UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA UDELAR

Métodos de resolución eficientes para problemas de producción complejos

Aplicación a un problema de ruteo de vehículos

Felipe Bocca
Francisco Ferriolo

Proyecto de grado presentado a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero de Producción.

Tutores

Pedro Piñeyro

Tribunal
Jimena Ferreira
Juan Machín
Pedro Piñeyro

Montevideo, Uruguay

Octubre de 2025

Agradecimientos

En primer lugar, queremos expresar nuestro agradecimiento al tutor del proyecto Pedro Piñeyro por su compromiso con nosotros y el proyecto. Agradecer por compartir su conocimiento, sus consejos, correcciones, su buena disposición y aliento hacia nosotros.

A las diferentes áreas del Parque Tecnológico Industrial del Cerro por brindar información relevante y su buena disposición. A los compañeros de la Cooperativa de Trabajo La Paloma, Pepe y Micaela, por compartir su experiencia diaria en los recorridos y por ayudarnos a comprender la importancia del problema que buscamos abordar.

Por último, agradecer a nuestras familias y amigos por ser un pilar clave para nuestra motivación y confianza a lo largo de este proceso.

Resumen

En este informe se presenta el trabajo realizado sobre la aplicación de métodos heurísticos en problemas de programación lineal entera mixta. Se dió inicio al trabajo relevando y analizando la literatura relacionada, consultando más de 80 documentos sobre métodos heurísticos para la resolución de este tipo de problemas.

En la elaboración del estado del arte se prestó principal interés en el funcionamiento de las heurísticas, además de la complejidad de los problemas, la calidad de los resultados y los tiempos de cómputo de los diferentes métodos. Se buscó la elección de métodos con buenos resultados y sencillos de implementar. Para poder evaluar los métodos seleccionados se decidió modelar un problema de ruteo de vehículos multi período. Específicamente, se tomó un problema real en donde se debe recoger los residuos valorizables generados en un mes por 54 empresas dentro del Parque Tecnológico Industrial del Cerro (PTIC), buscando minimizar la cantidad de distancia recorrida.

Para la implementación del problema y de los métodos heurísticos propuestos se utilizó el lenguaje de modelado matemático GMPL, en conjunto con el paquete de optimización GLPK. Esta herramienta permitió aprovechar diversas funcionalidades para la formulación y resolución de problemas de optimización, tales como la definición de variables, restricciones y funciones objetivo, así como la ejecución de procedimientos iterativos personalizados. Además, con el objetivo de analizar el comportamiento de los métodos heurísticos, se incorporaron nuevas instancias mediante el uso de una biblioteca de casos de prueba. Estas instancias, de diferentes tamaños y características, permitieron una evaluación más amplia del desempeño de las heurísticas.

Las heurísticas seleccionadas se dividen en dos objetivos, un método de búsqueda de solución inicial, *Feasibility Pump (FP)*, y dos heurísticas de mejora de la solución, *Objective Scaling Ensemble Approach (OSEA)* y *Local Branching (LB).* Se crearon dos nuevos métodos de resolución heurísticos a través de la concatenación de ambos métodos, generando *OSEA+LB* y *LB+OSEA* como dos nuevos métodos de mejora. De los resultados obtenidos se determina que *OSEA+LB* es el método heurístico con mejores resultados, superando en algunas instancias a métodos de resolución GLPK. Para el problema de aplicación fue posible disminuir el valor obtenido de la función objetivo en la mitad de tiempo de cómputo comparado con la resolución a través de GLPK. Los dos métodos de mejora logran mejorar el resultado inicial y, en particular, la fortaleza de *OSEA* es la disminución en los tiempos de cómputo mientras que para *LB* es la calidad de la solución en términos de resultado obtenido. Además, se comparte la mejor solución obtenida para el problema de ruteo de vehículos del PTIC presentando un mapa con los recorridos propuestos.

Se considera que el objetivo del proyecto fue alcanzado, logrando la elaboración de un documento estado del arte para los métodos heurísticos. Además, se logró la correcta implementación de tres heurísticas para la resolución de un problema real, alcanzando el mejor resultado mediante uno de estos métodos.

Palabras claves: Programación matemática, Mixed Integer Linear Programming, Vehicle Routing Problem, Heurísticas MIP, Optimización, Feasibility Pump, Local Branching, Objective Scaling Ensemble Approach.

Índice

Índice tablas

Índice figuras

Siglas y acrónimos

- AMPL: A Mathematical Programming Language
- ANSI C: Instituto Nacional Estadounidense de Estándares para el lenguaje de programación C
- API: Programación de Aplicaciones
- B&B: Branch & Bound (Ramificación y Acotación)
- CMPVRP: Problema de Ruteo de Vehículos con Capacidad y Multi Periodo.
- CVRP: Problema de Ruteo de Vehículos Capacitado
- FP: Feasibility Pump
- GLPK: Kit de Programación Lineal de GNU
- GMPL: Lenguaje de Programación Matemática de GNU
- LB: Local Branching
- LNS: Large Neighborhood Search (Búsqueda en Vecindario Ampliado)
- LP: Programación Lineal
- MILP: Programación lineal entera mixta
- MPVRP: Problema de Ruteo de Vehículos con Multi Periodo
- OSEA: Objective Scaling Ensemble Approach
- PTIC: Parque Tecnológico Industrial del Cerro
- SCIP: Solving Constraint Integer Programs
- VRP: Problemas de Ruteo de Vehículos

1. Introducción

En este proyecto se presenta el trabajo realizado sobre la aplicación de métodos heurísticos en problemas de programación lineal entera mixta (MILP, por sus siglas en inglés). El proyecto surge con el objetivo de profundizar en el estudio de los métodos de resolución de problemas MILP, enfocándose especialmente en instancias complejas. Algunas aplicaciones son problemas de planificación de la producción, de distribución de mercaderías, de recolección de residuos, de localización de depósitos y de asignación de tarea, entre otros (Wolsey, 1998; Hillier & Lieberman, 2010; Zhang *et al*., 2022).

En muchos de estos problemas, debido al tamaño de las instancias determinado por la cantidad de productos, cantidad de máquinas, cantidad de períodos y a características propias de los problemas, es inviable la resolución en tiempos acotados. Incluso a través de paquetes de *software* de optimización, el tiempo de cómputo requerido puede ser muy grande para los tiempos en que es requerida una solución (Klotz & Newman, 2013), fundamento para analizar la utilización de métodos alternativos de resolución. Se desea estudiar problemas *NP-hard* ya que son conocidos y desafiantes en la programación matemática, estos son problemas para los cuales generalmente no se conoce un algoritmo que los resuelva en tiempo polinomial en función del tamaño de la entrada. La clasificación de los problemas como *NP-hard* excede el alcance de este documento, para mayor detalle véase Havet (s.f.).

Dentro de los métodos de resolución de este tipo de problemas, las herramientas de vanguardia son los paquetes de *software* de optimización. Estos combinan heurísticas con métodos de búsqueda exactos buscando obtener soluciones de buena calidad (cercanas a la óptima) o incluso óptimas, en un tiempo razonable de cómputo de segundos o minutos (Achterberg & Wunderling, 2013). Se definen tres objetivos en este documento:

- La generación del estado del arte de métodos heurísticos en MILP
- La formulación e implementación de un problema MILP
- Resolución del problema mediante la implementación de diferentes heurísticas extraidas del estado del arte y análisis de los resultados

Por lo tanto, una vez culminada la etapa de investigación se creó el Apéndice I: Estado del Arte donde se realizó una revisión exhaustiva y actualizada del conocimiento existente sobre las heurísticas para problemas MILP. Este tiene como objetivo identificar, analizar y sintetizar las principales heurísticas que se han desarrollado hasta el momento. Además, se determinó cuáles son las líneas de investigación actuales y los desafíos abiertos en el área, prestando interés en la calidad de solución y tiempo insumido. Luego se pasó a una etapa de implementación de heurísticas seleccionadas del Apéndice I: Estado del Arte para la implementación, desarrollo y resolución de un problema real, más específicamente un problema de ruteo de vehículos con capacidad limitada y multi peridodo (CMPVRP, por sus siglas en inglés) dentro del Parque Tecnológico Industrial del Cerro (PTIC). Además, se experimentó con los métodos seleccionados y se analizaron los resultados para distintas situaciones y tamaños de problemas, comparándolos con métodos GLPK. Para esto se ejecutaron diversas instancias generadas a través de bibliotecas de problemas extraídas de Chaire de recherche en distributique – HEC Montréal (s.f.), además de la instancia generada

con los datos reales del problema. Para el logro de esta etapa se debió profundizar primero en la programación de las heurísticas y luego, en un desafío aún mayor, la codificación del problema y los métodos heurísticos.

El resto del documento se organiza de la siguiente manera: en la Sección 2 se presenta el marco teórico con los principales conceptos de programación matemática, paquetes de *software*, problemas de ruteo y heurísticas. En la Sección 3 se definen los problemas MILP abordando una aplicación real al problema de ruteo. El modelo matemático, la generación de datos y las técnicas de validación aplicadas se presentan en la Sección 4. En la Sección 5 se profundiza en los conceptos e implementación de heurísticas seleccionadas, mientras que en la Sección 6 se analizan los resultados obtenidos. Por último, en la Sección 7 se presentan las conclusiones y las posibles direcciones de trabajos futuros.

2. Marco teórico

En esta sección se introducen los conceptos de programación matemática, así como los métodos de resolución exactos y heurísticos. Por ello, en la Sección 2.1 se introducen los problemas de programación lineal entera mixta mientras que en la Sección 2.2 se introducen los paquetes de *software* de optimización de problemas MILP. Luego, en la Sección 2.3 se profundiza en el modelado específico de problemas de ruteo de vehículos (VRP por sus siglas en inglés) para culminar con una síntesis de las heurísticas aplicadas a problemas MILP en la Sección 2.4.

2.1 Programación lineal entera mixta

Un problema de programación lineal entera mixta busca optimizar una función objetivo lineal, sujeta a restricciones lineales, con variables de decisión discretas y continuas. Generalmente, las variables utilizadas son continuas y las variables discretas aparecen por estricta necesidad de las mismas. La existencia de estas se fundamenta en que muchas situaciones de la realidad requieren el uso de valores discretos, como cantidad de máquinas en la planificación y programación de la producción, enrutamiento de vehículos, y optimización en la industria energética (Wolsey, 1998). Vale aclarar que los problemas de programación lineal sin variables discretas (LP) pueden resolverse en forma eficiente mediante algoritmos de punto interior (exploran la región factible por puntos interiores) y mediante el método simplex (explora los vértices de la región factible), aunque existen algunos casos donde estos métodos no son eficientes (Spielman & Teng, 2004). Un concepto relevante para la resolución de problemas en programación matemática es el de la envolvente convexa. La envolvente convexa de un problema MILP se define como la menor región convexa que contiene a todas las soluciones factibles del problema (Wolsey, 1998).

A continuación se presenta una formulación general de los problemas MILP:

Minimizar	$c^T_1 x + c^T_2 y$	(1)
Sujeto a	$A_1 x + A_2 y \leq b$	(2)
	$x_j \geq 0 \quad \forall j \in J$	(3)
	$x_j \in \mathbb{Z} \quad \forall j \in J$	(4)
	$y_i \in \mathbb{R} \quad \forall i \in I$	(5)

Donde:

- J define al conjunto de índices tal que $J = \{1,\dots n\}$.
- I define al conjunto de índices tal que $I = \{1,\dots, o\}$.
- $x = (x_1, x_2,\dots, x_{n-1}, x_n)$ es el vector de las variables de decisión discretas de tamaño *n*.

- $y = (y_1, y_2, \ldots, y_{o-1}, y_o)$ es el vector de las variables de decisión continuas de tamaño *o*.
- $c_1 = (c_{1,1}, c_{1,2}, \ldots, c_{1,n-1}, c_{1,n})$ y $c_2 = (c_{2,1}, c_{2,2}, \ldots, c_{2,o-1}, c_{2,o})$ son los vectores coeficientes de la función objetivo.
- $A_1 \in \mathbb{R}^{mxn}$ y $A_2 \in \mathbb{R}^{mxo}$ son matrices que definen las restricciones lineales.
- $b = (b_1, b_2, \ldots, b_{m-1}, b_m)$ es el vector de términos independientes de las restricciones.

Un método exacto considerado fundacional para la resolución de problemas MILP es *Branch & Bound (B&B)* (Pochet & Wolsey, 2006). Este método es un procedimiento iterativo de numeración implícita que comienza relajando el dominio de las variables discretas del problema original. A partir de la solución obtenida se ramifica en subproblemas relajados en caso de que se viole la condición de integralidad de las variables. Además, en dicho caso, el valor funcional obtenido se toma como límite *(bound* en inglés*)* inferior en un problema de minimización o como límite superior en uno de maximización, ya que ninguna solución factible entera tomará un mejor valor funcional. Luego, con el objetivo de eliminar la solución obtenida, la cual es infactible en el MILP, se elige una variable fraccionaria como la variable de ramificación (*branch* en inglés) y se divide el problema en dos subproblemas relajados disjuntos. Cada subproblema se define con una restricción adicional del dominio de la variable de ramificación. Esta podrá tomar valores menores al entero por debajo y mayores al entero por encima en cada subproblema. Este proceso se repite en cada subproblema (nodo) generado mientras alguna variable continúe tomando valores fraccionarios y pueda ser seleccionada como variable de ramificación, o mientras no se cumpla otro criterio de terminación. La ramificación o poda de los nodos se realiza bajo ciertos criterios, los cuales aseguran que la poda se realizará si no es posible mejorar la solución obtenida. Por lo tanto, si la solución óptima del nodo cumple las restricciones originales de dominio entero no será ramificado. Si el nodo no tiene solución factible será eliminado terminando la ramificación del mismo. Por otro lado, en cada resolución de un nodo se actualiza el límite inferior (o superior) del valor funcional que puede ser obtenido de su ramificación si la solución es factible. Por lo tanto, si este es superior al valor funcional de una solución factible del MILP hallada, no tiene sentido continuar con su ramificación y la misma es abortada.

El método *B&B* es un método exacto y culmina cuando se haya explorado el árbol de búsqueda. La ramificación está en el núcleo de la complejidad exponencial y la forma en que se divide un problema dado en dos o más subproblemas puede tener un impacto muy significativo en el tamaño del árbol de búsqueda resultante. Por lo tanto, esta decisión puede ser crucial para poder resolver una instancia del problema en un tiempo razonable (Achterberg & Wunderling, 2013).

2.2 Paquetes de *software* de optimización

Los paquetes de *software* de optimización desempeñan un rol fundamental en la resolución de problemas MILP. En esta subsección se presentan los principales paquetes utilizados en la actualidad, destacando sus características y capacidades. Ejemplos de *software* comerciales

utilizados en la industria y en la academia son CPLEX, Xpress y Gurobi (Vrieswijk, 2023; Gurobi, s.f.-c; Guasque & Balbastre, 2022). Estos tres utilizan como base el método de *B&B* en donde se agregan diferentes métodos para mejorar la eficiencia del proceso de ramificación y acotación, tales como técnicas de planos de corte (*cutting planes* en inglés), heurísticas de búsqueda de soluciones factibles, y estrategias avanzadas de selección de nodos y variables (Riccardi *et al.*, 2021; Huang *et al.*, 2023; Zhang & Nicholson, 2019). Estos son utilizados para disminuir la complejidad del problema, pudiendo resolverlo en tiempos acotados y aportando soluciones de buena calidad. Dentro de los paquetes de *software* de optimización no comerciales se destacan SCIP y GLPK para la resolución de problemas MILPs.

Una de las funcionalidades de mayor interés de GLPK es la resolución a través de la combinación de *B&B* y planos de corte, resultando en un *Branch & Cut* como expresa en su guía Makhorin (2009). *Branch & Cut* utiliza restricciones adicionales específicas (llamadas planos de corte) para fortalecer el esquema de ramificación del *B&B*. Al optar por el uso de planos de corte en vez de la ramificación estos acotan la región factible del subproblema del nodo, buscando una solución factible mejor a la previamente encontrada. Con la misma, se espera encontrar una cota mejorada del valor funcional del problema para realizar una poda más eficiente. Dentro de los cortes más utilizados en MILP se encuentran los cortes de Gomory, entre otros, véase Noriega (2023) para profundizar en estos y en el método *Branch & Cut.*

Los paquetes de *software* de optimización hacen uso de diversas herramientas para la resolución de problemas de optimización. Estas permiten que, en muchos casos, la obtención de soluciones sea en tiempos de cómputo acotados. Dentro de estas herramientas se destaca la presolución, los planos de corte y diferentes heurísticas y métodos de descomposición (IBM, s/f; Gurobi Optimization, LLC, s.f.-f; Achterberg & Wunderling, 2013). La presolución busca disminuir el tamaño del problema y así su dificultad, transformando el modelo en uno equivalente que sólo será infactible o no acotado si el original lo es. Además, el valor funcional de la solución óptima del modelo será también el del modelo original. Cualquier solución factible del modelo presolucionado se puede relacionar a una solución factible del original (Gurobi Optimization, LLC, s.f.-d). Por lo tanto es una herramienta de mejora del modelado del problema, básicamente eliminando o sustituyendo variables (reemplazando por una combinación lineal de otras) y restricciones innecesarias. Por otro lado, los métodos de descomposición se fundamentan en el principio de dividir y conquistar. Este concepto consiste en descomponer un problema grande y complejo en subproblemas más pequeños y manejables. Estos y otros métodos son explicados en el Apéndice I: Estado del arte con mayor detalle.

