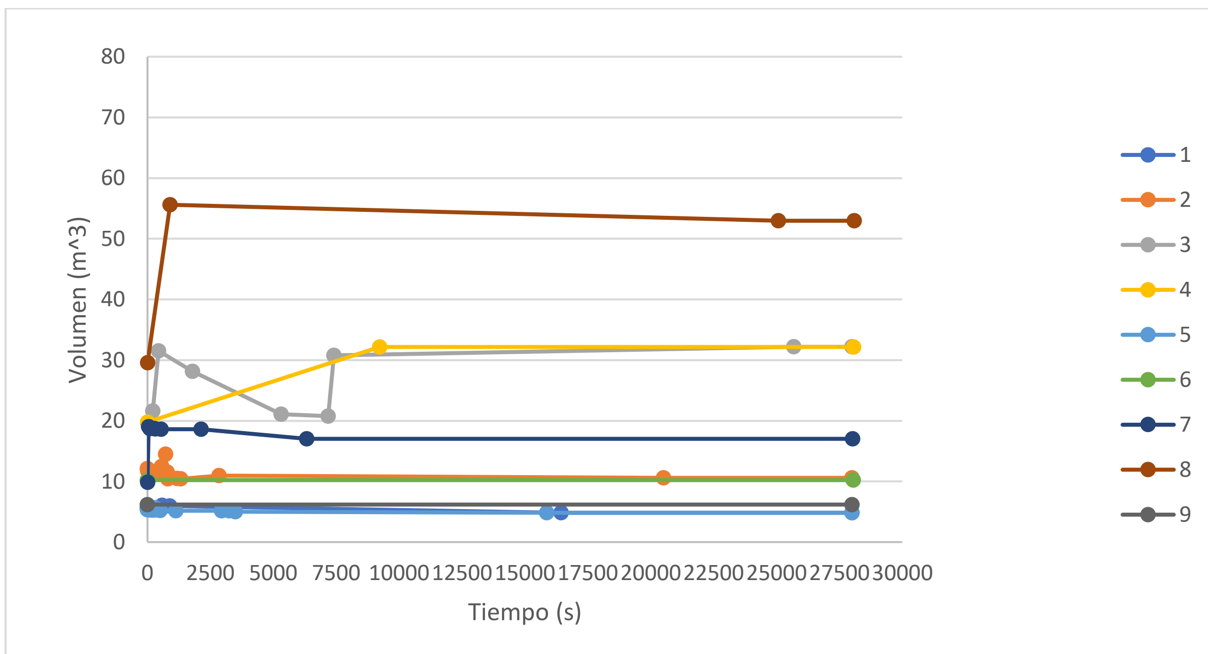


Figura 10: Volumen vs Tiempo

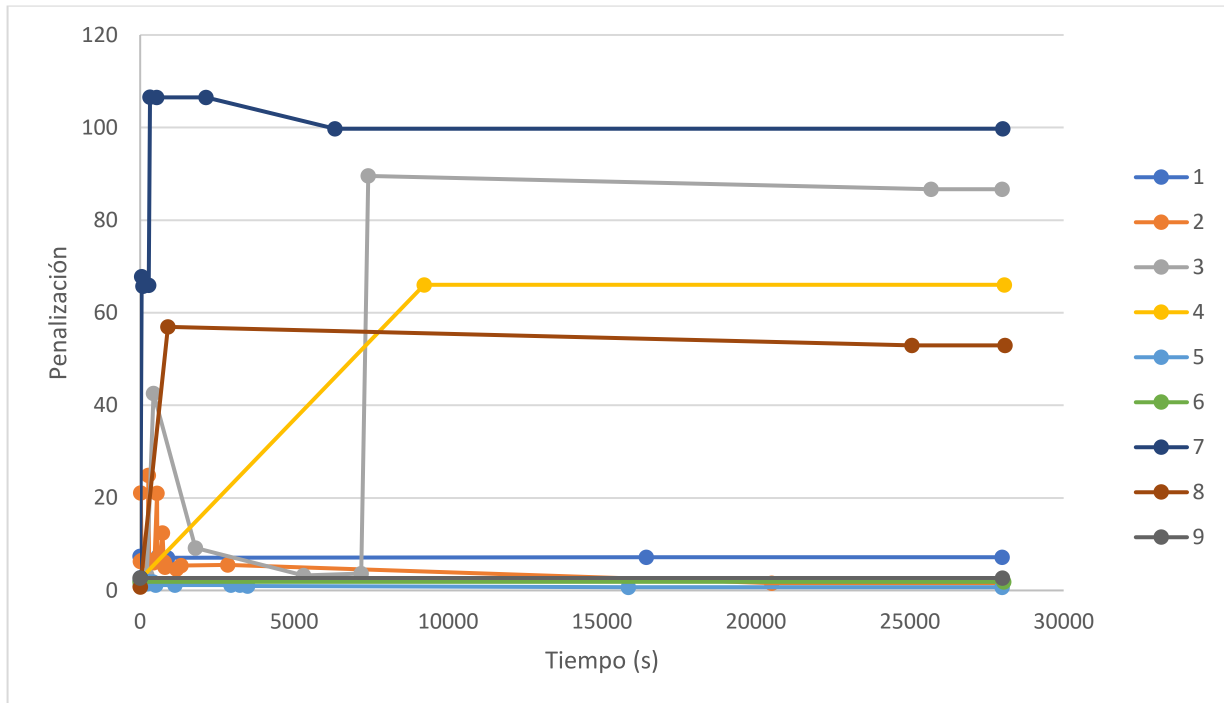


Figura 11: Penalización vs Tiempo

En las Figuras 9, 10 y 11 son mostrados los gráficos de los distintos componentes que evalúan a las soluciones en el tiempo: Figura 9, largo de la solución, Figura 10, volumen de la envolvente de la solución y Figura 11, penalización por posición delante o atrás de las cajas de otro cliente. Es posible notar que las soluciones mejoran respecto al largo durante la totalidad del tiempo, sin embargo, el grueso de la mejora ocurre casi al comienzo, antes de los primeros 1000 segundos de tiempo. Se observa también que en algunas instancias después de los 600 segundos, ocurren cambios poco convenientes en términos de penalización, y tomando en cuenta que en la realidad de una empresa se busca que el tiempo en que se encuentren soluciones sea bajo, se decide que el tiempo de cómputo será de 600 segundos.

5.3 Comparación de los algoritmos

Se compara el desempeño de las metaheurísticas *Harmony Search* y *Tabu Search*, esto puede realizarse debido a que ambas utilizan el mismo servidor y heurística constructiva, para esto se usaron 21 instancias. En la tabla 7, se observan los resultados que cada algoritmo presentó para las instancias del problema, mostrando el desempeño de la heurística constructiva, con los datos de largo y volumen de la solución, junto con su tiempo de cómputo. Por el otro, es posible verificar el largo, el volumen y la penalización que entrega el algoritmo de mejora en su mejor iteración.

A continuación, los resultados de cada modelo para las 21 instancias estudiadas.

			Wall Building modificado				*Tabu Search*						*Harmony Search*					
Instancia	N	n	L	VOL	Pen	Tiempo	L	VOL	Pen	Tiempo	Iter	% mejora	L	VOL	Pen	Tiempo	Iter	% mejora