Las diferencias de rendimiento entre paquetes de *software* (comerciales o no) evidencian distintos niveles de desarrollo de componentes como la presolución, la generación de cortes y las heurísticas. Los paquetes de *software* de optimización comerciales tienden a tener implementaciones más avanzadas y mejoradas en estos aspectos. El paquete de *software* SCIP, aunque es muy completo y modular no suele igualar la velocidad de los anteriores (Vrieswijk, 2023; Gurobi Optimization, LLC, s.f.-d).

2.3 Problemas de ruteo (VRP)

Los problemas de ruteo (VRP), introducidos por Dantzing & Ramser (1959) son problemas MILP conocidos en la literatura de investigación de operaciones. Este tipo de problemas suelen ser difíciles de resolver (*NP-hard*). Tal como se expresa en Lüer *et al.* (2009), los problemas de ruteo son el nombre genérico dado a la clase de problemas en los que se debe determinar una serie de rutas minimizando costos o distancias. Generalmente existe un vehículo o flota de vehículos que deben visitar una lista de clientes geográficamente dispersos, partiendo y culminando en uno o más depósitos. La función objetivo se puede expresar tanto en tiempo como en distancias. En base a las restricciones del problema se puede hacer ciertas diferenciaciones dentro de los VRP.

Un clásico problema de optimización es el *Travelling Salesman Problem* donde se desea encontrar el recorrido más corto que permita a un vendedor visitar exactamente una vez cada ciudad de un conjunto dado y regresar al punto de partida. Este y otros problemas que derivan del VRP son problemas que tienen restricciones de capacidad por vehículo y por lo tanto son llamados Problemas de Ruteo con Capacidad (CVRP por sus siglas en inglés). Dentro de otras restricciones se puede limitar el tiempo máximo permitido de trabajo, distancia máxima recorrida, etc. A continuación se presenta una formulación típica de problemas del tipo CVRP, extraída de Lüer *et al.* (2009) y diseñado para un sólo período de tiempo.

- L define al conjunto de ubicaciones tal que $L = \{0, 1, \ldots n\}$ donde 0 corresponde a la primera ubicación del depósito.
- K define al conjunto de vehículos tal que $K = \{1, \ldots, o\}$.
- Q_k es la capacidad del vehículo $k \in K$.
- st_{kl} es el tiempo de servicio de la demanda *l* por el vehículo $k \in K$.
- tt_{klm} es el tiempo de viaje desde *l* hasta *m* del vehículo $k \in K$.
- T_k es el máximo tiempo de ruta permitido para el vehículo $k \in K$.
- q_l es la demanda en $l \in L$.
- c_{lm} es el costo de moverse desde *l* a *m* con $(l, m) \in L \times L$.
- x_{klm} es la variable binaria que indica si la arista $(l, m) \in L \times L$ es transitada por el vehículo $k \in K$

Minimizar $$\sum_{l \in L} \sum_{m \in L} \sum_{k \in K} c_{lm} x_{klm} \quad (6)$$

Sujeto a $$\sum_{l \in L} \sum_{k \in K} x_{klm} = 1 \; \forall l \in L \backslash \{0\} \quad (7)$$

$$\sum_{l\epsilon L}\sum_{k\epsilon K} x_{kml} = 1\ \forall l\epsilon L\backslash\{0\} \quad (8)$$

$$\sum_{l\epsilon L} x_{klm} - \sum_{m\epsilon L} x_{kml} = 0\ \forall l\epsilon L,\ \forall k\epsilon K \quad (9)$$

$$\sum_{l\epsilon L}\sum_{m\epsilon L} q_l x_{klm} \leq Q_k\ \forall k\epsilon K \quad (10)$$

$$\sum_{l\epsilon L} st_{kl}\sum_{m\epsilon L} x_{klm} - \sum_{l\epsilon L}\sum_{m\epsilon L} tt_{klm} x_{klm} \leq T_k\ \forall k\epsilon K \quad (11)$$

$$\sum_{l\epsilon L\backslash\{0\}} x_{k0l} \leq 1\ \forall k\epsilon K \quad (12)$$

$$\sum_{l\epsilon L\backslash\{0\}} x_{kl0} \leq 1\ \forall k\epsilon K \quad (13)$$

En la función objetivo (6) se minimiza la sumatoria de los costos incurridos si se realiza el trayecto correspondiente. Las restricciones (7) y (8) garantizan que sólo un vehículo visite cada nodo de demanda. Por otra parte, con las restricciones (9) se busca mantener la continuidad de las rutas, ya que se exige que si un arco entra a un nodo, entonces tiene que salir. Además, con las restricciones (10) y (11) se respetan las capacidades de los vehículos, así como sus tiempos máximos de viaje. Finalmente, con las restricciones (12) y (13) se evita que se exceda la disponibilidad de vehículos (Lüer *et al.*, 2009).

2.4 Heurísticas para problemas MILPs

Como fue mencionado en la Sección 1, uno de los objetivos del proyecto fue la creación del Apéndice I: Estado del arte enfocado en la aplicación de heurísticas en problemas MILP. Para comprender el significado se hace una distinción entre métodos exactos y métodos heurísticos. Los métodos exactos son aquellos que garantizan la resolución de problemas hallando la solución óptima, si existe, siempre y cuando los recursos computacionales sean suficientes (tiempo y memoria). Las heurísticas, por otro lado, no garantizan la obtención de la solución óptima del problema pero tienden a ser más rápidas y suelen dar soluciones de calidad aceptable. Por calidad aceptable se entiende que, si bien no son óptimas, presentan un valor objetivo cercano al óptimo o que al menos satisfacen el objetivo perseguido. Además, cumplen con las restricciones del problema dentro de un margen tolerable para su aplicación práctica (Boschetti *et al.*, 2023). Por lo tanto, en el contexto del proyecto, el objetivo de la utilización de heurísticas será abordar la dificultad implícita en problemas complejos (*NP-hard*) como son mayoritariamente los problemas MILP. Con ello se procura encontrar soluciones en tiempos razonablemente acotados. Según Fischetti M. & Lodi A., (2014) se ha demostrado que el impacto de las heurísticas en el aspecto psicológico del usuario es significativo, especialmente al observar una solución factible de alta calidad obtenida en una etapa temprana del proceso.

En los problemas MILP *NP-hard* no sólo es difícil encontrar la solución óptima sino que también, en muchos casos, es difícil encontrar una solución factible inicial (Fischetti & Lodi, 2014). Para abordar esta problemática, en la literatura se han desarrollado heurísticas de

solución inicial, que buscan encontrar una solución que cumpla con los requerimientos de factibilidad. Por otro lado, se han desarrollado heurísticas de mejora, que parten de una solución inicial y la mejoran, sin necesariamente encontrar la solución óptima pero sí satisfacer un requerimiento.

Según el ámbito de ejecución, las heurísticas pueden estar planteadas para un problema genérico o para un problema específico. Las heurísticas genéricas se pueden utilizar en cualquier problema MILP, mientras que las heurísticas específicas están planteadas para problemas particulares. Este trabajo presenta interés en heurísticas genéricas, en las que no se aprovecha información relativa a la estructura y características específicas de un problema, sino que sirven para cualquier problema MILP. Dentro de algunas heurísticas destacadas se encuentran *Feasibility Pump (FP)* (Fischetti *et al.*, 2005), *Tabu Search* (Løkketangen *et al.*, 1994), *Objective Scaling Ensemble Approach* (OSEA) (Zhang & Nicholson, 2019), *Local Branching* (Fischetti & Lodi, (2003), *Local Search* (Shaw, (1998)) y *Large Neighborhood Search* (LNS) (Shaw, (1998)). A modo introductorio, en la Tabla 1 se describe la metodología de algunas de estas heurísticas seleccionadas.

Tabla 1 - Función de heurísticas seleccionadas

Heurística	Método
Feasibility Pump	Alterna entre soluciones continuas y enteras de distintos problemas hasta encontrar una solución factible del problema MILP. Será detallada en la Sección 5.3 al ser un método seleccionado para su implementación.
Local Branching	Agrega restricciones que limita las búsquedas relajadas de mejora de la solución a una distancia acotada de la misma, selecciona variables a destruir y resuelve el MILP resultante. Será detallada en la Sección 5.5 al ser un método seleccionado para su implementación.
Tabu Search	Mejora la solución en base a un registro de soluciones previas.
Local Search	Parte de una solución inicial y explora soluciones vecinas, aceptando un cambio si mejora el valor de la función objetivo.
RINS	Explora un vecindario entre la solución óptima relajada y la mejor solución entera encontrada.
Diving	Simula decisiones de ramificación, fijando variables según alguna estrategia (ej. fraccionalidad).
OSEA	Fija un conjunto de variables al valor nulo, reduciendo la región factible del MILP. Será detallada en la Sección 5.4 al ser un método seleccionado para su implementación.

Sub-MIP	Utiliza la resolución de un modelo reducido como un problema auxiliar.
Large Neighborhood Search	Relaja parcialmente una solución entera buscando una mejor solución en un vecindario amplio.
Rounding	Redondea soluciones fraccionales de manera efectiva para obtener soluciones factibles.

Estas heurísticas, entre otras, son desarrolladas en el Apéndice I: Estado del arte, pudiendo ser consultado para la profundización de la aplicación de las mismas. Este último documento clasifica las heurísticas según el método utilizado e indica el propósito específico de cada heurística. Los métodos son clasificados en: métodos de aprendizaje automático, descomposición, relajamiento, redondeo, estocásticos, vecindad (*neighborhood*) y fijación.

- Las heurísticas de aprendizaje automático (*machine learning*) utilizan información de una base de datos para poder definir, ajustar y fijar los valores de las variables. Utilizan información brindada por algún método o información de problemas relacionados como base. Estas heurísticas han tomado mayor relevancia en los últimos años con el aumento de prácticas relacionadas con el aprendizaje automático y la inteligencia artificial.

- Un método basado en descomposición divide al problema original en otros subproblemas más sencillos de resolver, los cuales van a permitir hallar una solución al problema original.

- Las heurísticas basadas en *relax* relajan el dominio de las variables discretas para poder aplicar algoritmos basados en variables continuas. Los algoritmos para problemas con variables continuas permiten la obtención de la solución óptima de forma más sencilla y estas heurísticas tienen diversas formas de volver a instalar la restricción de integralidad.

- Las heurísticas de redondeo se basan en redondear determinadas variables discretas a medida que se resuelven los problemas relajados resultantes.

- Las heurísticas estocásticas son técnicas de resolución de problemas que utilizan como base elementos de aleatoriedad o procesos probabilísticos para encontrar soluciones.

- Las heurísticas basadas en *neighborhood* exploran una región factible alrededor de una solución base que se irá actualizando, acotando así la región y acelerando la búsqueda.

- Las heurísticas basadas en fijación justamente fijan variables discretas del problema utilizando diferentes métodos. Esto genera un menor número de variables a optimizar, disminuyendo el tamaño del problema y así facilitando la resolución.

Relevada la literatura acerca de la implementación de heurísticas en problemas MILP, se propuso caracterizar a las utilizadas con foco en problemas VRP y por ello se brindan ciertos ejemplos representativos. En Jaikishan & Patil (2019) se utiliza una heurística llamada *Greedy*

Randomized Adaptive Search Procedure (GRASP) que utiliza un enfoque de dos fases. Primero se halla una solución inicial para luego mejorarla a través de un método *neighborhood*. La fase inicial se basa en un enfoque *greedy* que aprovecha la estructura específica del problema CVRP con heurísticas de construcción secuencial de rutas. Un enfoque *greedy* hace referencia a la elección de la opción óptima local, con la esperanza de llegar a una solución óptima global. La heurística GRASP comienza creando la ruta del primer vehículo, para ello partirá de un depósito elegido y se calculará para todos los nodos el costo de ser el siguiente destino. Con estos valores se propone formar una lista con los nodos que su costo no sobrepasen una cota y elegir el siguiente destino al azar de esa lista.

Este proceso se repite hasta llenar la capacidad del vehículo y definir la ruta siguiente. El proceso continúa hasta completar la capacidad de todos los vehículos sin cubrir la demanda (infactible y comienza nuevamente) o hasta haber cubierto la demanda (encontrando una solución inicial). Por otro lado, a la hora de aplicar la metodología *neighborhood,* para la mejora de la solución inicial, se definen funciones específicas para VRP, estudiando el efecto de cambios unitarios en la solución inicial obtenida y así evaluar la solución. Básicamente, dos pares de nodos de la mejor solución encontrada hasta el momento son elegidos aleatoriamente con la condición de que uno defina un tramo de la ruta de un vehículo y el otro un tramo de la ruta de otro vehículo. A partir de aquí se definen cuatro regiones vecinas según las posibles soluciones pertenezcan a alguno de los siguientes conjuntos:

- La solución es obtenida suprimiendo un tramo de un vehículo para asignarlo al otro.
- La solución es obtenida intercambiando los dos tramos entre vehículos (siendo los dos tramos disjuntos).
- La solución es obtenida invirtiendo el orden de uno o ambos tramos.
- La solución es obtenida intercambiando los tramos desde el nodo inicial elegido de cada par hasta el final del recorrido.

En términos de resultados, en el documento se expone que CPLEX resuelve los problemas pequeños pero tiene dificultades con problemas más grandes. Por otro lado se concluye que con las heurísticas como GRASP se pueden obtener soluciones en tiempos más cortos con igual o mejor calidad que con métodos exactos.

Por otro lado, en Goeke (2019) se busca resolver un problema CVRP mediante la implementación de un método exacto de *column generation* con la utilización de LNS para la optimización del método. A su vez, presenta un funcionamiento similar al propuesto en la anterior documento donde primero se busca una solución inicial y con el método *neighborhood* LNS se pretende mejorar la solución. Para encontrar una solución factible inicial se adapta un método de Clarke & Wright (1966) que consiste en un método específico para CVRPs en el que, igual que en el caso anterior, se generan trayectos individuales para luego evaluar la combinación de los mismos. Por lo tanto nuevamente se aplica una heurística de construcción secuencial de rutas que saca provecho del tipo de problema, potencialmente encontrando una solución factible. Para la segunda fase, la de mejora de la solución, se plantea una LNS en regiones definidas por potenciales soluciones que sólo difieren en el valor de una cantidad acotada de variables. Estas variables a destruir, a las que se le permite tomar otro valor, pueden ser elegidas mediante distintos métodos. Estos incluyen elección aleatoria, elección de las variables no deseadas identificadas como variables que conspiran contra la mejora de la solución, por proximidad entre nodos ya que se espera que en la nueva solución sean

asignados a un mismo vehículo, etc. Además, en el documento se detalla que LNS podría funcionar como método en solitario, siendo capaz de tanto encontrar una solución inicial como mejorarla para algunos casos.

De la revisión de literatura específica de aplicación en VRP se observa que la tendencia, así como en los dos casos anteriores, es la utilización de heurísticas en dos fases. En la primera fase se comienza con un método capaz de encontrar una solución inicial con técnicas de construcción secuencial de rutas. Estas se presentan como las más utilizadas para este tipo de problemas en específico. Las heurísticas de solución inicial forman un rol importante en la obtención de soluciones factibles para luego mejorarlas, especialmente para tratar con problemas difíciles (Joncour, *et al.*, 2022). Luego, a través de un método generalmente de *neighborhood* se busca mejorar esa solución hallada en la fase anterior.

Los métodos *neighborhood*, como se concluye en el Apéndice I: Estado del arte, son los que mejor se adaptan a los distintos tipos y características de los problemas, lo que explica el alto desarrollo de investigación de estas técnicas. Si bien no son las mejores metodologías para la resolución de problemas de gran tamaño, siendo superadas por heurísticas de descomposición o de aprendizaje automático, se compensa con una menor dificultad de aplicación. Además permiten una alta flexibilidad para la relación entre los tiempos y la calidad de solución ajustando parámetros claves como el tamaño del vecindario y la frecuencia de perturbaciones de la solución para escapar de óptimos locales.

Dentro de las heurísticas de *neighborhood* se destaca el método *Feasibility Pump* como un método fundacional. Este despertó el interés en muchos investigadores para generar diversas adaptaciones del mismo para la obtención de una solución inicial como se describe en Apéndice I: Estado del arte. En cuanto a heurísticas de *neighborhood* de solución final se recomienda observar la Sección 4.1.3 del Apéndice I: Estado del arte. En el mismo se puede profundizar en distintos métodos que han sido desarrollados en la literatura, los mismos se basan en la definición de regiones factibles pequeñas a través de distintos métodos. Estos pueden ser a través del acote de las variables entre valores específicos, fijación de variables, acote de la región factible por la definición de una función distancia especificando una cota, etc.

En este documento, serán implementadas las heurísticas *FP, OSEA* y *LB,* siendo desarrolladas en detalle en las Secciones 5.3, 5.4 y 5.5 respectivamente. Se consideran *FP* y *LB* como métodos fundacionales *neighborhood*, de extensa aplicación en problemas MILP y se desea implementar el método *OSEA* como uno de fijación y aplicación específica en problemas de ruteo, permitiendo la comparación entre distintos tipos de métodos.

3. Caso de estudio

En esta sección se detalla el problema real a resolver para evaluar las heurísticas que fueron implementadas. El análisis específico de este problema está motivado por la continuación del trabajo de pasantía de la carrera de grado de uno de los integrantes del equipo. En la Sección 3.1 se brinda un breve resumen del contexto del Parque Tecnológico Industrial del Cerro y en la Sección 3.2 se detalla el funcionamiento del proceso de recolección de los residuos clasificados. Por último, en la Sección 3.3 se presenta la problemática.

3.1 Contexto del Parque Tecnológico Industrial del Cerro

El Parque Tecnológico Industrial del Cerro es una iniciativa pública llevada adelante por la Intendencia de Montevideo y ubicado en el barrio del Cerro, Montevideo. El proyecto, comenzado en el año 1997, implicó la recuperación desde un estado ruinoso de los predios que antiguamente ocupaba la Planta Artigas del EFCSA (Establecimientos Frigoríficos del Cerro SA). Con la culminación del proyecto en el año 1999 con el paso de los años se transformó en un parque tecnológico que alberga un total de 56 empresas y 10 incubadoras. El PTIC recibe diariamente, entre trabajadores, estudiantes y otros agentes, más de 1500 personas. Los emprendimientos tienen gran heterogeneidad en sus áreas de trabajo donde se destacan los rubros de alimenticia, eléctrica, madera, ambiental, metalúrgica, naval, servicios, plástico, química y textil. En la Figura 1 se puede observar una toma aérea del PTIC, las inmediaciones del mismo ocupan 20 hectáreas que se encuentran ubicadas en la calle Haití 1500 en los padrones 43085, 427486, y 418373 del municipio A del departamento de Montevideo.

Figura 1 - Vista del PTIC y sus inmediaciones.

En el entorno inmediato del PTIC existe una zonificación urbana densa como se puede observar en la Figura 2 donde se observa la presencia de algunos asentamientos. Se destaca la proximidad al arroyo Pantanoso, el cual se encuentra muy deteriorado no solo por la contaminación histórica del mismo, sino por los basurales existentes en el entorno.

Figura 2 - Vista satelital de las inmediaciones del PTIC.

En base a la normativa para parques tecnológicos-científicos y con el objetivo de avanzar hacia una economía circular en el PTIC, en 2019 se inauguró la Planta de Clasificación de Residuos Sólidos Valorizables. Con esta se buscó una gestión ambientalmente correcta para la totalidad de los residuos sólidos industriales. Estos son generados por el sector empresarial dentro del PTIC y con la posterior valorización tercerizada de los mismos se sigue una estrategia de economía circular para fomentar la competitividad, crear nuevos puestos de trabajo y mantener un crecimiento económico sostenible. Además, es de vital importancia para el PTIC el correcto funcionamiento de la Planta de Clasificación para poder continuar su desarrollo de manera sostenible.

Con el objetivo de disminuir el pasivo ambiental en la Usina Felipe Cardoso, en el año 2020 el área ambiental del PTIC redactó un plan piloto de recolección de residuos del mismo. Este abarcó un análisis del alcance y los recursos necesarios para poner en funcionamiento la recolección de residuos sólidos valorizables por las empresas del PTIC. De esta manera se creó el plan de valorización de residuos del PTIC que consiste en la recolección de residuos clasificados de las empresas a través de recorridos y el recibimiento en planta.

3.2 Proceso de recolección de los residuos clasificados

Originalmente la creación de los recorridos del sistema de recolección comenzó con el análisis de la demanda de los principales generadores de residuos valorizables. En base a este análisis se creó un recorrido al que se le asignó una frecuencia semanal, añadiendo empresas cercanas para un mayor aprovechamiento de la capacidad del vehículo. Para la satisfacción de

la demanda del resto de las empresas no se utilizó una metodología establecida sino que se crearon dos recorridos en base a la cercanía entre empresas. La frecuencia de estos recorridos se estableció como quincenal y mensual.

La cooperativa “La Paloma” trabaja en la recolección de residuos valorizables urbanos. Esta cuenta con una planta de clasificación que está ubicada en las inmediaciones del estadio Luis Tróccoli, a partir de ahora llamada planta del Tróccoli. Por ello, la dirección del PTIC designó que sea la propia cooperativa quien gestione la Planta de Clasificación de Residuos Sólidos Valorizables dentro del PTIC. Los residuos valorizables generados en el PTIC son llevados a esta planta para su posterior venta a gestores habilitados. En la Figura 3 se realiza un diagrama del sistema de valorización del PTIC donde se representa la gestión de los residuos.

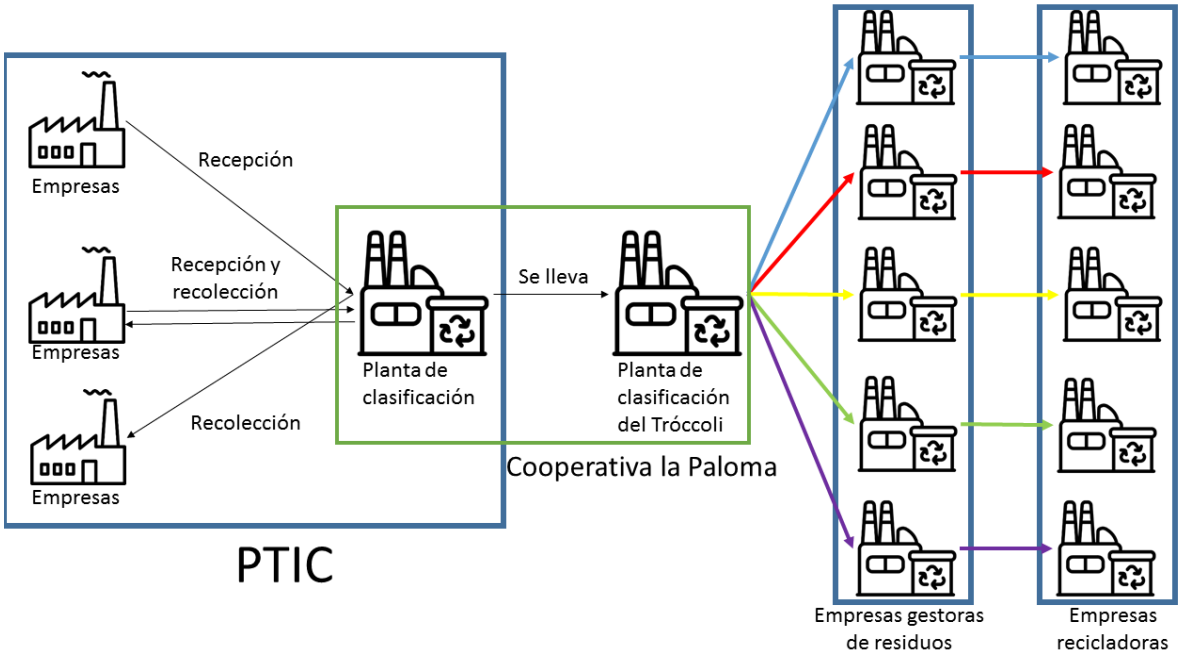

Figura 3 - Representación gráfica del funcionamiento del sistema de valorización del PTIC.

Son dos los trabajadores de la cooperativa que realizan los recorridos, utilizando un contenedor móvil de aproximadamente 800 litros. Además, cuentan con una balanza, una enfardadora y varios recipientes donde se depositan los residuos sólidos valorizables ya clasificados. Para el transporte desde la Planta de Clasificación de Residuos Sólidos Valorizables, a partir de ahora llamada simplemente planta de clasificación, hasta la planta del Tróccoli se cuenta con un motocarro y un camión. La jornada de los trabajadores es de 9:00 a 12:00 y la planta de clasificación es co-gestionada entre los pasantes del área de gestión ambiental del PTIC y la cooperativa La Paloma. En los recorridos, los residuos valorizables son clasificados por cada empresa y el personal de dicha cooperativa realiza la tarea de recolectar, transportar y registrar estos residuos. Luego, la mayoría de los residuos valorizables son enfardados según el tipo y son entregados a gestores habilitados que se encargan de su posterior valorización.

Los residuos sólidos valorizables son generados por las empresas en sus procesos productivos, administrativos, de compras, etc. En este contexto, se entiende por residuos sólidos valorizables al cartón, papel blanco, nylon, plástico PET, plástico PEAD y metales. El resto de residuos valorizables como pueden ser espuma plast, papel color, entre otros, no

cuentan en la actualidad con un mercado para su venta y posterior valorización por lo que no son considerados. Otros residuos que se dejan por fuera son los orgánicos compostables, que ya se recolectan una vez por semana en todo el parque a través de un acuerdo con el Centro Uruguay Independiente (CUI) los cuales utilizan los residuos para compostaje en la huerta del PTIC.

Considerando disconformidades por parte de diferentes clientes del servicio acerca de la frecuencia en la que los residuos son recolectados se llevó a cabo un análisis de los recorridos. Este fue llevado a cabo por los pasantes relevando los datos históricos de recolección, realizando los recorridos conjuntamente con los trabajadores de la cooperativa, entre otras actividades.

3.3 Definición del problema

Con la información recabada por los pasantes, se observó una disminución en la captación de residuos valorizables a lo largo de los semestres y se identificaron problemas que surgen de la forma actual de los recorridos, estos básicamente son que:

- Se culminan recorridos sin visitar empresas debido a la limitación de capacidad del contenedor móvil.

- Se visitan empresas que no tienen residuos generados mientras que otras empresas han alcanzado la capacidad de sus contenedores y no han sido visitadas.

- Las empresas, habiendo colmado la capacidad de sus contenedores, optan por el manejo propio de sus residuos valorizables, en muchos casos desechándolos como residuos comunes.

Planteada esta realidad se busca conocer si es posible evitar o reducir estos inconvenientes mediante la optimización de los recorridos. Para ello, se propone modelar el problema como un MILP, relevando y formalizando las restricciones y características clave del sistema, con el fin de analizar y mejorar su funcionamiento. Este problema se puede modelar como una variación VRP. Específicamente, se trata de un Problema de Ruteo con Capacidad y Multi Período (CMPVRP por sus siglas en inglés) con un solo vehículo (Jafarian-Moghaddam & Shahpoori-Arany, 2024). En los Problemas de Ruteo con Capacidad (CVRP) el vehículo está limitado por algún tipo de capacidad y se debe cumplir cierta demanda de los clientes, no solo visitarlos. Los Problemas de Ruteo Multi Período (MPVRP) consisten en planificar las rutas de los vehículos a lo largo de un horizonte temporal, determinando los recorridos que se realizarán en los distintos períodos de tiempo.

4. Modelado, validación y preparación de datos del problema MILP

En esta sección, a partir de la definición de la problemática brindada en la Sección 3.3 se propone un modelado del problema MILP en la Sección 4.1. Este modelo es validado en la Sección 4.2. Por último, en la Sección 4.3 se presenta la forma en que los valores de los parámetros necesarios son generados, tanto para el problema MILP del PTIC en la Sección 4.3.1 como para otras instancias presentadas en la Sección 4.3.2.

4.1 Modelado del problema MILP

El problema de optimización de los recorridos del PTIC se define como el problema de minimización de la distancia a recorrer por los trabajadores en la recolección de residuos. Dicha recolección es llevada a cabo utilizando un único recipiente con ruedas como vehículo, con espacio limitado, en un horizonte de tiempo de un mes. Se debe visitar las empresas incorporadas al plan de valorización del PTIC las cuales generan residuos valorizables que se deben recolectar. Por el horario de disponibilidad de los trabajadores para estas tareas sólo se puede realizar un recorrido por día, buscando captar la totalidad de los residuos generados por las empresas. Para el correcto modelado se define que no se puede visitar una misma empresa en menos de 5 días para permitir la acumulación de los residuos valorizables. En base al relevamiento llevado a cabo en la Sección 4.3 se asume una cantidad fija de generación de residuos por mes de parte de las empresas. Además, se supone que al primer día del mes ya se generan los residuos de todo el mes.

Con estas suposiciones se desea hacer énfasis en la optimización de los recorridos y estudiar su factibilidad. Al tratarse de residuos limpios y secos, que no generan lixiviados o emisiones que impliquen un riesgo para la salud pública o el ambiente, es viable analizar un aumento de la capacidad de almacenamiento de estos materiales en las empresas sin que esto suponga un problema sanitario o ambiental. Es por ello que los tiempos de estadía de los residuos valorizables en los contenedores de los clientes no son tenidos en cuenta, asumiendo que los problemas de capacidad de los mismos serán resueltos con un aumento de estas capacidades de ser necesario.

En base a la información presentada y para abordar la problemática propuesta se presenta a continuación el modelo matemático del problema MILP.

- N define al conjunto de empresas pertenecientes al sistema de valorización de residuos tal que $N = \{0, 1,..., M\}$. La primera posición se designa al centro de clasificación y *M* es el número de empresas en el sistema de valorización del PTIC.

- T define al conjunto de días hábiles dentro de un mes tal que $T = \{0, 1,..., m\}$ en donde varía el tamaño *m* según el mes y el año.

- D_{ij} es la distancia entre *i* y *j* con $(i, j) \in N \times N$ expresada en unidades de distancia.

- *C* es la capacidad máxima del contenedor de recolección expresada en unidades de masa.
- Q_i es la cantidad de residuos generados por mes en el punto i con $i \in N$ expresada en unidades de masa.
- x_{ijt} es la variable binaria que indica si se hace el trayecto de i a j el día t, con $(i, j) \in N \times N, t \in T$.
- y_{it} es la variable continua que indica la cantidad de residuos que se toman en el punto i el día t, con $i \in N$, $t \in T$ expresada en unidades de masa.
- q_{it} es la variable continua que indica la cantidad de residuos a recolectar actualizado en el punto i el día t, con $i \in N$, $t \in T$ expresada en unidades de masa.
- z_t es la variable binaria que indica si se hace o no algún recorrido el día t, con $t \in T$.
- u_{it} es la variable continua asociada al orden en que se visita el nodo i el día t, con $i \in N, t \in T$.

$$min \sum_{t\epsilon T} \sum_{i\epsilon N} \sum_{j\epsilon N} D_{ij} x_{ijt} \quad (14)$$

$$\sum_{i\epsilon N} y_{it} \leq C \; \forall t\epsilon T \quad (15)$$

$$y_{it} \leq q_{it} \; \forall i\epsilon N, \; \forall t\epsilon T \quad (16)$$

$$\sum_{t\epsilon T} y_{it} = Q_i \; \forall i\epsilon N \quad (17)$$

$$y_{it} \leq C(\sum_{j\epsilon N,} x_{ijt}) \; \forall i\epsilon N, \; \forall t\epsilon T \quad (18)$$

$$q_{it} = Q_i - \sum_{n=0}^{t-1} y_{in} \; \forall i\epsilon N, \; \forall t\epsilon T \quad (19)$$

$$\sum_{j\epsilon N} x_{0jt} = z_t \; \forall t\epsilon T \quad (20)$$

$$\sum_{j\epsilon N} x_{j0t} = z_t \; \forall t\epsilon T \quad (21)$$

$$Mz_t \geq \sum_{i\epsilon N} \sum_{j\epsilon N} x_{ijt} \forall t\epsilon T \quad (22)$$

$$\sum_{j\epsilon N, j\neq i} x_{ijt} = \sum_{j\epsilon N, j\neq i} x_{jit} \forall t\epsilon T, \; \forall i\epsilon N \quad (23)$$

$$x_{iit} = 0 \; \forall t \epsilon T, \; \forall i \epsilon N \quad (24)$$

$$\sum_{n \epsilon T,\, n=t}^{t+4} \sum_{i \epsilon N, j \neq} x_{ijn} \leq 1 \; \forall i \epsilon N, \; \forall t \epsilon T \quad (25)$$

$$u_{it} - u_{jt} + M * x_{ijt} \leq M - 1 \; \forall i \epsilon N \backslash \{0\}, \; \forall j \epsilon N \; / \; j \neq i, \; \forall t \epsilon T \quad (26)$$

$$z_t \leq \sum_{i \epsilon E \backslash \{0\}} u_{it}, \; \forall t \epsilon T \quad (27)$$

$$x_{ijt} \epsilon \{0, 1\} \quad (28)$$

$$z_t \epsilon \{0, 1\} \quad (29)$$

$$q_{it} \geq 0 \quad (30)$$

$$y_{it} \geq 0 \quad (31)$$

$$0 \leq u_{it} \leq M \quad (32)$$

En la función objetivo (14) se minimiza la sumatoria de distancias recorridas a lo largo del mes. La familia de restricciones (15) limitan la recolección de un día por la capacidad del contenedor. Las restricciones (16) limitan lo que se puede recolectar en cada empresa por los residuos generados en la misma. En las restricciones (17) se obliga a realizar los recorridos buscando que por mes se tomen todos los residuos valorizables generados por las empresas. Las restricciones (18) obligan a que si se recogen residuos en *i*, se continúe con el recorrido. La familia de restricciones (19) dan sentido a las variables de demanda, estas variables van a disminuir según se van tomando los residuos. Las restricciones (20) y (21) obligan a los recorridos a comenzar y terminar en el centro de clasificación. Las restricciones (22) permiten determinar qué días se hacen recorridos. La familia de restricciones (23) obligan que los recorridos sean un circuito. La restricción (24) evita que se haga un viaje de un nodo a sí mismo. Las restricciones (25) buscan restringir que no se puede volver a visitar una empresa en los siguientes 5 días. Las restricciones (26) y (27) buscan forzar el ordenamiento una vez que se hacen los recorridos y de esta manera eliminar subciclos (recorridos que no contienen el depósito). Las restricciones (28) a (32) son de dominio y determina el rango de valores que puede tomar las variables. Las restricciones (28) y (29) determinan que las variables sean binarias. Las restricciones (30) y (31) determinan las variables que toman valores reales positivos. Las restricciones (32) acotan el valor de la variable asociada al orden de los recorridos.