1	3	83	145	5,71	7,38	0,74	145	5,71	7,38	0	0	0,00	145	5,71	7,38	0	0	0,00
2	3	141	366	12,10	0,887	11,22	345	12,42	6,87	603,19	56	5,74	363	22,67	37,11	373,65	7	0,82
3	6	681	619	19,65	1,524	5,82	579	31,51	42,5	420	3	6,46	619	19,65	1,52	5,8195	0	0,00
4	7	573	620	19,79	2,475	4,87	620	19,79	2,48	0	0	0,00	620	19,79	2,48	0	0	0,00
5	4	132	180	5,36	2,18	1,18	180	5,22	1,16	503,89	56	0,00	180	5,30	1,88	508,3	14	0,00
6	6	222	344	10,23	1,86	1,91	344	10,23	1,86	0	0	0,00	344	10,23	1,86	0	0	0,00
7	5	300	298	9,85	2,77	2,52	281	18,61	107	543,7	12	5,70	298	9,85	2,77	0	0	0,00
8	4	435	910	29,58	0,77	3,81	910	29,58	0,77	0	0	0,00	910	29,58	0,77	0	0	0,00
9	4	103	205	6,19	2,69	0,90	205	6,19	2,69	0	0	0,00	205	6,19	2,69	0	0	0,00
10	1	29	40	1,00	0	0,29	40	1,00	0	0	0	0,00	40	1,00	0,00	0	0	0,00
11	3	84	190	5,82	2,41	0,72	180	6,61	2,54	8,1	3	5,26	180	7,12	12,23	547,25	35	5,26
12	3	160	164	5,47	7,37	1,43	149	5,47	15,2	229,66	18	9,15	164	5,47	7,37	0	0	0,00
13	4	126	180	6,63	8,78	1,16	164	9,30	27	587,44	61	8,89	178	9,69	29,69	253,09	10	1,11
14	5	387	578	17,36	0,81	0,00	578	17,36	0,81	0	0	0,00	578	17,36	0,81	0	0	0,00
15	5	170	207	5,35	0,18	1,45	207	5,35	0,18	0	0	0,00	190	5,71	1,19	284,78	5	8,21
16	4	389	528	16,34	0,49	3,33	528	16,34	0,49	0	0	0,00	528	16,34	0,49	0	0	0,00
17	6	528	614	21,24	2,19	4,51	614	19,83	4,75	252,89	6	0,00	614	21,24	2,19	0	0	0,00
18	3	501	533	16,53	0,81	4,29	533	16,53	0,81	0	0	0,00	533	16,53	0,81	0	0	0,00
19	4	387	595	17,54	0,83	3,28	580	24,31	11,5	671,01	10	2,52	595	17,54	0,83	0	0	0,00
20	4	195	191	6,55	6,7	1,66	176	10,38	59,8	22	2	7,85	186	6,23	32,20	386,22	4	2,62
21	4	125	150	3,89	1,05	1,07	148	4,88	2,07	519,96	64	1,33	150	3,89	1,05	0	0	0,00
Promedio			364,6	11,53	2,58	2,67	357,4	13,17	14,2	207,71	13,86	2,52	362,9	12,24	7,02	112,34	3,57	0,86