4.2 Validación

La validación de un modelo de programación lineal es una etapa esencial en el desarrollo de modelos de optimización, ya que asegura que el modelo es adecuado y representativo del problema que se busca resolver. Al implementar la validación se encontraron diferentes errores del modelo, lo que permitió corregir antes de su implementación. Las instancias fueron

ejecutadas en el sistema operativo en que fueron desarrollados los métodos. Específicamente, Ubuntu con la versión 24.04.2 LTS a través de la máquina virtual de Virtualbox, en una computadora de uso personal con características: CPU Intel Core i5 - 12400f - 2,50 GHz - 8 Gb de RAM.

Se plantearon un total de ocho casos. En los primeros cuatro, se analizan situaciones límite en las que el modelo pierde validez o deja de ser aplicable. Primero se considera el caso en que no hay empresas generadoras de residuos y en el segundo el caso en que no hay días para hacer los recorridos. En el tercer caso se plantea que las empresas no generan residuos y por último, en el cuarto caso se plantea que la capacidad del contenedor es nula. Por lo tanto, para todos estos casos se espera que el resultado sea infactible debido a que se imposibilita la realización de los recorridos. El resultado, tal como se puede observar en el Apéndice II Validación del Modelo, fue el esperado.

Siguiendo con la validación se planteó un caso general, pequeño y controlado para observar si la solución óptima del problema es la esperada. Por lo tanto se creó un caso con un límite de tiempo de dos días hábiles donde se recogen residuos de tres empresas generadoras cuya suma de residuos a recolectar sea menor a la capacidad del contenedor, confeccionando un problema factible. En este caso, la solución óptima es visitar todas las empresas en un solo circuito cuyo resultado es la suma de las distancias de cada empresa hasta cerrar el circuito, obteniéndose el resultado esperado. Por otro lado, para comprobar el buen funcionamiento de las restricciones (12), que obligan a que no se visite una empresa en menos de 5 días hábiles, se crea un caso de prueba. Para hacerlo, se creó un único punto de recolección donde se genera mayor cantidad de residuos que la capacidad del contenedor y 6 días de tiempo límite para realizar los recorridos. Para este caso se obtuvo el resultado esperado, el primer día se visita la empresa, se toman residuos y 5 días después se vuelve para tomar los residuos restantes. Por último, como se profundiza en el Apéndice II Validación del Modelo, se crearon dos casos más. Uno para comprobar la negación de generación de subciclos y otro para comprobar que el modelo busca la optimalidad de las distancias.

Una vez validado el modelo, se procedió a resolver el caso de estudio del PTIC con los datos relevados en la Sección 4.3 y un tiempo máximo de 24 horas para obtener una solución inicial. Luego de este periodo, el paquete de *software* fue incapaz de encontrar la misma. Por lo tanto se procedió a una revisión específica de literatura acerca de problemas VRP, buscando la razón por la que, ya con el problema validado, el *software* de optimización no lograba encontrar una solución como era esperado en el tiempo designado. Cabe destacar que para ese momento ya se había ejecutado el método *Feasibility Pump* de la Sección 5.2, obteniendo soluciones factibles en promedio de 15 minutos, por lo que resultaba extraño que paquetes de *software* no fueran capaces de obtener una solución inicial en tiempos menores .

A través de la revisión de literatura se logró disminuir considerablemente la dificultad del problema. Para ello se observó que, si bien las variables u_{it} deben pertenecer a números discretos, no es necesario especificar esto en el modelado, por el contrario, se libera a que tomen valores reales. Si bien estas variables indican el orden de visita de los nodos y se espera que sean números discretos, como se formula en de Prado *et al*. (2022), si se permite que las mismas tomen valores reales, el resultado de las mismas serán números discretos. Debido a la disminución de dificultad que este cambio representa, se observó una notable disminución en los tiempos de cómputo de resolución a través de paquetes de *software*. *Feasibility Pump* en su procedimiento ya relajaba estas variables y por ello se observaban tan superiores resultados en comparación. En particular, luego de estos cambios y como se presentará en los resultados de

la Sección 6.2, fue posible hallar soluciones factibles en tiempos acotados con los paquetes de *software* utilizados. Con ello, se cumple con uno de los objetivos que era asegurar la factibilidad en la solución del problema.

4.3. Relevamiento y generación de datos

En esta sección se presentan cómo son generados los parámetros necesarios para la resolución del problema MILP presentado. Primero se mostrará cómo fueron generados para el problema de aplicación del PTIC y luego cómo fueron generados para analizar otras instancias de problemas MILP con la misma formulación.

4.3.1 Generación de datos del problema MILP del PTIC

Los valores de los parámetros del modelo son: el número de empresas asociadas al sistema de recolección, las distancias entre el centro de clasificación y estas, la cantidad de residuos generados en cada empresa y la capacidad del carro de recolección.

La distancia D_{ij} entre los puntos *i* y *j* se obtuvo a través de la aplicación Autocad. Se decidió utilizar esta herramienta ya que el PTIC cuenta con un área de arquitectura que brindó un plano del PTIC en Autocad permitiendo la localización de nodos con sus respectivas coordenadas en los ejes cartesianos. A aquellas empresas que se encuentran en altura en algún piso de los edificios, se agrega la altura en el eje z. Utilizando las coordenadas de todos los puntos se determinó la distancia entre estos utilizando la distancia euclidiana. Esta matriz distancia, cuadrada, simétrica y cuya diagonal se conforma por valores nulos, fue hallada a través de un código implementado en Python. En la Figura 4 se representa el plano del PTIC ubicando de color amarillo al punto de distribución (planta de clasificación de residuos) y en magenta los puntos de recolección (las empresas del PTIC). Con esto en cuenta se definen un total de 31 emprendimientos a visitar en 21 días hábiles dentro de un mes.

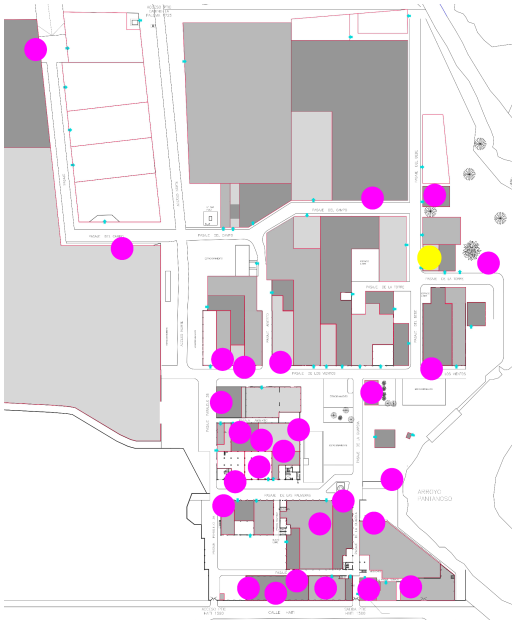

Figura 4 - Plano del PTIC

Un ejemplo de las distancias obtenidas se encuentra en la Tabla 2 donde se presenta una submatriz de los datos de distancia entre 10 puntos de recolección, siendo el punto 1 el del depósito. En el uso de los datos, priorizando la privacidad y el compromiso con la información, no se utilizan los nombres de las empresas y emprendimientos que forman parte del sistema de valorización. Por este motivo, a cada una de las organizaciones se le designa un número identificativo el cual se utiliza a la hora de mostrar los resultados.

Tabla 2 - Matriz distancia del problema original acotada a 10 puntos.

Puntos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.00	240.15	107.75	225.48	23.25	210.33	166.04	269.33	147.66	177.30
2	240.15	0.00	133.12	46.52	262.25	70.39	169.07	54.76	112.03	116.38
3	107.75	133.12	0.00	125.31	129.35	106.82	108.09	162.40	50.89	89.13
4	225.48	46.52	125.31	0.00	248.52	103.52	190.06	101.24	122.88	136.67
5	23.25	262.25	129.35	248.52	0.00	229.70	177.58	289.81	166.54	194.47
6	210.33	70.39	106.82	103.52	229.70	0.00	101.88	63.82	63.35	61.10
7	166.04	169.07	108.09	190.06	177.58	101.88	0.00	161.87	71.27	72.28
8	269.33	54.76	162.40	101.24	289.81	63.82	161.87	0.00	125.52	116.90
9	147.66	112.03	50.89	122.88	166.54	63.35	71.27	125.52	0.00	49.26
10	177.30	116.38	89.13	136.67	194.47	61.10	72.28	116.90	49.26	0.00

Tabla 3 - Cantidad de residuos Q_i que se debe tomar acotada a 20 empresas.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.00	16.49	3.27	4.45	2.09	53.54	4.31	7.42	1.05	3.70	0.88	1.02	1.45	0.37	1.19	32.56	7.58	1.79	0.54	0.58

Por otro lado, el valor del parámetro de capacidad máxima *C* se toma como un promedio de diversos valores de peso en donde se observó que el carro móvil estaba lleno al finalizar un recorrido. Para la cantidad de residuos generados por mes Q_i se tomó un promedio mensual de los valores obtenidos en el correr del año. Para ello se utilizaron datos históricos del año 2023 creando un promedio de la cantidad de residuos generados en todo el año. Un ejemplo de los resultados obtenidos aparece en la Tabla 3 donde se muestran 20 datos de residuos a recolectar de 20 puntos diferentes. Como se mencionó antes, el punto "1" es el punto de salida de los recorridos y allí no se genera una demanda de residuos a recoger.

4.3.2 Generación de datos del problema MILP para otras instancias

Con el objetivo de probar el modelo y las heurísticas en otras condiciones se decide crear instancias de problemas con nuevos valores y tamaños extraídos de Chaire de recherche en distributique – HEC Montréal (s.f.). Estos brindan una base de datos para las coordenadas de los puntos de recolección, demandas, cantidad y capacidad de los vehículos. Los datos no se usaron directamente sino que sufrieron algunos cambios para que se ajusten al tipo de problema en el que se trabaja.

En primer lugar, el modelo propuesto no contempla más de un vehículo por lo que en todos los casos se modifica por uno solo. Por lo tanto se modificó la capacidad del vehículo haciéndola suficiente para la factibilidad del problema. Para ello se igualó la capacidad del vehículo a la suma de las capacidades de todos los vehículos del problema de la biblioteca. Las instancias de la biblioteca son de gran tamaño, la más pequeña contiene 60 puntos de recolección, causando una gran demora en los tiempos de cómputo para los ordenadores de uso personal, imposibilitando obtener soluciones en menos de 24 horas. Por tal motivo se decidió crear 2 tipos de instancias en las que se reduce el tamaño: una con la misma cantidad de empresas que el PTIC (31) y otra donde se varía la cantidad de las empresas entre 10, 20, 30 y 40.

De esta forma se determinan un total de 9 instancias de las cuales 5 de ellas se realizan con 31 emprendimientos modificando los valores de residuos generados y distancias. Para las 4 instancias restantes se varió la cantidad de emprendimientos entre 10, 20, 30 o 40 además de las distancias y cantidades de residuos generados. Con estos dos tipos de instancias se esperan variaciones en la dificultad de los problemas tanto por los cambios de tamaño del problema como de los parámetros utilizados. En la Tabla 4 se presenta un resumen de las diferentes instancias. En la tercera columna se indica si los datos fueron obtenidos de la realidad del PTIC o son problemas generados a partir de bibliotecas de problemas de ruteo de vehículos. En la cuarta columna se presenta un factor creado para la variación de la dificultad de los problemas.

Tabla 4 - Instancias de funcionamiento del modelo

Instancia	Cantidad de empresas	Obtención de datos	Factor alfa (modificador de la demanda)
1	31	Realidad	1
2	31	Literatura	1,2

3	31	Literatura	0,8
4	31	Literatura	1,7
5	31	Literatura	1
6	10	Literatura	1
7	20	Literatura	1
8	30	Literatura	1
9	40	Literatura	1

En las instancias 2 hasta la 5, donde la cantidad de emprendimientos es igual a la del problema original, surgió la inquietud de que los problemas sean infactibles debido a que se están modificando los datos por los de la bibliografía. Si bien se redujo el tamaño de los problemas, se decidió mantener la demanda de recolección de cada uno de los puntos, debiendo analizar si la capacidad del vehículo debe ser modificada para hacer factible el problema. La nueva capacidad de vehículo para las instancias 2 a 5 se determina a través de la ecuación (33) en la cual se utiliza un ajuste proporcional de la capacidad del vehículo según la demanda de la instancia inicial 1, que se sabe que es factible. De este modo se busca asegurar la factibilidad de los problemas. De todas formas, antes de ejecutar las nuevas instancias se demostrará la factibilidad corriendo el modelo en GLPK hasta encontrar una solución inicial.

$$C^i = \frac{\sum_{n \epsilon N} Q_n^i}{\sum_{n \epsilon N} Q_n^0} \times C^0 \times \alpha \qquad (33)$$

En la ecuación (33) Q_n^i es la demanda de la empresa y C^i es la capacidad del vehículo con $n \in N, i \in I'$, siendo N el conjunto de empresas e $I' = \{2, 3, 4, 5\}$ el subconjunto de las instancias a las que se aplica la ecuación. Por último, se definió un factor $\alpha \in [0,8; 1,7]$ que busca, a través de la variación de la capacidad del contenedor, la variación de la dificultad del problema a través del tamaño de la región factible. Esto se realiza con el objetivo de observar cómo influye en la implementación de los métodos propuestos distintas variaciones.

5. Implementaciones de las heurísticas

En esta sección se presenta el trabajo realizado en el proyecto para el modelado de los problemas MILP y la implementación de distintos métodos heurísticos. Estas implementaciones fueron llevadas a cabo en el lenguaje de programación C++ utilizando distintos paquetes de *software* y las bibliotecas provistas por estos paquetes. De esta forma, la Sección 5.1 presenta la implementación del problema utilizando el lenguaje de modelado de GLPK. La Sección 5.2 aborda la implementación de la heurística FP, mientras que la Sección 5.3 se enfoca en el método heurístico OSEA. En la Sección 5.4 se describe la implementación de la heurística LB. Por último, en la Sección 5.5 se presenta la implementación del problema en otros paquetes de *software*.

5.1 Implementación del modelo

Una vez definido el problema, existen distintos paquetes de *software* destinados a resolver problemas MILP, tal y como se vio en la Sección 2.2. En este sentido, uno de los lenguajes adecuados para comenzar con la implementación de este tipo de problemas es GMPL (GNU *Math Programming Language*). Este es el lenguaje de modelado del paquete GLPK (GNU *Linear Programming Kit*), una herramienta de optimización de código abierto desarrollada por el proyecto GNU. Al igual que AMPL (*A Mathematical Programming Language*), consultar AMPL, (s.f.), uno de los lenguajes de modelado más conocidos y utilizados en la optimización matemática, GMPL permite expresar modelos de programación lineal y entera de manera declarativa. Además, ofrece una estructura que separa el modelo de los datos. GMPL está fuertemente inspirado en la sintaxis y filosofía de AMPL, por lo que muchos modelos escritos en AMPL pueden adaptarse con menores modificaciones a GMPL. El lenguaje seleccionado para la implementación del modelo fue GMPL debido a que el lenguaje de modelado GMPL posee facilidad en la escritura del código. El código del modelo es presentado en el Apéndice III: Modelado del problema en GNU, pudiéndose observar una estructura prácticamente idéntica a la del modelo de la Sección 4.1. Además, es un paquete conocido por los integrantes del proyecto y del cual se conoce una biblioteca para el uso de diferentes funcionalidades, ver Makhorin (2009).

Las limitaciones del GMPL son observadas cuando se cargan los datos del problema de aplicación, el cual tiene 22134 variables y 22291 restricciones. Debido al tamaño del problema y su complejidad implícita, el paquete de *software* de optimización no encuentra una solución óptima del problema en tiempos razonables. El modelo fue ejecutado en distintas computadoras personales, con diversas capacidades de hardware y, aún tras más de 48 horas de procesamiento, GLPK no logró encontrar la solución óptima. Sin embargo, el paquete de *software* encuentra una solución inicial en tiempos de cómputo menores a una hora, asociándose un gap de dualidad de 80% aproximadamente. Originalmente no se obtenía un resultado inicial luego de las 48 horas de ejecutado el paquete de *software*, pero al relajar las variables u_{it} tal como fue explicado en la Sección 4.2 se hallaron las soluciones mencionadas.

Del análisis del funcionamiento de las heurísticas para problemas MILP mencionadas en la Sección 2.4 se presenta la necesidad de la utilización de otro lenguaje de programación. Este debe de permitir hacer uso de funcionalidades como volver a ejecutar modelos, asignación de nuevas variables o restricciones y permitir hallar y guardar los resultados alcanzados. Estas

funcionalidades suponen la modificación de los procesos de búsqueda de la solución, limitando así el uso directo de GMPL. En este sentido, GLPK está organizada como una biblioteca *callable* que brinda un conjunto de rutinas o funciones escritas en lenguaje de programación ANSI C que permite llamarlas directamente desde un lenguaje de programación. Específicamente, GLPK permite la creación de modelos y la resolución de problemas LP y MILP. Además, contiene funcionalidades varias del lenguaje de C++ como la utilización de números pseudoaleatorios que son de utilidad en los diferentes métodos. Con estas, si bien el modelado requiere de mayor esfuerzo de codificación, luego es posible gestionar la búsqueda de acuerdo a la necesidades (Makhorin, 2009).