Tabla 7: Resultados 21 instancias heurística constructiva y metaheurísticas 600 segundos de cómputo

5.4 Discusión de los resultados

En la tabla 7 se pueden ver los resultados de ambos algoritmos para las 21 instancias para 600 segundos de cómputo, en ella se pueden observar por un lado los resultados del largo, el volumen, tiempo, iteración y la penalización de las soluciones junto con el porcentaje de mejora de tanto *Harmony Search (WBm/H)* y *Tabu Search (WBm/T)* agregando también la heurística *Wall Building* modificado.

Como puede observarse, los mejores resultados respecto al largo de las soluciones son encontrados mediante la metaheurística *Tabu Search* en combinación con la heurística constructiva, mejorando

en un 2,52 % el largo con respecto a la heurística constructiva superando a la metaheurística *Harmony Search* que presenta un 0,86 % de mejora.

Es posible apreciar que es el algoritmo que mayor valor de penalización presenta con respecto a sus pares, esto puede deberse al gran uso de componentes aleatorios al buscar mejorar las soluciones, permitiendo que sea posible encontrar nuevas y más innovadoras soluciones al mismo tiempo que se corre el riesgo de alejar demasiado las cajas de un mismo cliente entre sí. Otra razón, es que la penalización tiende a aumentar a medida que se realizan cambios. Cabe destacar, que en estudios anteriores (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), los valores de penalización son similares a los encontrados en esta investigación.

Es necesario mencionar que hay una gran parte de las instancias que no mejoran durante los 600 segundos, si el tiempo de compilación fuera mayor, la gran mayoría de las instancias mejorarían, como puede observarse en la tabla 6 del proceso de parametrización. También, muchos de estos casos se deben a que la heurística constructiva utilizada logra muy buenos resultados y cuesta un gran tiempo de compilación encontrar mejores soluciones,

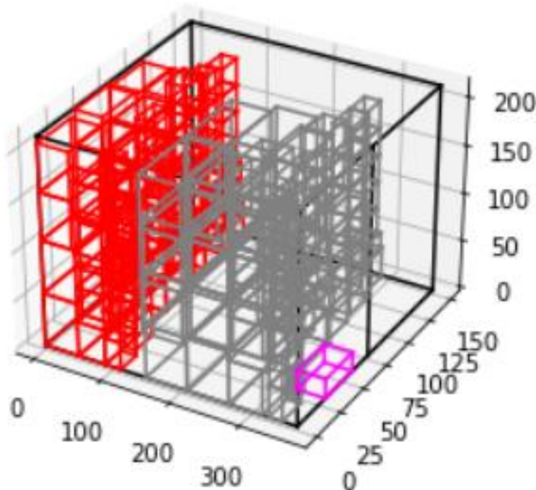


Figura 12: Instancia 2 heurística *Wall Building* modificado

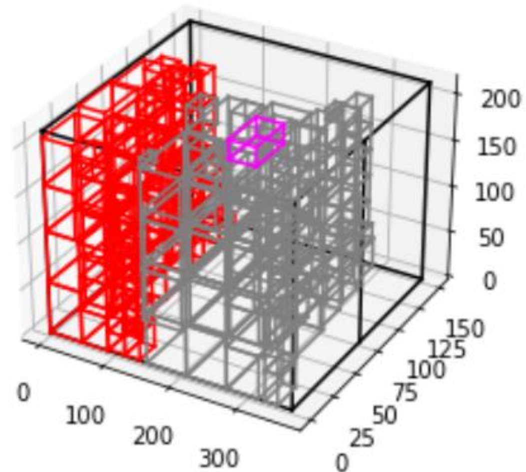


Figura 13: Instancia 2 heurística Wall Building modificado y metaheurística Tabu Search

En la figura 12 se puede observar la solución para la instancia 2 de la heurística *Wall Building* modificado entregando un largo de 366 cm, en la figura 13 se observa a la solución con la heurística

Wall Building modificado. Se puede ver que la penalización presentada en esta instancia se debe a la caja morada ubicada encima de la formación de cajas gris, al dejar a la caja morada detrás de cajas grises, al tiempo que se disminuye a 345 el largo de la solución.