Como se mencionó anteriormente, el sistema operativo que se decide utilizar para desarrollar, compilar y ejecutar los programas es Ubuntu con la versión 24.04.2 LTS. Este sistema es elegido ya que se intentó utilizar la aplicación de Visual Studio Code en el entorno brindado por Windows (11 y 10) ya que es el sistema operativo instalado en las computadoras personales de los integrantes, pero se encontraron dificultades. El principal inconveniente se originó entre el compilador del lenguaje de C++ y la biblioteca de GLPK para Windows, impidiendo la compilación y la utilización de las funcionalidades de la biblioteca en el código. Una vez probado en el sistema operativo de Ubuntu, este demostró una mayor facilidad a la hora de instalar los paquetes necesario a través de dos simples líneas de código:

Compilador de C++: sudo apt install gcc g++

Biblioteca de GLPK: sudo apt-get install glpk-utils libglpk-dev glpk-doc

Una vez ejecutado estos códigos en la línea de comandos de Ubuntu, el llamado de las diferentes funcionalidades tanto de C++ como de la biblioteca de GLPK funcionaron sin inconvenientes. Para la implementación del modelo se utilizó la biblioteca de GLPK haciendo uso de diferentes funcionalidades que serán desarrolladas a continuación:

- glp_create_prob: Genera un puntero a un objeto donde se definirá todo el problema.
- glp_set_obj_dir: Define si la optimización es una minimización o una maximización.
- glp_set_prob_name: Da nombre identificativo al problema.
- glp_add_rows: Añade tantas restricciones como sea necesario.
- glp_add_cols: Añade tantas variables como sea necesario.
- glp_set_row_bnds: Define el tipo de desigualdad de la restricción y el valor independiente.
- glp_set_col_bnds: Define las cotas inferiores y superiores (en caso de existir) de las variables.
- glp set col kind: Define el tipo de las variables (binaria, entera o continua).
- glp_set_obj_coef: Define por qué valor es multiplicada cada variable en la función objetivo.

- glp_set_mat_row: Define los parámetros a multiplicar cada variable en cada restricción.
- glp_simplex: Resuelve el problema con todas las variables relajadas.
- glp_intopt: Resuelve el problema MILP con el método de *branch & cut*.
- glp_write_prob: Escribe el problema en formato de GLPK para poder analizar el correcto funcionamiento del código.

Utilizando estas funcionalidades se logró escribir el problema utilizando esta nueva forma de codificar el modelo. Este proceso fue dificultoso principalmente por la inexperiencia del grupo en la utilización del mismo. La principal dificultad consistió en la codificación del modelo la cual es extensa, debiendo comprender detalles como los índices y su correcta definición para cada variable. Otra dificultad fue la inexperiencia del equipo trabajando con bibliotecas de funciones y específicamente con la de GLPK donde algunas se debió de estudiar todas las funciones y su correcto uso.

En la Figura 5 se presenta un extracto del código de modelado del problema, en este caso se define el primer conjunto de variables. Se puede observar por lo tanto como primero se crean las variables a través del comando *glp_add_cols* y se procede a definir las primeras $E \times E \times T$ variables. Para la correcta definición de cada variable se recurre a parámetros auxiliares, *index* y *auxaux*, que permiten enumerar las variables y asignar el correcto coeficiente de las mismas en la función objetivo. Además, para cada variable se define su nombre, el tipo (enteras, binarias o discretas) y los límites, *0* y *1* para variables binarias por ejemplo. Las variables presentadas en la Figura 5 representan si el arco entre dos nodos es realizado en el período de tiempo correspondiente, valiendo *1* si efectivamente es realizado y *0* en el caso contrario.

```
//Definicion de variables

glp_add_cols(lp, cantvar);

int index=0;
int auxaux=0;
for (int t = 1; t <= T; t++) {
    auxaux=0;
    for (int j = 1; j <= E; j++) {
        for (int i = 1; i <= E; i++) {
            index++;
            auxaux++;
            string nombre = "x" + to_string(i) + "" + to_string(j) + "" + to_string(t);
            glp_set_col_name(lp, index, nombre.c_str());
            glp_set_col_kind(lp, index, GLP_BV);
            glp_set_col_bnds(lp, index, GLP_DB, 0, 1);
            glp_set_obj_coef(lp, index, Distancia[auxaux]);
        }
    }
}

varfijas = E * E * T;
```

Figura 5 - Definición de variables

En la Figura 6 se muestra un ejemplo en donde se observa la representación de la restricción (15) del modelo del problema del PTIC, mostrado en la Sección 4.1. Se observa cómo se debe definir *T* restricciones, donde *T* es la cantidad de días en el que se pueden hacer los recorridos. Luego de definir el nombre de la restricción y acotar superiormente por la capacidad del contenedor, se debe indicar el coeficiente que multiplica a cada una de las variables que forman parte de la restricción. En el caso de la restricción (15), las únicas variables que conforman esta restricción son las y_{it}. Para identificarlas en el código se decidió crear un conjunto con la unión de todos los conjuntos de variables. Este conjunto es definido de tal forma que tiene un orden en el que los primeros elementos corresponden a las variables x_{ijt}, seguidas por y_{it}, q_{it}, z_t y u_{it} respectivamente. Esto quiere decir que los primeros $E \times E \times T$ elementos del conjunto son las variables x_{ijt}, los siguientes $E \times T$ elementos son las variables y_{it} y así con el resto de variables. Para definir los coeficientes que multiplican a cada variable, en cada restricción, se utiliza un *contador.* Este indica la cantidad de coeficientes en la restricción que son distintos de *0* y, además, se definen dos *arrays* (*indi* y *mat*) que guardan los valores de los coeficientes (*1* en el caso de la Figura 6) y la ubicación de la variable correspondiente en el conjunto. De esta forma, se cargan los dos arrays y la cantidad de variables diferentes de cero (*contador -1)* en *glp_set_mat_row*.

```
for (int res = 1; res <=  T; res++){
    string nombre = "restriccion" + to_string(res);
    glp_set_row_name(lp, res, "r1");
    glp_set_row_bnds(lp, res, GLP_UP, 0.0, C);
    contador=1;
    for (int var = 1; var <= cantvar; var++){
        if ((E*E*T + E*(res-1)< var) && (var <= E*E*T+E*res)) {
            indi[contador]=var;
            mat[contador]=1.0;
            contador++;
        } else {
        }

    }
    glp_set_mat_row(lp, res, contador-1, indi, mat);

}
```

Figura 6 - Definición de restricciones

Una vez escrito el modelo, el mismo fue validado a través de la implementación de las mismas pruebas ejecutadas en la Sección 4.2 a través del comando *glp_intopt* mencionado. En esta etapa se identificaron los desafíos previamente mencionados y los mismos fueron superados a partir del aprendizaje a través de la guía GLPK de Makhorin (2009).

5.2 Elección de heurísticas

En esta sección se detalla cuales heurísticas fueron seleccionadas para la implementación y el motivo de su elección. Para la resolución del problema planteado se optó por el uso de *Feasibility Pump* (Bertacco *et al.*, 2007), *Local Branching* (Huang *et al.*, 2023) y *Objective*

Scaling Ensemble Approach (Zhang & Nicholson, 2019). Se tomó la decisión de implementar una heurística de solución inicial, *Feasibility Pump*, y dos heurísticas de mejora como *Local Branching* y *OSEA.* Analizar el comportamiento de dos heurísticas de mejora ayuda a observar mejoras y debilidades de cada una. Además, como se observó en la Sección 2.4, las heurísticas planteadas para VRP están basadas en la obtención de una solución inicial para luego mejorarla. La principal diferencia con la literatura consultada es que en este documento se plantean métodos generales para diversos problemas, sin utilizar estructuras y datos específicos del VRP.

De esta forma se analizan los métodos implementados para problemas MILP, sin el aprovechamiento de características específicas de problemas de ruteo. Las soluciones factibles obtenidas mediante el método *FP* serán mejoradas por *LB* u *OSEA,* analizando los resultados en calidad de la solución y tiempo de ejecución de cada una. Particularmente se seleccionó *FP* debido a ser una heurística muy utilizada en la literatura, capaz de obtener una solución de manera eficiente y fácil de implementar (Berthold *et al.*, 2018). Por otro lado, los otros métodos son destacados en el Apéndice I: Estado del arte como métodos eficientes y rápidos. Además, en el documento se destaca que una de las aplicaciones del método *OSEA* son los problemas VRP, debido a que la mayoría de las variables tomarán el valor nulo ya que son las variables binarias.

Se escoge la heurística *LB* por la importancia adjudicada en la literatura consultada y ser un método *neighborhood*, alineado así a la tendencia del tipo de heurísticas utilizadas en problemas de ruteo. Por otro lado, la heurística OSEA, método de fijación, es elegida por ser especialmente aplicable a problemas de ruteo y permitir la comparación entre heurísticas de distintos métodos. A continuación se detalla cada una de las heurísticas y se detalla su implementación.

5.3 Implementación de Feasibility Pump

FP es un método de solución inicial, lo que quiere decir que su objetivo es determinar una solución factible a los problemas MILP. Además, es un método general que no utiliza ninguna característica del problema que resuelve, sino que se puede aplicar a cualquier tipo de problema. La metodología de *FP* se basa en la obtención de dos trayectorias de puntos, idealmente convergentes. La primera trayectoria se genera mediante el redondeo de la parte discreta de la solución obtenida de resolver el problema MILP relajado (permitir que las variables discretas sean continuas). Esta solución redondeada, sin embargo, no garantiza estar dentro de la región factible del MILP. La segunda trayectoria se obtiene resolviendo un nuevo problema lineal, cuyo objetivo es encontrar el punto dentro de la región factible relajada que minimice la distancia al punto generado en la primera trayectoria. El proceso se repite de manera iterativa hasta que, potencialmente, ambas trayectorias convergen en un mismo punto. El método ha demostrado ser muy eficaz en la resolución de problemas MILP, por lo que diferentes autores a lo largo del tiempo han variado el método creando nuevas heurísticas y expandiendo el ámbito de ejecución (Mexi *et al.*, 2023; Berthold *et al.*, 2018).

Específicamente *FP* comienza con la resolución del problema original al que se relajan las variables discretas. Al resultado obtenido, denominado x^*, se redondea mediante la función de redondeo escalar ($[\cdot]$) al valor entero más cercano. Se aclara que este redondeo solo se realiza a las variables originalmente discretas. Para la obtención del primer punto de la segunda

trayectoria se crea un nuevo problema lineal. En este se busca un punto que cumpla estar dentro de la envolvente convexa del problema original relajado y que sea lo más cercano posible a [*x**] (a partir de ahora, $[x\ *] = \tilde{x}$). Se determina un nuevo problema LP en (34), donde se minimiza la función de distancia (Δ) como la norma L_1 .

$$min\ \{\Delta(x, \tilde{x})\ :\ Ax \geq b\} \qquad (34)$$

Se define la norma L_1 de $\tilde{x}$ a un punto genérico $x \epsilon \{x \epsilon \mathbb{R}^n\ /\ Ax \geq b\}$ como:

$$\Delta(x, \tilde{x}) = \sum_{j \epsilon J} \left| x_j - \tilde{x}_j \right| \qquad (35)$$

Se aclara que *J* es el conjunto de variables discretas de la formulación genérica, por lo que cuando hablamos de la distancia entre los puntos se trabaja con las variables discretas. Esto es así ya que el resto de variables no necesitan ser redondeadas y, por lo tanto, no se toman en cuenta en la función distancia $\Delta(x, \tilde{x})$. Hay varios detalles que se deben de tener en cuenta con esta formulación, en primer lugar se observa como la norma L_1 no es una función lineal, por lo que lo representado en (34) no se trata de un problema MILP. Para poder tratarlo como tal se observa que para cualquier $\tilde{x}$ se puede escribir la función distancia como:

$$\Delta(x, \tilde{x}) = \sum_{j \epsilon J: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \epsilon J: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \epsilon J: l_j < \tilde{x}_j < u_j} d_j \qquad (36)$$

En (36) se define l_j como la cota inferior y u_j la cota superior de la variable x_j con $j \epsilon J$ (permitiendo los casos no acotados, $l_j = -\infty$ y/o $u_j = \infty$). Se define que las variables d_j deben cumplir una serie de nuevas restricciones que se deben de sumar al modelo con función objetivo (34), obligando que a través de la minimización de la distancia tomen el valor absoluto,estas son:

$$d_j \leq x_j - \tilde{x}_j \text{ y } d_j \geq \tilde{x}_j - x_j \qquad \forall j \epsilon J: l_j < \tilde{x}_j < u_j \qquad (37)$$

Con esta precaución se logra representar la norma L_1 quitando los valores absolutos, permitiendo así la linealidad de la función. Este cambio no es gratuito en términos de la complejidad del problema, pues incrementa el número de variables y restricciones del problema. Tomando el modelo general MILP de la Sección 2.1 con los cambios propuestos en (36) y (37), se obtiene la forma de determinar el punto más cercano a $\tilde{x}$ como una formulación de problema MILP que es relajado:

$$min\ \{\Delta(x, \tilde{x})\ :\ \overline{A}x \geq \overline{b}\} \qquad (38)$$

Resumiendo, el método comienza con la obtención de un punto discreto $\tilde{x}$ al redondear la solución del problema relajado. Luego, se busca hallar un nuevo punto x^* minimizando la distancia con el discreto $\tilde{x}$ y resolviendo el problema *LP* (38). Si se encuentra una solución al problema pero se cumple que $\Delta(x^*, \tilde{x}) \neq 0$ se redondea el valor de x^* y se obtiene un nuevo $\tilde{x}$, repitiendo el proceso de forma iterativa. El método culmina cuando $\Delta(x, \tilde{x}) = 0$ donde ambos

puntos son iguales, logrando obtener un punto discreto que cumple las restricciones lineales. En otro caso culmina el método por algún tipo de criterio de terminación (tiempo o cantidad de iteraciones).

Se decide la elección de este método para incorporar una heurística fundacional que puede aplicarse a cualquier problema MILP. A su vez, se había determinado que *Feasibility Pump* logró ser muy eficiente en soluciones para problemas binarios difíciles y de gran tamaño (Fischetti *et al*., 2005). Para profundizar en el funcionamiento de este método se presenta en la Figura 7 el pseudocódigo de la implementación de la heurística *FP*. Al algoritmo se le debe ingresar un conjunto de entradas para su funcionamiento, estas son: *Maxiter,* que determina la máxima cantidad de veces que se repite el proceso de búsqueda, *A*, la matriz de restricciones del problema que se quiere resolver y *b*, el vector de variables independientes. Se define *c*, como el vector de coeficientes de la función objetivo, *T*, parámetro para definir la cantidad de variables a modificar en caso de detectar cycling y *KK*, cantidad de iteraciones a partir de la cual se evalúa la posibilidad de aplicar *restart*. Estos parámetros ayudan a definir el problema que se quiere resolver y también modifican la calidad y el tiempo de la resolución del método.

El funcionamiento de la heurística *FP* incluye elementos estocásticos que contribuyen a evitar ciclos o estancamientos. Para esto se hace uso de herramientas de aleatoriedad brindadas por el lenguaje de C++ para el uso de números pseudoaleatorios. Como se muestra en la línea 4 del pseudocódigo, se debe inicializar una semilla que luego es utilizada para la generación de los números pseudoaleatorios. Guardar el valor de la semilla es clave para poder volver a realizar el mismo experimento y poder analizar el funcionamiento y sus resultados. A nivel de código, se agrega la línea *#include<random>* al comienzo del código para acceder a las funcionalidades de la biblioteca estándar para generación de números aleatorios. Luego, se debe añadir *std::random::random_device rd* donde se crea un generador de números random llamado *rd* el cual es utilizado para generar una semilla. Con una semilla encontrada, se crea el generador de números pseudoaleatorios con *std::mt19937 gen(semilla)* desde el cual se van a obtener las diferentes distribuciones aleatorias.