6. Conclusiones

En este trabajo se aborda el problema de carga de camión en tres dimensiones, con cajas heterogéneas, respetando las restricciones tipo LIFO (último en entrar, primero en salir) ubicando las cajas que se bajarán primero más cercanas a la puerta del camión (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Se proponen 2 nuevas heurísticas constructivas y un nuevo algoritmo de mejora basado en la metaheurística *Tabu Search.* Debido a que no hay muchas investigaciones resolviendo este problema en específico, no fue posible encontrar algoritmos de resolución que usaran *Tabu Search* para el *Truck Loading Problem*, por ende, se generó una metaheurística basada en el algoritmo original trabajado por (Glover, Taillard, & Taillard, 1993).

Estos algoritmos fueron aplicados en el IDE jupyter-notebook 6.3.0 en el lenguaje Python 3.7 y probados en las instancias y con las funciones utilizadas en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021), funciones que acercan las soluciones al cumplimiento de las restricciones del problema. Se compararon tanto el largo de las soluciones, como el volumen y penalización por la separación de las cajas de un mismo cliente.

En primer lugar, se compararon las heurísticas constructivas entre sí, seleccionando la que mejores resultados de largo entregó, la cual fue la heurística constructiva *Wall Building* modificada, ésta superó en un 15,1 % a la heurística *Wall Building* y en un 20,4% a la heurística *Touching Perimeter* en el largo de las soluciones para las primeras 9 instancias tomando 3,67 segundos en promedio para cada instancia. Después, se realizó un proceso de parametrización con la metaheurística *Tabu Search* utilizando la heurística constructiva *Wall Building* modificado. Para parametrizar el tiempo del algoritmo *Tabu Search*, se realizó un estudio de la frecuencia con que cambiaban las soluciones en las primeras 9 instancias, determinando que 600 segundos serían suficientes para iterar dicho algoritmo. Finalmente, se compararon ambas metaheurísticas, *Harmony Search,* desarrollado en (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021) y, la metaheurística *Tabu Search,* desarrollada en esta investigación, ambas usando como heurística constructiva a *Wall Building* modificado. Este proceso entregó resultados favorables para la metaheurística *Tabu Search* en cuanto al largo de las soluciones, mejorando en un 2,52% en el largo de las soluciones con respecto a la solución inicial versus un 0,86% entregado por *Harmony Search*. para las 21 instancias en 600 segundos de cómputo, por otro lado,

Cabe destacar, que ya de por sí la heurística constructiva *Wall Building* con modificación presenta buenos resultados, haciendo importante que se considere esta heurística como base para complementar con otro algoritmo.

Como trabajo futuro, podría considerarse trabajar las metaheurísticas desarrolladas buscando disminuir la penalización, quizás agregando nuevas funciones o regulando el funcionamiento de las metaheurísticas. Otra cosa que podría realizarse en torno a la metaheurística *Tabu Search,* podría ser desarrollar otros métodos de generación de vecindarios, donde en lugar de mover una caja, pudieran intercambiarse posiciones de cajas mediante funciones que lo permitan. También, podría usarse la heurística y estructura del algoritmo *Touching Perimeter* para desarrollar algoritmos de mejora que usen la lógica de buscar un área de contacto mayor en partes de su proceso. Así como agregar nuevas características al problema, como rotación o estabilidad de la carga.

7. Glosario

Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de algún tipo de problema

Cajas heterogéneas: Se refiere a que las cajas del problema pueden presentar diferencias en cuanto a tamaño y peso haciendo el problema más complejo

Heurística: Arte de inventar por parte de un ser humano, con la intención de crear estrategias, métodos, criterios, que ayuden a solucionar problemas a través del ingenio, la creatividad, pensamiento abstracto

Heurística Constructiva: Clasificada dentro del grupo de heurísticas, son conocidas por la rapidez en encontrar soluciones de buena calidad para problemas de localización, se utilizan a menudo como solución inicial para algoritmos de mejora metaheurísticos

LIFO: "*Last in First Out*" o "último en entrar, primero en salir", método usado en logística para ordenar productos según fecha de entrega o caducidad, ordenando los últimos productos en entregar al principio del espacio de carga, ayudando a ahorrar tiempo y recursos

Logística: Conjunto de los medios necesarios para llevar a cabo un fin determinado de un proceso complicado

Metaheurística: Método heurístico para resolver un problema computacional general, a diferencia de las heurísticas normales, esta presenta métodos más complejos de resolución y son utilizados para resolver problemas de optimización combinatoria.

Pseudocódigo: El pseudocódigo es una forma de escribir los pasos que va a realizar un programa de la forma más cercana al lenguaje de programación que se utilizará posteriormente. Es como un falso lenguaje, lenguaje humano e idioma del interlocutor.

Tenencia tabú: Intervalo de tiempo o iteraciones durante las cuales un atributo o movimiento permanece prohibido para el algoritmo tabú.

Vecindario: Consiste en una posibilidad de transformación para una solución obtenida, mediante un paso o una cantidad de pasos.