El método posee dos criterios de terminación, uno al alcanzar una solución inicial y el otro por pasarse de cantidad máxima de iteraciones, como se observa en la línea 6 del pseudocódigo. Se utilizó una cantidad máxima de 10.000 iteraciones debido a que es el valor utilizado en Bertacco *et al*., (2007). Utilizando la solución de la iteración anterior, se crea la matriz de restricciones $\overline{A}$ y el vector de valores independientes $\overline{b}$ necesarios para la creación de (38). A nivel de código, estos cambios causan la incorporación de un cambio de variables que son necesarias para la implementación de la heurística. Esto último es así porque la función objetivo es creada a través de la definición de las variables, como se pudo observar en la Figura 5, imposibilitando el uso de directo de la forma (36). Se crean tantas variables como variables discretas y se distinguen 3 tipos de nuevas variables: cuando $\tilde{x}_j = l_j$, $\tilde{x}_j = u_j$ o $l_j < \tilde{x}_j < u_j \ \forall j \epsilon J$. Los dos primeros casos se dan cuando el valor de las variables discretas al redondear son iguales a algunas de sus cotas. En estos caso, se realiza un cambio de variable de forma que: $v_j = u_j - x_j$ y $z_j = x_j - l_j \ \forall j \epsilon J$. De esta forma se cambia la función distancia de (36) por:

$$\Delta(x, \tilde{x}) = \sum_{j \epsilon J : \tilde{x}_j = l_j} z_j + \sum_{j \epsilon J : \tilde{x}_j = u_j} v_j + \sum_{j \epsilon J : l_j < \tilde{x}_j < u_j} d_j \qquad (39)$$

Se debe cambiar el modelo para reflejar estos cambios, por esto se crean dos nuevos tipos de restricciones:

$$v_j = u_j - x_j \quad \forall j \epsilon J: \tilde{x}_j = u_j \tag{40}$$

$$z_j = x_j - l_j \quad \forall j \epsilon J: \tilde{x}_j = l_j \tag{41}$$

Por otro lado, se generan las variables d_j como se explicó en (37). De esta forma, utilizando el problema de optimización de la ecuación (34) con función objetivo la minimización de (39) y añadiendo las restricciones (37), (40) y (41) a las restricciones originales se obtiene el modelo relajado a resolver. Al resolver este problema de optimización, se obtiene el valor objetivo *x**, que en caso de ser igual a $\tilde{x}$ culmina el método exitosamente. En otro caso, se continúa con el procedimiento de actualización.

En caso de detectar un ciclo, esto quiere decir que $[x\ ^*] = x$, entonces se procede al paso estocástico. Este, referido en la línea 15 del pseudocódigo, tiene como objetivo modificar aquellas variables que perjudican en mayor medida a la función objetivo, y evitar entrar nuevamente en un ciclo. Para hacerlo, se definen los parámetros $\lambda_j = \left| x_j^* - \tilde{x}_j \right| \forall j \epsilon J$ que representan las distancias entre las variables x_j^* y el valor de las $\tilde{x}_j$. Luego se seleccionan un conjunto de *TT* variables discretas, siendo *TT* un número aleatorio. A aquellas variables del conjunto que cumplan $x_j^* > \tilde{x}_j$ se suma *1* al valor de $\tilde{x}_j$ para la próxima iteración, y si $x_j^* < \tilde{x}_j$, se le resta *1*. La cantidad de variables a modificar (*TT*) está dada por una función de distribución uniforme con límites $\frac{T}{2}$ y $\frac{3T}{2}$, donde *T* es un parámetro de entrada del modelo. Para determinar su valor se experimentó variando el mismo y se definió en *20*.

A partir de la línea 16 del pseudocódigo, se observa otro mecanismo del método para evadir mínimos locales. Para determinar su aplicación se tiene en cuenta dos criterios, si luego de *KK* iteraciones no se logra una mejora de la función distancia de al menos de un 10 % o si se encuentra la misma solución a la encontrada en iteraciones pasadas. Si alguno de los criterios se cumple se modifican las entradas con una probabilidad *p* representada por la distancia al entero más cercano. Para modelarlo se recurrió a una variable aleatoria *w*, con una distribución uniforme entre *0* y *1.* En los casos donde $w_j \leq p\ \forall j \epsilon J$ se modifica la entrada como se observa en la línea 18 y no en otro caso. Es de destacar que si bien se explica el método de forma general, en nuestro caso no tenemos variables discretas no binarias, por lo que este último paso implica un *flip* de la variable seleccionada y las variables d_j no son creadas.

Feasibility Pump

Procedimiento FEASIBILITY PUMP ($Maxiter$, A, b, c, T, KK)
Inicializar x^*:= argmin$\{c^T x : Ax \geq b\}$ (Resolución del problema relajado)
Redondeo de la solución relajada $\bar{x} := [x^*]$
Inicializar semilla (generación de números pseudo aleatorios)
iteración := 0
while ($\Delta(x, \bar{x}) > 0$ **and** iteración $\leqslant$ Maxiter) **do**
  iteración:=iteración+1
  se actualiza $\bar{A}$ y $\bar{b}$
  $x^* := argmin\{(\Delta(x^*, \bar{x}) : \bar{A}x \geq \bar{b}\}$
  if $\Delta(x^*, \bar{x}) > 0$ **then**
    if $[x_j^*] \neq \bar{x}_j$ para al menos un j ϵ $\mathcal{J}$ **then**
      Actualizar $\bar{x} := [x^*]$
    else
      Para cada j ϵ $\mathcal{J}$ se define $\lambda_j := |x_j^* - \bar{x}_j|$
      Modificar los TT=unif(T/2,3T/2) componentes de $\bar{x}$ con mayor λ_j
      if $(0.9 \times \Delta(x^*_{\text{iteración}-KK}, \bar{x}_{\text{iteración}-KK}) \leq \Delta(x^*_{\text{iteración}}, \bar{x}_{\text{iteración}}))$ **or** $(\exists t \in [0; \text{iteración} - 1] \ / \ x^*_{\text{iteración}} = x^*_t)$ **then**
        w:=unif(0, 1)
        $\bar{x}_j := 1 - \bar{x}_j$ con una probabilidad de p $\forall j \epsilon \mathcal{I}$
      end if
    end if
  end if
end while
Fin

Figura 7 - Pseudocódigo de la heurística Feasibility Pump

La principal ventaja al utilizar *FP* es la de evitar expresiones de problemas MILP, pasando a resolver únicamente problemas LP, los cuales generalmente son más fáciles de resolver (Spielman & Teng, 2004). De todas formas, es un método que al aumentar la cantidad de variables y restricciones es capaz de enlentecer los tiempos del método comparado con la relajación del problema MILP. También se cree importante agregar que, si bien *FP* permite la solución de problemas generales MILP, funciona especialmente bien en problemas con variables binarias. Esto se debe a que, en los problemas con variables enteras generales (no binarias), las variables pueden tomar una mayor cantidad de valores posibles, lo que amplía el espacio discreto de búsqueda en comparación con el caso binario. Como consecuencia, en los problemas MILP con variables no binarias, es más probable que el método caiga en ciclos o tenga mayores dificultades para converger.

5.4 Implementación de OSEA

OSEA es una heurística de mejora que se basa en que sólo una pequeña proporción de las variables de las soluciones de los problemas de programación entera son distintas de cero (Zhang & Nicholson, 2019). En el caso de los VRP suele ser cierto ya que existen muchas posibilidades o rutas para ir de un punto a otro pero sólo una será escogida. Específicamente, de 20143 variables discretas del modelo MILP del PTIC, 20143 toman el valor *0*, un *99.7%.* Por tanto, la idea del método es eliminar cierta cantidad de variables fijándolas a cero para así reducir el tamaño del problema. Para ello, se toma un grupo de soluciones factibles del problema y se eliminan aquellas variables que en todas estas soluciones toman el valor nulo. El grupo de soluciones seleccionadas puede obtenerse de distintos métodos, destacando las obtenidas a partir de la relajación del problema por su facilidad de resolución.

Un posible problema con la idea presentada es que las soluciones factibles obtenidas para la posterior fijación de variables sean de mala calidad. En este caso, existe un alto riesgo de fijar variables que perjudican la función objetivo. Para mitigar este riesgo lo que se propone es evaluar el mismo problema con variaciones en los coeficientes de la función objetivo. Haciendo esto, en Zhang & Nicholson (2019) argumentan que si una variable toma cero en la solución, aún cuando el costo asociado (coeficiente de la función objetivo) se redujo a fracciones del costo original, entonces dicha variable probablemente no sea útil en el problema. Además, proponen que el costo absoluto asociado a una variable no es tan importante como el costo relativo respecto a otras. En ello basan el método iterativo por lo que se fundamenta que sólo se están eliminando las variables que no aportan a la optimalidad de la solución.

En la Figura 8 se muestra el pseudocódigo de la heurística. Como primer paso, en la línea 2 se crea el parámetro $M \gg 2$ que será utilizado para la inicialización de los costos de la función objetivo en fracciones pequeñas de los originales. *M* se define como la sumatoria de todos los costos de la función objetivos que están asociados a las variables discretas. En la línea 3 del pseudocódigo se definen los parámetros de la función objetivo que se utilizan en la primera iteración del método. Estos van a generar un nuevo problema al que se le cambia la función objetivo con los nuevos costos hallados. Estos nuevos coeficientes se utilizan para permitir el aumento relativo del costo de las variables discretas en las siguientes iteraciones. El nuevo problema es relajado y se resuelve en cada iteración en la línea 6 del pseudocódigo dentro del ciclo de máximo $N_{máx}$ iteraciones propuesto en la línea 5.

Entre las líneas 7 y 10 se puede observar la fracción del código que procura la selección de las variables a eliminar. En estas se modifica el costo de las variables en la función objetivo iterativamente. Se evalúa cada una de las variables de la solución del modelo y para cada variable positiva se aumenta el costo en la función objetivo, pues $\frac{c_j}{x_j^{iterOSEA-1}+1} \geq \frac{c_j}{M}$ para el primer caso. Para las siguientes iteraciones, al haber aumentado el costo anterior, se esperaría que $x_j^{iterOSEA} \leq x_j^{iterOSEA-1}$. Por otro lado, si la variable toma valor nulo no se modifica el costo, implicando un aumento relativo a las demás variables. De esta forma, se obtienen nuevos costos en cada iteración definidos con la solución del problema relajado. Los valores de la siguiente iteración se actualizan como se muestra a continuación:

$$\overline{c_j^{iterOSEA}} = \frac{c_j}{x_j^{iterOSEA-1}+1} \quad (42)$$

Con esta definición los nuevos coeficientes varían cuán atractivas son las variables para la función objetivo. Una variable es atractiva cuando es preferible para la función objetivo seleccionar esta variable en vez de otra. Cuando el resultado de una variable x_j tienda a cero, el coeficiente no se modifica. El ajuste dinámico de las variables ayuda en la obtención de distintas soluciones enteras evitando la eliminación de variables poco atractivas para el problema original.

En el caso de que la solución encontrada entre una iteración y la siguiente sea la misma, no tendrá sentido permanecer en el ciclo de la línea 5 del pseudocódigo, por lo que se sale del mismo como se presenta en la línea 13. De otra forma, el proceso iterativo terminará por el parámetro de control $N_{máx}$. Este parámetro de entrada de la heurística es clave, pues para un $N_{máx}$ grande la cantidad de soluciones analizadas para la posterior fijación de variables será grande. Potencialmente en tal caso se fijarán pocas variables, lo que provocaría una baja limitación del espacio de búsqueda para su uso práctico. Por otro lado, si la cantidad $N_{máx}$ es pequeña se dice que las soluciones no son lo suficientemente diversas. Esto, potencialmente, causa que el espacio de búsqueda esté muy acotado por una fijación excesiva de variables y no permita la obtención de una buena solución al problema MILP original. De esta forma, la definición de $N_{máx}$ es clave y su valor fue determinado mediante la variación del mismo y observación de los resultados.

Una vez se culmine el proceso iterativo se procede con la creación del conjunto de soluciones de los diferentes problemas relajados. Es un conjunto de tamaño máximo *p* en donde se obtienen soluciones no necesariamente factibles al problema MILP original. Estas se utilizan para la obtención de las variables a fijar y se define el conjunto *S* cuyos elementos son las soluciones $x^{iterOSEA}$ con $iterOSEA \epsilon P$ tal que $P = \{p \epsilon \mathbb{N}: 1 \leq p \leq N_{máx}\}$. Ya que el conjunto puede contener soluciones no factibles del MILP original, es posible fijar un conjunto de variables que provoque que no existan soluciones factibles del MILP generado. Por este motivo se incluye una solución factible al conjunto *S*, la cual es la obtenida a través del método *FP* La inclusión de esta solución al conjunto *S* asegura que el método *OSEA* sea capaz de encontrar una solución factible del problema original que en el peor de los casos sea la misma que se utilizó como solución factible incluida en *S*. Con esto en cuenta se define *S* como $S = x^0, x^1, \ldots, x^p$ con $p \epsilon P$.

OSEA

Procedimiento OSEA(N_{max}, x^0, MILP)
Inicializar valor de $M := M \leftarrow \Sigma_{j\epsilon\mathcal{I}}|c_j|$
Inicializar valor de $c_j^0 := M \leftarrow \frac{c_j}{M}\ \forall j\epsilon\mathcal{I}$
iteración iter=0
while $n \leq N_{max}$ **do**
  Se resuelve el problema relajado y se obtiene:
  $x^{iter} := argmin\{c_{iter}^T x : Ax \geq b\}$
  if $x_j^{iter} > 0\ \forall j\epsilon\mathcal{I}$ **then**
    $c_j^{iter} := \frac{c_j}{x_j^{iter-1}+1}$
  else
    $c_j^{iter} = c_j^{iter-1}$
  end if
  if $c_j^{iter} = c_j^{iter-1}\ \forall j\epsilon\mathcal{I}$ **then**
    Break
  end if
  iter:=iter+1
end while
Sea el conjunto $S = x^0, x^1, ..., x^p$, $p\epsilon\mathcal{P}$ con $\mathcal{P} = \{p\epsilon\mathbb{N} : 0 \leq p \leq N_{max}\}$
if $x_j^p = 0, j\epsilon\mathcal{I}, \forall p\epsilon\mathcal{P}$ **then**
  Fijar la variable x_j en el problema MILP original
end if
Resolver el problema con variables fijadas $x_{OSEA} := argmin\{c^T x : \bar{A}x \geq \bar{b}\}$
$z_{OSEA} := c^T x_{OSEA}$
Fin

Figura 8 - Pseudocódigos de la heurística OSEA

Una vez se defina *S* se procede a fijar en el problema original toda variable que sea cero para todos los elementos de *S*. Esto es:

$$x_j = \begin{cases} \text{Fijar a 0} & \text{si } s_j = 0 \forall s\epsilon S \\ \text{No hacer nada} & \text{en otro caso} \end{cases}$$

Ya con las variables fijadas se resuelve el problema MILP generado, esperando que la cantidad de variables fijadas cause una disminución en los tiempos de resolución y obtener una mejora a la solución actual.

5.5 Implementación de Local Branching

La heurística *Local Branching* es una técnica iterativa diseñada para mejorar la búsqueda de soluciones en problemas de optimización binaria, introducida en Fischetti & Lodi (2003). Esta es una heurística de mejora basada en la selección de un vecindario en el cual sea posible mejorar las soluciones halladas. Por tanto, se busca un vecindario que, al resolver el problema MILP, en este se pueda encontrar una mejor solución a la previamente obtenida. En particular, en esta heurística, el vecindario estará definido por las potenciales soluciones que sólo difieren en una cantidad acotada de entradas o variables. Partiendo de la solución factible inicial, iterativamente un conjunto de variables serán seleccionadas para ser destruidas. Destruir, dentro de un problema MILP, se refiere a fijar todas las variables, exceptuando aquellas que pertenezcan al conjunto seleccionado para destruir. Al fijar el resto de las variables y resolver el problema se optimiza un problema MILP reducido del problema original, únicamente teniendo en cuenta el conjunto de variables previamente destruidas.

Por lo tanto, el objetivo de la heurística es mejorar la solución encontrada en tiempos de cómputo acotados al explorar una región limitada. Para ello se acota el espacio factible alrededor de la solución previamente obtenida y se encuentra un conjunto de variables a destruir. En esta región potencialmente se encontrarán mejores soluciones en menores tiempos de cómputo que resolviendo el problema completo. Para definir la vecindad se crean nuevas restricciones que fijan la cantidad de variables que pueden diferir de la mejor solución encontrada hasta el momento. Esta se basa en la distancia de Hamming para variables binarias (Huang, *et al.*, 2023), definida como:

$$\sum_{i \epsilon S_1} (1 - x_i) + \sum_{i \epsilon S_0} x_i \leq k \qquad (43)$$

En esta función S_0 y S_1 representan los conjuntos de las variables binarias que toman el valor *1* o *0* respectivamente en la solución alrededor de la cual se construye la vecindad. El parámetro *k* representa un ajuste del tamaño de la vecindad, pues implica cuántas variables pueden ser diferentes de la solución actual. Con la región factible definida se procede a resolver el problema MILP con las restricciones originales y agregando las restricciones observadas en la ecuación (43). Aunque la región factible sea reducida se espera que del problema se obtenga una solución mejorada del problema original. Sin embargo, es posible que el tamaño no sea lo suficientemente grande para mejorar la solución actual, por lo tanto en el método se propone un ajuste del parámetro *k* para escapar de estos óptimos locales del método de búsqueda.