8. Referencias

Ailian, W., Baisong, C., Jun, Z., & Liangfeng, L. (2010). Three-dimensional Packing by a Tabu Search Algorithm in Military Airlif Loading. *IEEE 2010 International Conference on Optoelectronics and Image Processing*. doi:10.1109/icoip.2010.166

Alonso, M. T., Alvarez-Valdes, R., Parreño, F., & Tamarit, J. M. (2016). Algorithms for Pallet Building and Truck Loading in an Interdepot Transportation Problem. *Mathematical Problems in Engineering*.

Anabalón Romero, N., Barros Vásquez, M., & Medina Durán, R. (2021). Modified Harmony Search for a Truck Loading Problem Application. *IEEE LATIN AMERICA TRANSACTIONS, 14*(8).

Araya, I., & Riff, M. -C. (2014). A beam search approach to the container loading problem. *Computers & Operations Research*, 100-107.

Batista, B. M., & Glover, F. (2006). *Introducción a la Búsqueda Tabú.* Obtenido de http://leeds-faculty.colorado.edu/glover/fred%20pubs/329%20-%20Introduccion%20a%20la%20Busqueda%20Tabu%20TS_Spanish%20w%20Belen (11-9-06).pdf

Bortfeldt, A., & Mack, D. (2007). A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research, 183*, 1267-1279.

Bortfeldt, A., & Wäscher, G. (2013). Constraints in container loading – A state-of-the-art review. *European Journal of Operational Research*, 1–20. doi:10.1016/j.ejor.2012.12.006

Bortfeldt, A., Gehring, H., & Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing, 29*, 641-662.

Chen, C. S., Lee, S. M., & Shen, Q. S. (1995). An analytical model of the container Loading problem. *European Journal of Operationa Research*, 68-76.

Costa, M. d., & Captivo, M. E. (2014). Weight distribution in container loading: a case study. *International Transactions in Operational Research, 23*, 239-263 . doi: 10.1111/itor.12145

Crainic, T. G., Perboli, G., & Tadei, R. (2009). A Two-Level Tabu Search for the Three-dimensional Bin Packing Problem. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*.

do Nascimiento, O. X., de Queiroz, T., & Junqueira, L. (2021). Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm. *Computers & Operations Research, 128*(105186).

Do Nascimiento, O. X., Queiroz, T. A., & Junqueira, L. (2020). Practical Constraints in the Container Loading Problem:. *Computers and Operations Research*. doi:10.1016/j.cor.2020.105186

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

Fekete, S. P., & Schepers, J. (2003). A Combinatorial Characterization of Higher-Dimensional Orthogonal Packing. *Mathematics of Operations Research, 29*(2), 353-368.

Fekete, S. P., Schepers, J., & van der Veen, J. C. (2007). An exact Algorithm for Higher-Dimensional Orthogonal Packing. *Operations Research, 55*(3), 569-587.

Geem, Z. W., & Kim, J. H. (2001). A New Heuristic Optimization Algorithm: Harmony Search. *Simulation Councils Inc.*, 60-68.

Gehring, H. (1997). A genetic algorithm for solving the container Loading problem. *International Transactions in Operational Research*, 401-418.

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A Tabu Search Algorithm for a Routing and Container Loading Problem. *Transportation Science, 40*(3), 342-350.

Glover, F., Taillard, E., & Taillard, E. (1993). A user's guide to tabu search. *Annals of Operations Research* (41), 1–28 .

Hamdi-Dhaoui, K., Labadie, N., & Yalaoui, A. (2012). Algorithms For The Two Dimensional Bin Packing Problem With Partial Conflicts. *RAIRO Operations Research*.

Iori, M., Martello, S., & Monaci, M. (2003). Metaheuristic Algorithms For The Strip Packing Problem. *Optimization and Industry: New Frontiers. Applied Optimization, 78*. doi:https://doi.org/10.1007/978-1-4613-0233-9_7

Junqueira, L., Morabito, R., & Sato Yamashita, D. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations research, 39*(1), 74-85.

Lim, A., & Zhang, X. (2005). The Container Loading Problem. *Proceedings of the 2005 AM Symposiun on Applied Computing*.

Liu, J., Yue, Y., Dong, Z., & Keech, M. (2011). A novel hybrid tabu search approach to container loading. *Computers & Operations Research, 28*(4), 797-807.

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). doi:https://doi.org/10.1287/ijoc.11.4.345

Lorie, J., & Savage, L. (1955). THREE PROBLEMS IN RATIONING CAPITAL. *The Journal of Business, 28*(4).

Martello, S., & Vigo, D. (1997). Exact Solution of the Two-Dimensional Finite Bin Packing Problem. *Management Science, 44*(3), 388-399.

Martello, S., Pisinger, D., & Vigo, D. (1997). The Three-Dimensional Bin Packing Problem. *Technical Report DEIS*.

Morillo, D., Moreno, L., & Díaz, J. (2014). Metodologías Analíticas y Huerísticas para la solución de Programación de Tareas con Recursos Restringidos (RCPSP): una revisión, Parte 1. *ing. cienc., 10*(19), 247-271.

Novas, J. M., Ramello, J. I., & Rodríquez, M. A. (2020). Generalized disjunctive programming models for the truck loading problem: A case study from the non-alcoholic beverages industry. *Transportation Research Part E*(101971), 1366-5545. doi:https://doi.org/10.1016/j.tre.2020.101971

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research, 141*(2), 382-392.

Pönitz, T. (13 de Agosto de 2020). *Github*. Obtenido de https://github.com/tasptz/py-touchingperimeter

Respen, J., & Zufferey, N. (2016). Metaheuristics for truck loading in the car production industry. *International Transactions in operational research*, 277-301. doi:10.1111/itor.12306

Santana, J. B., Campos Rodriguez, C., García López, F. C., García Torres, M., Melián Batista, B., Moreno Pérez, J. A., & Moreno Vega, J. M. (2004). Metaheurísticas: una revisión actualizada. *La Laguna, 02*.

Stepanenko, S. (2008). Global Optimization Methods based on Tabu Search.

Viegas, J. L., Vieira, S. M., Henriques, E. M., & Sousa, J. M. (2015). A Tabu Search Algorithm for the 3D Bin Packing Problema in the Steel Industry. *Proceedings of the 11th Portuguese Conference on Automatic Control*, 355-364.

Wang, Z., Lang, M., Zhou, X., Przybyla, J., & Sun, Y. (2017). A Tabu Search Algorithm for Loading Containers on Double-Stack Cars. *Advances in Intelligent Systems and Interactive*, 49-56.

Wäscher, G., HauBner , H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 1109-1130.

Wisniewski, M. A., Ritt, M., & Buriol, L. S. (2011). A Tabu Search Algorithm for the Capacitated Vehicle Routing Problem with Three-Dimensional Loading Constraints. *XLIII Simpósio Brasileiro de PESQUISA OPERACIONAL* .

Yamashita, D. S., & Morabito, R. (2015). Uma nota sobre modelagem matemática de carregamento de caixas dentro de contêineres com considerações de estabilidade da carga. *Production, 25*(1), 113-124. doi:http://dx.doi.org/10.1590/S0103-65132013005000041

Yang, X.-S. (2009). Harmony Search as a Metaheuristic Algorithm. *Springer-Verlag Berlin Heidelberg 2009*, 1-14.

Zhu, W., Oon, W. -C., & Lim, A. (2012). The six elements to block-building approaches for the single container loading problem. *Applied Intelligence*, 431-445.

UNIVERSIDAD DE CONCEPCION – FACULTAD DE INGENIERÍA
RESUMEN DE MEMORIA DE TITULO

Departamento : Departamento de Ingeniería Industrial
Carrera : Ingeniería Civil Industrial
Nombre del memorista : Mauricio Ignacio Ulloa Leyton
Título de la memoria : Aplicación de metaheurística *Tabu Search* para el problema de carga de camión
Fecha de la presentación oral:

Profesora Guía : Rosa Medina Durán
Profesor(es) Revisor(es) : Lorena Pradenas
Concepto : Logística
Calificación :

Resumen

El problema de carga de camión consiste en ubicar n cajas dentro de un camión, las cuales no pueden ser rotadas y deben ubicarse de forma paralela a las paredes del camión respetando la regla de ubicar las cajas de un mismo clientes juntas y, respetando el orden de entrega, así como también, se busca proteger la integridad de las cajas añadiendo restricciones de fragilidad al problema.

En el presente trabajo se realiza un estudio de 2 algoritmos que resuelven el problema de carga de camión, diseñados a partir de la implementación de tres heurísticas constructivas que dan una solución rápida y factible para el problema, y dos algoritmos metaheurísticos que mejoran las soluciones entregadas por dichas heurísticas constructivas.

Con el objetivo de evaluar el desempeño de los algoritmos, se utilizan los datos y rutas usados en Anabalón et al. (2021) (Anabalón Romero, Barros Vásquez, & Medina Durán, 2021). Se consideró un camión con un ancho de 150 cm, un alto de 215 cm, largo infinito y 21 instancias diferentes. Los resultados muestran que algunos algoritmos y heurísticas superan las soluciones de investigaciones anteriores al compararlas entre sí, obteniendo mejores resultados con la heurística constructiva *Wall Building* modificado y con la metaheurística *Tabu Search* desarrollada en esta investigación.