Para un mayor entendimiento y detalle se muestra en la Figura 9 el pseudocódigo de la heurística. En la línea 1 se definen los parámetros necesarios. Estos son parámetros de control del tamaño de vecindario k_T, α, β, la solución actual x^0 y el parámetro de terminación del problema *Maxiter.* k_T es el tamaño inicial utilizado para el vecindario y α y β son dos parámetros utilizados para variar el valor de *k* de una iteración a otra de ser necesario. Además se genera un número aleatorio de semilla en la línea 4 ya que el método tiene una componente estocástica. En la línea 6 comienza el procedimiento iterativo y en la 7 se construyen las restricciones descritas en (43), definiendo la nueva región factible del problema binario. El método culminará una vez que se hayan alcanzado la cantidad de *Maxiter* iteraciones sólamente. Una vez se añaden las nuevas restricciones al problema original, en la línea 8 se resuelve el nuevo problema MILP relajado. Esto se realiza para definir cuales son las variables

que se desean destruir y se considera un método *greedy* ya que de esta optimización local se espera obtener una global. Con esta solución relajada se construye un parámetro Δ_i, definido como el valor absoluto de la diferencia entre las entradas de la solución actual y la solución anterior, tal como se observa en la línea 9. Luego se define el conjunto $\overline{\mathcal{X}^{iter}}$ para la iteración *iter* cuyos elementos son las entradas de las variables que cumplen que $\Delta_i > 0$. Esto es:

$$\overline{\mathcal{X}^{iter}} := \left\{ i:\ \Delta_i > 0,\ i \epsilon J \right\} \qquad (44)$$

A través de este criterio se buscará destruir las variables en donde la diferencia entre la solución inicial y la relajada es mayor, buscando la creación de vecindarios con potencial a encontrar mejores soluciones. Según el tamaño del conjunto de $\overline{\mathcal{X}^{iter}}$ se diferencian casos para continuar con la heurística. En caso de que el tamaño sea mayor o igual a k^{iter} entonces hay suficientes variables para destruir y se selecciona aquellas entradas cuyo Δ_i sea el más grande como se observa en la línea 10. En casos que varias entradas de la variable tengan un mismo Δ_i, entonces se decide tomar un criterio estocástico seleccionando alguna de las opciones con probabilidad uniforme.

En otro caso, el tamaño del conjunto $\overline{\mathcal{X}^{iter}}$ es menor al tamaño del vecindario k^{iter}, por lo tanto no hay suficientes variables para destruir. El criterio de selección será tomar las primeras entradas $\left|\overline{\mathcal{X}^{iter}}\right|$ pertenecientes al conjunto. Luego se tomará la cantidad necesaria de variables de forma aleatoria entre las que el valor Δ_i es igual a cero, tal como se observa en la línea 13. Por lo tanto, se define el conjunto $\mathcal{X}^{*iter}$ tal que:

$$\overline{\mathcal{X}^{*iter}} := \left\{ i:\ \Delta_i = 0,\ i \epsilon J \right\} \qquad (45)$$

De este conjunto se toman de forma aleatoria $k^{iter} - \left|\overline{\mathcal{X}^{iter}}\right|$ elementos los cuales se añaden a un nuevo conjunto $\mathcal{X}^{iter}$ que se define de la unión de estos elementos con $\overline{\mathcal{X}^{iter}}$ como se observa en la línea 14. De igual manera se van a destruir las variables de las entradas pertenecientes al conjunto $\mathcal{X}^{iter}$, como se muestra en la línea 16. Una vez identificadas las variables a destruir se procede a resolver el problema MILP fijando el resto de variables, obteniéndose el resultado $\tilde{x}^{iter+1}$ como se observa en la línea 17.

En caso de obtener una mejor solución del problema MILP comparada con la solución de la iteración anterior, entonces se actualiza el valor de la solución y se restablece el valor *k* en caso de que se haya cambiado. En caso de que la solución empeore o sea igual que su antecesor, entonces se define un nuevo tamaño de vecindario k^{iter} según un parámetro α, aumentando el valor de k^{iter} multiplicándolo por $\alpha > 1$. Además, se busca acotar el valor de k^{iter} según el parámetro β, con $\beta < 1$, multiplicado a la cantidad de variables binarias del problema MILP con el objetivo de hacer que el problema a resolver no sea tan grande y difícil. Por lo tanto, se define k^{iter} como:

$$k^{iter+1} = min\{\alpha \cdot k^{iter}, \beta \cdot |J|\} \qquad (46)$$

Esta forma de definir k^{iter+1} ayuda a evitar quedarse estancado en mínimos locales al expandir el espacio de búsqueda del vecindario de ser necesario.

Local Branching

Procedimiento LOCAL BRANCHING(k_T, x^0, α, β, Maxiter, MILP)
Sea el conjunto $\mathcal{P}$ con $\mathcal{P} = \{p \epsilon \mathbb{N} : 0 \leq p \leq Maxiter\}$
Inicializar iter=0 con iter $\epsilon \mathcal{P}$
Inicializar semilla (generación de números pseudo aleatorios)
Inicializar mejor resultado $x^* = x^0$
while iteración $\leq$ Maxiter **do**
  Añadir al problema nuevas restricciones (Hamming Ball):
  $\sum_{i \epsilon [n]: x_i^t = 0} x_i^{t+1} + \sum_{i \epsilon [n]: x_i^t = 1} (1 - x_i^{t+1}) \leq k_{iter}$
  Se resuelve el problema relajado y se obtiene $\tilde{x}^{iter+1} := argmin\{c^T x : \tilde{A}x \geq \tilde{b}\}$
  Se define $\bar{\mathcal{X}}^{iter} := \{x_i : \Delta_i > 0, i \epsilon \mathcal{N}\}$ con $\Delta_i = |\tilde{x}_i^{iter+1} - x_i^{iter}| \; \forall i \epsilon \mathcal{N}$
  if $|\bar{\mathcal{X}}^{iter}| \geq k^{iter}$ **then**
    Se toman k_{iter} elementos de $\bar{\mathcal{X}}^{iter}$ con Δ_i más alto creando un nuevo $\mathcal{X}^{iter}$
  else
    Se toman $k^{iter} - |\mathcal{X}^{iter}|$ elementos de forma random de $\{x_i : \Delta_i = 0, i \epsilon \mathcal{N}\}$ llamando al conjunto $\mathcal{X}'$
    Se crea $\mathcal{X}^{iter} = \bar{\mathcal{X}}^{iter} \cup \mathcal{X}'$
  end if
  Se destruyen las variables pertenecientes a $\mathcal{X}^{iter}$ en el problema orginal
  Se resuelve el MILP y se obtiene $x^{iter+1} := argmin\{c^T x : \hat{A}x \geq \hat{b}\}$
  if $\tilde{x}^{iter+1} > x^*$ **then**
    Se actualiza el valor del mejor resultado $x^* = x^{ieracion+1}$
    $k_{iter} = k_T$
  else
    Se varía $k_{iter} = min\{\alpha \times k_{iter}, |\mathcal{I}| \times \beta\}$
  end if
  iter=iter+1
end while
Se devuelve el mejor valor x^* obtenido
Fin

Figura 9 - Pseudocódigo de la heurística Local Branching

Una vez que se llegue a la máxima cantidad de iteraciones entonces el método culmina con la mejor solución obtenida.

5.6 Implementación con Gurobi y SCIP

En esta sección se introduce la implementación de otros paquetes de *software* para la resolución de problemas MILP. Con esta se desea comprobar la diferencia en calidad de soluciones encontradas por paquetes comerciales y por paquetes no comerciales. Dentro de los paquetes de *software* vistos en la Sección 2.2 se escogieron los paquetes SCIP y Gurobi para la comparación. Gurobi fue seleccionado como representante de los paquetes de *software* comerciales más avanzados mientras que SCIP fue seleccionado como el paquete no comercial más utilizado. La comparación tiene como objetivo evaluar la calidad de las soluciones obtenidas mediante el solver GLPK, además, el caso de aplicación escogido es el problema VRP del Parque Industrial del Cerro. En resumen se busca determinar en qué medida las soluciones obtenidas con GLPK se aproximan a aquellas generadas por herramientas más avanzadas y robustas como SCIP y Gurobi.

5.6.1 Optimizador de Gurobi

En primera instancia se eligió Gurobi como el paquete de *software* comercial a ser utilizado. Como se observó en la Sección 2.2, junto a XPRESS y CPLEX, es uno de los paquetes más eficientes y se encontró facilidad para la adquisición de una licencia gratis académica. Para la correcta instalación y validación de la licencia existen instructivos actualizados en el portal de usuario de Gurobi (Gurobi Optimization, LLC, s.f.-e; Gurobi Optimization, LLC, s.f.-a). Esta licencia permite el acceso completo al optimizador sin restricciones de funcionalidades ni de tamaño de los problemas.

Completada la instalación se tiene acceso a bibliotecas de distintos lenguajes de programación. Además, con el paquete existe la posibilidad de resolver el problema a través de una línea de comando directa (Gurobi Optimization, LLC, s.f.-b), análoga a la utilizada para la resolución con GLPK observada al principio de la Sección 5.1. Sin embargo, a modo introductorio del uso de la biblioteca de Gurobi en C++ se genera un código en el lenguaje C++ de la resolución de los problemas, pudiendo controlar los métodos de resolución. Los parámetros de control incluyen, por ejemplo, las heurísticas utilizadas, planos de corte, etcétera. La utilización de esta biblioteca no posee soporte para leer problemas a través de archivos .mod y .dat ya que estos son específicos para GLPK en el lenguaje C++. Por lo tanto, para procesar el problema aprovechando la estructura generada, existen líneas de código para la transcripción del mismo al formato .lp. Esta línea de código se presenta a continuación:

```
glpsol --math problema.mod --data datos.dat --wlp problema.lp
```

Una vez obtenido el problema en formato .lp se procede a la formulación del código para la resolución de problemas MILP como el presentado en la Figura 10. En este código se muestra cómo cargar el problema y ejecutar los comandos de resolución del mismo. La solución encontrada es ingresada en un archivo .log, análogo a archivos .sol en la ejecución en GLPK.

```cpp
#include "gurobi_c++.h"
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
int main(int argc, char *argv[]) {
if (argc != 2) { // Verifica que se haya pasado exactamente un argumento (el archivo .lp)
    cout << "Uso: " << argv[0] << " problema.lp" << endl;
    return 1;
}
try { // Crea un entorno de Gurobi sin iniciarlo todavía
    GRBEnv env = GRBEnv(true);
    env.set("LogFile", "gurobi.log");            // Configura el archivo de log donde se escribirá toda la salida del solver
    env.start();                                    // Inicia el entorno de Gurobi
    GRBModel model = GRBModel(env, argv[1]);        // Crea el modelo leyendo el archivo pasado por parámetro
    model.set(GRB_DoubleParam_TimeLimit, 1500.0); // Establece un límite de tiempo de 3600 segundos (1 hora) para la resolución
    auto start = high_resolution_clock::now();     // Inicializa cronómetro
    model.optimize();                               // Ejecuta la optimización del modelo
    auto end = high_resolution_clock::now();        // Para cronómetro
    double time_to_solution = duration_cast<duration<double>>(end - start).count(); // Calcula tiempo
    int status = model.get(GRB_IntAttr_Status);  // Obtiene el estado de la solución (por ejemplo, óptima, factible, sin solución, etc.)
    if (status == GRB_OPTIMAL || status == GRB_SUBOPTIMAL || status == GRB_TIME_LIMIT) {
        double objval = model.get(GRB_DoubleAttr_ObjVal);  // Obtiene el valor de la función objetivo
        cout << "Solución factible encontrada!" << endl;
        cout << "Valor objetivo: " << objval << endl;
        cout << "Tiempo hasta factible (total): " << time_to_solution << " segundos" << endl;
        GRBVar* vars = model.getVars(); // Obtiene todas las variables del modelo
        int numvars = model.get(GRB_IntAttr_NumVars);
        for (int i = 0; i < numvars; i++) { // Muestra los valores asignados a cada variable en la solución
            cout << vars[i].get(GRB_StringAttr_VarName) << " = " << vars[i].get(GRB_DoubleAttr_X) << endl;
        }
    } else {
        cout << "No se encontró solución factible." << endl;
    }
} catch (GRBException e) {
    cerr << "Error de Gurobi: " << e.getMessage() << endl;
    return 1;
} catch (...) {
    cerr << "Error desconocido durante la ejecución." << endl;
    return 1;
}
return 0;
}
```

Figura 10 - Implementación en código de Gurobi

Este código, con la información brindada en el mismo, es un ejemplo introductorio del uso de la biblioteca de Gurobi, estos comandos y todas las funciones disponibles se encuentran en la guía oficial de Gurobi en el entorno C++ (Gurobi Optimization, LLC, s.f.-f).

5.6.2 Optimizador de SCIP

En segunda instancia se implementó el paquete de *software* de SCIP (*Solving Constraint Integer Programs*) como el paquete de *software* de mejor rendimiento dentro de las opciones no comerciales, tal como fue observado en la Sección 2.2. El procedimiento de implementación es análogo al de Gurobi, sin la necesidad de obtener una licencia académica o paga. Nuevamente se hace utilización de la biblioteca ofrecida, en este caso por SCIP, a través de su Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) en el lenguaje de programación C++. Para la introducción a la implementación de comandos básicos se presenta el código generado en la Figura 11. Los procedimientos de búsqueda de la solución pueden ser controlados y la información necesaria se puede encontrar en la documentación oficial de SCIP (SCIP, s.f.).

```cpp
#include <iostream>
#include <string>
#include "SCIPOptSuite-9.2.1-Linux-ubuntu24/include/scip/scip.h"
#include "SCIPOptSuite-9.2.1-Linux-ubuntu24/include/scip/scipdefplugins.h"
#include "SCIPOptSuite-9.2.1-Linux-ubuntu24/include/soplex/exceptions.h"
using namespace std;
int main(int argc, char** argv)
{
if (argc != 2) {
cout << "Uso: " << argv[0] << " problema.lp" << endl;
return 1;
}
SCIP* scip = NULL;
SCIP_CALL( SCIPcreate(&scip) );
SCIP_CALL( SCIPincludeDefaultPlugins(scip) );


SCIP_CALL( SCIPreadProb(scip, argv[1], NULL) );

SCIP_CALL( SCIPsetRealParam(scip, "limits/time", 1500.0) );
//SCIP_CALL( SCIPsetIntParam(scip, "limits/solutions", 1) );
SCIP_CALL( SCIPsolve(scip) );

if (SCIPgetNSols(scip) > 0) {
SCIP_SOL* sol = SCIPgetBestSol(scip);
double objval = SCIPgetSolOrigObj(scip, sol);
double time_first = SCIPsolGetTime(sol);
cout << "Solucion factible encontrada!" << endl;
cout << "Valor objetivo: " << objval << endl;
//cout << "Tiempo hasta factible: " << time_first << " segundos" << endl;
SCIP_CALL( SCIPprintBestSol(scip, NULL, FALSE) );
} else {
cout << "No se encontro solucion factible." << endl;
}
SCIP_CALL( SCIPfree(&scip) );
return 0;
}
```

Figura 11 - Implementación en código de SCIP

En el código se configura un parámetro clave para el proceso de resolución que es el límite máximo de tiempo permitido para encontrar una solución. Luego se lanza el proceso de resolución del problema y si durante este proceso se encuentra una solución factible, el programa informa el valor objetivo de esa solución y el tiempo transcurrido hasta encontrarla. En el caso de que no se logre encontrar una solución factible dentro del límite de tiempo establecido, el programa comunica esta situación al usuario. Finalmente, se realiza la liberación de los recursos utilizados por SCIP antes de finalizar la ejecución.

6. Análisis de resultados

En esta sección se presentan los resultados obtenidos a lo largo del desarrollo de la implementación del problema VRP dentro del PTIC en diferentes instancias. Para ofrecer una visión estructurada y comprensible, los resultados se han organizado en diferentes subsecciones. En primer lugar, en la Sección 6.1 se explica la metodología usada para la experimentación numérica. En la Sección 6.2 se comienza con el análisis de resultados para el problema del PTIC con foco en los métodos heurísticos. En la Sección 6.3 se culmina el análisis de los resultados del PTIC utilizando los paquetes de *software* Gurobi y SCIP. En la Sección 6.4 se continúa con la experimentación de los métodos heurísticos para el resto de instancias generadas utilizando datos extraídos de la literatura. Por último, en la Sección 6.5 se analiza el mejor resultado obtenido de las instancias del PTIC, creando la representación gráfica del ruteo.

6.1 Diseño de la experimentación

Una vez generados los datos de las instancias como fue explicado en la Sección 4.3 se definió la forma en que se llevaría a cabo la experimentación de los métodos implementados. Analizando el funcionamiento de las heurísticas seleccionadas (*FP, OSEA y LB)* se notó que las mismas cuentan con componentes estocásticos dentro de su funcionamiento, lo cual introduce variabilidad en los resultados obtenidos. Debido a ello se consideró necesaria la ejecución de cada instancia o problema al menos 5 veces para así exponer resultados que escapen de casos particulares y muestren una desviación estándar razonable. Este número de corridas por instancia está fundado en la utilización del máximo tiempo disponible para las ejecuciones de las mismas.

Los métodos utilizados son *FP*, *OSEA* y *LB*. Tanto *OSEA* como *LB* son heurísticas de mejora, por lo tanto, para la comparación entre las heurísticas *OSEA* y *LB* se requiere de una solución inicial factible. Para hallarla, se ejecuta previamente la heurística inicial *FP*, por lo que cada una de las corridas comenzará obligatoriamente por *FP*. Por otro lado, ambas heurísticas de mejora permiten la ejecución consecutiva, pudiendo entonces analizar dos nuevos métodos heurísticos que surgen de la concatenación de las mismas. Por lo tanto, se utiliza la solución obtenida del primer método de mejora para comenzar con el segundo método heurístico de mejora. Teniendo en cuenta que *LB* trabaja sobre el vecindario de la mejor solución obtenida hasta el momento y *OSEA* genera un conjunto de soluciones para fijar variables, se estima un mejor funcionamiento de la ejecución primero de *OSEA* y luego de *LB*. Esto debido a que mientras mejor sea la solución inicial, más cerca se podría estar de, al menos, un óptimo local de buena calidad. De igual forma, se evalúa la posibilidad de ejecutar primero *LB* y luego *OSEA* y se obtienen cuatro métodos heurísticos como se diagrama a continuación.