9. Anexos

4 vec y 5 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	6,41	7,2	181,97	36
2	317	10,86	5,377	20529	1678
3	567	37,30	132,554	22854	103
4	610	37,40	31,89	13262	77
5	168	5,30	3,04	14988	1675
6	334	14,43	16,66	26988	522
7	281	13,17	56,39	11609	209
8	910	29,58	0,774	0	0
9	205	7,78	2,78	22749	2366

Anexo 1: Experimentos vecindarios

4 vec y 6 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	6,41	7,2	175,565	36
2	317	10,86	5,377	19229	1678
3	567	37,30	132,55	22628	103
4	610	37,40	31,89	28075	77
5	168	5,30	3,04	14961	1675
6	334	14,43	16,66	27554	522
7	281	13,17	56,39	11341	209
8	910	29,58	0,774	0	0
9	205	7,78	3	23.010	2366

Anexo 2: Experimento 4 vecindarios y 6 tenencia

4 vec y 7 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	6,41	7,2	173,636	36
2	317	10,86	5,377	18798,23	1678
3	567	37,30	132,56	22545	103
4	610	37,40	31,9	28112	77
5	168	5,30	3,04	15163	1675
6	334	14,43	16,66	26786	522
7	281	13,17	56,29	11438	209
8	910	29,58	0,774	0	0
9	205	7,78	3	22.891	2366

Anexo 3: Experimento 4 vecindarios y 7 tenencia

5 vec y 5 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	4,83	7,197	15972	2129
2	317	10,62	1,59	20464	1782
3	567	32,20	86,651	25591	113
4	614	32,16	66,034	9265	56
5	180	4,83	0,703	15852	1729
6	344	10,23	1,862	0	0
7	281	17,04	99,762	5999	99
8	850	52,95	52,97	26297	236
9	205	6,19	2,685	0	0

Anexo 4: Experimento 5 vecindarios y 5 tenencia

5 vec y 6 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	4,83	7,197	16381	2129
2	317	10,62	1,59	20845,89	1782
3	567	32,20	86,652	25225	113
4	614	32,16	66,034	28161	56
5	180	4,83	0,703	15694	1729
6	344	10,23	1,862	0	0
7	281	17,04	99,76	10870	99
8	850	52,95	52,967	24623	236
9	205	6,19	2,69	0	0

Anexo 5: Experimento 5 vecindarios y 6 tenencia

5 vec y 7 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	4,83	7,197	16439,69	1782
2	317	10,62	1,592	20514	1782
3	567	32,20	86,652	25691	113
4	614	32,16	66,034	9224,14	56
5	180	4,83	0,703	15864	1729
6	344	10,23	1,862	0	0
7	281	17,04	99,76	6330	99
8	850	52,95	52,97	25075	236
9	205	6,19	2,685	0	0

Anexo 6: Experimento 5 vecindarios y 7 tenencia

6 vec y 5 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	5,92	7,09	427,69	87
2	317	10,14	4,89	12807	1022
3	567	33,76	124,56	19802	88
4	620	19,79	2,48	0	0
5	180	5,00	0,98	5802	625
6	344	10,23	1,86	0	0
7	281	17,00	99,09	6401	125
8	876	56,95	60,71	26136	244
9	205	5,77	1,87	14281	1446

Anexo 7: Experimento 6 vecindarios y 5 tenencia

6 vec y 6 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	5,92	7,0862	433,66	87
2	317	10,14	4,887	11855,04	1022
3	567	33,76	124,56	19230	88
4	620	19,79	2,475	0	0
5	180	5,00	0,98	5625	625
6	344	10,23	1,862	0	0
7	281	17,00	99,09	6658	125
8	876	56,95	60,71	25532	244
9	205	5,77	1,87	13919	1446

Anexo 8: Experimento 6 vecindarios y 6 tenencia

6 vec y 7 ten

	L	VOL	Pen	Tiempo	Iteración
1	141	5,92	7,086	414,99	87
2	317	10,14	4,8866	11662	1022
3	567	33,76	124,565	19976	88
4	620	19,79	2,475	0	0
5	180	5,00	0,98	6136	625
6	344	10,23	1,862	0	0
7	281	17,00	99,09	6360	125
8	876	56,95	60,705	26180	244
9	205	5,77	1,87	15142	1446

Anexo 9: Experimento 6 vecindarios y 7 tenencia

	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	4,83	7,197	16439,7	1782
2	R08-1	317	10,62	1,592	20514	1782
3	R01	567	32,20	86,652	25691	113
4	R20	614	32,16	66,034	9224,14	56
5	R67	180	4,83	0,703	15864	1729
6	R80	344	10,23	1,862	0	0
7	R22	281	17,04	99,76	6330	99
8	R30-2	850	52,95	52,97	25075	236
9	R72-1	205	6,19	2,685	0	0

Anexo 11: cajas_movidas = 4

	cm=6	L	Vol	Pen	Tiempo	Iteración
1	R08-2	141	6,39	7,21	10856,2	1200
2	R08-1	317	10,14	4,87	17403,4	1245
3	R01	607	19,5	2,37	436,08	3
4	R20	620	19,79	2,47	0	0
5	R67	165	5,47	4,66	24129,1	1707
6	R80	334	16,77	10,18	18307,6	354
7	R22	281	18,61	106,31	5078	95
8	R30-2	890	29,36	1,06	1320,86	12
9	R72-1	205	6,19	2,69	0	0

Anexo 10: cajas_movidas = 6

	cm=8	L	Vol	Pen	Tiempo	Iteración
1	R08-2	139	11,72	32,86	4024	497
2	R08-1	317	10,78	5,32	14657	1140
3	R01	569	38,58	70,62	14258	60
4	R20	620	19,79	2,47	0	0
5	R67	170	6,01	3,22	27015	1490
6	R80	344	10,23	1,86	0	0
7	R22	284	10,05	37,25	26176	276
8	R30-2	890	41,25	20,87	13487	104
9	R72-1	205	6,19	2,69	0	0

Anexo 12: cajas_movidas = 8

	cm=10	L	Vol	Pen	Tiempo	Iteración
1	R08-2	141	9,77	18,19	20299,76	2275
2	R08-1	317	10,87	23,99	20635,92	1534
3	R01	574	32,53	89,01	7743,03	33
4	R20	620	19,79	1,23	0	0
5	R67	165	5,48	21,74	17012,21	933
6	R80	344	10,23	1,86	0	0
7	R22	281	17,19	100,45	5684,06	91
8	R30-2	890	29,31	1,16	16382,52	121
9	R72-1	205	6,06	2,64	5364,43	492

Anexo 13: cajas_movidas = 10

	cm=12	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	6,33	7,01	18134,7	1992
2	R08-1	317	10,63	5,12	7125,63	510
3	R01	579	45,98	54,33	14014,7	56
4	R20	620	19,79	2,48	0	0
5	R67	168	5,56	5,25	5062,18	397
6	R80	344	10,23	1,86	0	0
7	R22	284	13,15	49,6	4534,68	70
8	R30-2	890	57,26	63,8	15153	103
9	R72-1	205	6,19	2,69	0	0

Anexo 14: cajas_movidas = 12

p4=5	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	6,42	7,29	25041,7	2919
2	R08-1	317	10,84	5,46	11113,6	942
3	R01	567	34,12	133,29	5401,79	25
4	R20	620	19,79	2,48	0	0
5	R67	180	5,22	1,11	22161,2	2434
6	R80	344	10,23	1,862	0	0
7	R22	281	17,09	99,78	487,36	10
8	R30-2	881	41,47	20,45	24005	224
9	R72-1	205	6,19	2,69	0	0

Anexo 15: Iteraciones_intensificación = 5

p4=10	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	4,83	7,197	16439,69	1782
2	R08-1	317	10,62	1,592	20514	1782
3	R01	567	32,20	86,652	25691	113
4	R20	614	32,16	66,034	9224,14	56
5	R67	180	4,83	0,703	15864	1729
6	R80	344	10,23	1,862	0	0
7	R22	281	17,04	99,76	6330	99
8	R30-2	850	52,95	52,97	25075	236
9	R72-1	205	6,19	2,685	0	0

Anexo 16: Iteraciones_intensificación = 10

p4=15	cm = 4	L	VOL		Pen	Tiempo	Iteración
1	R08-2	141	8,02		11,34	20208	2624
2	R08-1	317	10,14		5,23	5291	475
3	R01	567	28,87		121,03	16191	71
4	R20	606	44,74		142,13	12214	77
5	R67	168	5,58		2,93	24255	1829
6	R80	344	10,23		1,862	0	0
7	R22	281	17,04		99,88	7649	153
8	R30-2	876	70,53		104,17	13889	136
9	R72-1	205	6,19		2,685	0	0

Anexo 17: Iteraciones_intensificación = 15

pi = 5	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	4,83	7,197	16439,7	1782
2	R08-1	317	10,62	1,592	20514	1782
3	R01	567	32,20	86,652	25691	113
4	R20	614	32,16	66,034	9224,14	56
5	R67	180	4,83	0,703	15864	1729
6	R80	344	10,23	1,862	0	0
7	R22	281	17,04	99,76	6330	99
8	R30-2	850	52,95	52,97	25075	236
9	R72-1	205	6,19	2,685	0	0

Anexo 18: Proceso_intensificación = 5

pi = 10	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	141	5,96	7,32	16352	1907
2	R08-1	315	10,81	24,67	24931,5	2172
3	R01	567	26,29	90,82	23722,5	102
4	R20	620	19,79	2,48	0	0
5	R67	168	5,32	1,91	7916,83	692
6	R80	344	10,23	1,862	0	0
7	R22	281	17,04	99,36	12037,4	212
8	R30-2	882	55,99	56,94	934,38	10
9	R72-1	205	6,19	2,685	0	0

Anexo 19: Proceso_intensificación = 10

pi = 15	cm = 4	L	VOL	Pen	Tiempo	Iteración
1	R08-2	140	12,41	31,01	4739,04	542
2	R08-1	315	10,55	5,24	2557,1	233
3	R01	579	19,81	2,97	4196,32	20
4	R20	620	19,79	2,45	0	0
5	R67	170	5,56	3,16	22830,1	1900
6	R80	344	10,23	1,862	0	0
7	R22	281	17,04	99,36	12387,53	212
8	R30-2	881	55,48	54,9	20329,12	187
9	R72-1	205	6,19	2,685	0	0

Anexo 20: Proceso_intensificación = 15