- *FP* + *OSEA*: Primero se aplica *FP* para obtener una solución inicial y luego se mejora la solución mediante *OSEA*.
- *FP* + *LB*: Primero se aplica *FP* y luego se mejora la solución mediante LB.
- *FP* + *OSEA* + *LB*: Primero se aplica *FP*, segundo se aplica *OSEA* para mejorar la solución y tercero se finaliza con *LB* como mejora adicional.

- *FP + LB + OSEA*: Primero se aplica FP, segundo se aplica *LB* como primera mejora y tercero se aplica *OSEA* como mejora complementaria.

Las instancias serán ejecutadas en el sistema operativo que fueron implementados los métodos, específicamente Ubuntu con la versión 24.04.2 LTS a través de la máquina virtual de VirtualBox. La computadora utilizada es una de uso personal con las siguientes características: CPU Intel Core i5 - 12400f - 2,50 GHz - 8 Gb de RAM. Los tiempos son expresados en segundos de CPU y todas las corridas se realizaron dedicando todo el poder computacional del PC. Debido a que se desea analizar el funcionamiento y rendimiento de las heurísticas, las mismas deberían ser comparadas contra la implementación de métodos exactos. Sin embargo, la alta accesibilidad a paquetes de *software* de optimización para la resolución de los problemas MILPs en la actualidad ha dejado obsoleta la utilización de métodos exactos únicamente. Es por ello que, teniendo en cuenta el número de herramientas incorporadas por defecto en los paquetes de *software,* se decide que las heurísticas sean comparadas contra uno no comercial, más específicamente GLPK. Este último, en su configuración predeterminada, utiliza planos de corte y *presolve en* su método de resolución. Dado que en un entorno práctico se requiere obtener soluciones en tiempos operativos y considerando que GLPK puede requerir largos períodos de ejecución, se estableció un límite de tiempo de dos horas para su ejecución. Este se consideró razonable desde el punto de vista computacional y aplicable en contextos reales.

Todos los métodos implementados se corren utilizando el mismo paquete de optimización. Esto quiere decir que, por ejemplo, para las iteraciones de *FP* donde se debe resolver problemas LP o al resolver el problema con las variables fijadas en *OSEA*, se utiliza GLPK. Específicamente se utiliza la versión GLPSOL--GLPK LP/MILP Solver 5.0. Esto permite una mayor consistencia en las comparaciones entre las heurísticas y el método.

Los parámetros de entrada de cada uno de los métodos se determinaron experimentalmente, comenzando con los valores propuestos en la literatura consultada de los diferentes métodos evaluados. Esto se logró en una etapa temprana de la implementación en donde se ejecutó el problema VRP del PTIC con los datos generados en la Sección 4.3.1. En esa etapa se ejecutaron los diferentes métodos y se evaluó cómo afectaba cada parámetro al resultado. Primero se comenzó con *FP* y luego se evaluaron *OSEA* y *LB* ya que requieren de una solución inicial. Para el resto de las instancias, los datos definidos obtenidos de bibliotecas se extrapolaron al tamaño del problema requerido. Se buscó mantener la mayoría de los parámetros para cada instancia para así evaluar la robustez de los métodos. De esta forma, se puede estudiar el comportamiento del método frente a la variabilidad de las instancias. Los valores de los parámetros usados en el problema VRP del PTIC con los datos generados en la Sección 4.3.1 se representan en la Tabla 5.

Tabla 5 - Valores de los parámetros

Método	Parámetro	Valor
Feasibility Pump	*MaxIterFP*	10000
	T	20
	KK	70
OSEA	$N_{máx}$	300

Local Branching	*MaxiterLB*	10
	k_T	400
	α	1,5
	β	0,5

Las únicas instancias para las que se modifican los parámetros son para el segundo conjunto de instancias (desde la 6 hasta la 9 en la Sección 4.3.2), donde el tamaño del problema varía. Los únicos parámetros que se modificaron en estos casos son: el tamaño de la vecindad k_T en *LB,* la cantidad de variables a cambiar en caso de encontrar un ciclo *T* y cada cuantas iteraciones *KK* se detectan repeticiones en *FP.* Se consideró que estos parámetros están relacionados con el tamaño del problema, lo que provoca que, en instancias pequeñas, valores altos de cualquiera de estos parámetros puedan requerir un tiempo de cómputo elevado. Por otro lado, en instancias grandes, los valores bajos pueden resultar insuficientes para explorar una región factible donde se pueda encontrar una mejor solución. Por lo tanto, para cada uno de los parámetros se consideró una relación lineal con el tamaño del problema. De esta forma se modificó estos parámetros como se muestra a continuación:

$$Parámetro = Tamaño\ actual * \frac{Parámetro\ base}{Tamaño\ base} \qquad (47)$$

Donde el parámetro será el nuevo valor de alguno de los tres parámetros (k_T, *T* y *KK)*. El *tamaño actual* hace referencia a la cantidad de empresas consideradas. El *parámetro base* y *tamaño base* son los valores asignados en la primera instancia (con los datos generados en la Sección 4.3.1). De esta forma, para los casos en donde el tamaño actual es el mismo que el tamaño de la primera, los parámetros permanecerán invariados.

En resumen, teniendo en cuenta las 9 instancias creadas en la generación de datos y los 4 métodos a evaluar, se ejecutarán un total de 200 corridas. Estas están compuestas por dos tipos, primero se realizan 9 corridas ejecutadas en GLPK, sin componente estocástica, a través del comando *glpsol* (una por cada instancia). Por otro lado, se realizan 5 corridas por cada instancia y método heurístico *(FP-LB), (FP-OSEA), (FP-OSEA-LB), (FP-LB-OSEA)*, exceptuando la primera instancia con datos del PTIC, a la cual se le realizó 10 corridas por método.

6.2 Resultados alcanzados para el problema de aplicación

En esta sección se pone énfasis en la primera instancia donde se ejecutan los datos del problema de ruteo del PTIC. Una vez ejecutadas las 10 corridas para cada método heurístico se desea comparar el valor funcional obtenido con el valor obtenido por el paquete GLPK en tiempos comparables. En la Figura 12 se pueden observar los valores y tiempos promedios de las ejecuciones realizadas, además de la desviación estándar para cada método. En el gráfico se representa mediante barras los resultados promedios de la función objetivo y mediante líneas los diferentes tiempos de cómputo. Los tiempos promedios se toman como la suma de tiempos de cada uno de los métodos y se comparan con el valor funcional de la primera

solución de GLPK y su tiempo de cómputo. De estos resultados se destaca el método heurístico resultante de la concatenación *FP+OSEA+LB* como el método de mejor calidad por balance tiempo-valor funcional. Para acudir a los datos de todos los resultados obtenidos, consultar el Apéndice IV: Resultados Tablas donde se pueden observar los tiempos y valores funcionales obtenidos por los métodos heurísticos, así como sus resultados intermedios.

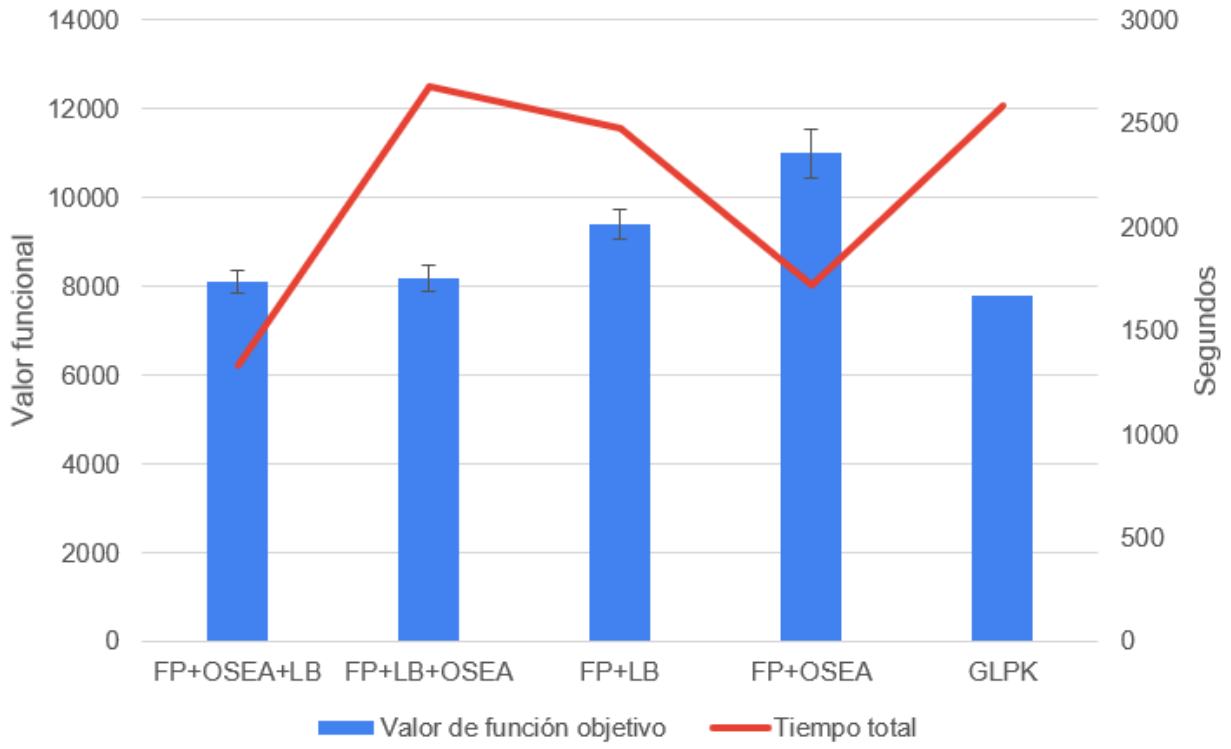

Figura 12 - Tiempos y valores funcionales promedio

El tiempo promedio de cómputo considerando todos los métodos heurísticos ejecutados es de 2051 segundos (34 minutos), pero GLPK no es capaz de en este tiempo encontrar una solución factible. Es a los 2584 segundos (43 minutos) de ejecución de GLPK que es posible encontrar la primera solución, con un valor funcional de 7801 unidades de distancia y un gap de dualidad del 84 %. Al ser un gap de dualidad tan alto no se puede asegurar la calidad de la solución, por lo que no se puede determinar la calidad de las soluciones alcanzadas para ninguno de los métodos. El valor obtenido mediante GLPK sólo es superado en tres corridas de *FP+OSEA+LB* y en una de *FP+LB.* En promedio, el método *FP+OSEA+LB* supera en un 4% el valor funcional de GLPK con 8098 unidades de distancia (con objetivo de minimización) pero lo logra en la mitad de tiempo, con un promedio de 1333 segundos (22 minutos). Para *FP+LB+OSEA* el valor funcional supera en un 5% el de GLPK en un tiempo de cómputo 4% mayor. Para *FP+LB* se observa un valor funcional 20% por encima en el 96% del tiempo de cómputo de GLPK y para *FP+OSEA* un valor funcional 40% por encima en el 66% del tiempo de cómputo. En base a estos resultados se destaca el método *FP+OSEA+LB* como superior al paquete GLPK para este problema por la calidad del resultado en un balance valor funcional y tiempo de cómputo.

De los métodos heurísticos se determina que *FP+OSEA+LB* es quien alcanza mejores resultados tanto en tiempos como en valores funcionales. Además, para este caso se nota una

sinergia entre los métodos de mejora comenzando con *OSEA* y continuando con *LB,* obteniendo resultados mejores, al menos en tiempo de cómputo, que al aplicar las heurísticas por separado. Se observa una notable diferencia en los tiempos de cómputo de la ejecución comparado con *FP+LB*. Por ello se concluye que la aplicación del método *OSEA* reduce notoriamente los tiempos de cómputo necesarios en la aplicación de *LB*. Se puede entonces concluir que, para este problema, el método *LB* es el principal responsable de la mejora en el valor funcional de la solución mientras que el método *OSEA* acelera el proceso.

Buscando mostrar el impacto que tiene el factor estocástico de los métodos se expone que el peor resultado obtenido de FP+OSEA+LB es de 9468 unidades de distancia, mientras que el mejor es 6870,9 unidades de distancia. En el tiempo de cómputo es aún más notoria la diferencia, pues el menor tiempo de cómputo obtenido fue 408 segundos mientras que el mayor es de 2996 segundos, aproximadamente 7 veces más. Con estos resultados se comprueba la necesidad de realizar múltiples corridas para la correcta evaluación de resultados.

En la Figura 13 se pueden observar los mejores resultados obtenidos de las corridas en función del valor funcional de las soluciones encontradas en cada corrida de la instancia. De este análisis se observa nuevamente el mejor resultado en el método *FP+OSEA+LB*. Además, se deja en evidencia que con la implementación de las heurísticas, al menos en algunos casos, se puede obtener mejores resultados que con paquetes de *software* como GLPK en tiempos de cómputo claramente menores. Por ejemplo, la mejor corrida de *FP+OSEA+LB* y *FP+LB+OSEA* logran mejores soluciones en tiempos menores. Por otro lado, *FP+LB* logra mejor solución en tiempos mayores. Por último, *FP+OSEA* logra un peor valor en la solución, en tiempos menores al de *FP+LB*.

Por otro lado, luego de dos horas de corrida de GLPK el resultado de GLPK es de 7529 unidades de distancia, aún con un gap de dualidad alto del 83 %. Este valor funcional encontrado por GLPK en 7200 segundos es mejorado por la implementación de *FP+OSEA+LB* en 2 de las 10 corridas con tiempos menores a 1800 segundos y en una por el método *FP+LB* con tiempo de cómputo de 3781 segundos (63 minutos).

Es de notar que este último valor mostrado de tiempo total insumido en la concatenación *FP+LB* es la única corrida de valores extraordinarios (tiempo de cómputo) como se puede consultar en el Apéndice IV: Resultados Tablas. Este tiempo de resolución se debe a que en esta corrida se agrandó el tamaño del vecindario en más de una ocasión, generando un problema más grande y resultando en mayores tiempos de resolución. A su vez, esto también causó que el resultado funcional obtenido fuera mejor debido a que con un mayor espacio de búsqueda fue capaz de encontrar mejores soluciones.

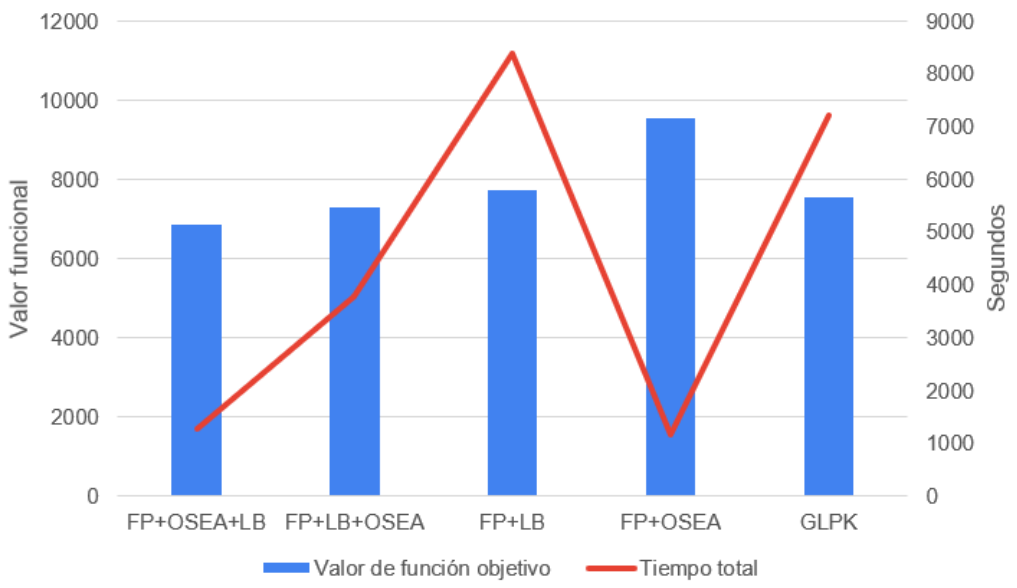

Figura 13 - Tiempos y valores funcionales de las mejores corridas

Al comparar el rendimiento de las heurísticas de mejora en solitario *FP+LB* y *FP+OSEA* se busca determinar cual logra mejores resultados. Se observa cómo en promedio los resultados alcanzados por LB son mejores que los alcanzados por OSEA en un 15%. Por otro lado, evaluando los tiempos de cada uno de los métodos se observa que, en promedio, *LB* insume 3,6 veces el tiempo de cómputo insumido por *OSEA*. De esta forma, se logra determinar que como métodos aislados, LB logra un mejor resultado en la función objetivo pero con peores resultados en tiempo de cómputo en comparación con OSEA.

Realizando nuevamente una comparativa de los métodos *FP+OSEA+LB* y *FP+LB+OSEA* se observa un mejor comportamiento por parte de aplicar primero *OSEA* y luego *LB* tanto para los valores funcionales como el tiempo de cómputo promedio. Específicamente, se observa en el Apéndice IV: Resultados Tablas cómo, al aplicar primero *LB* y luego *OSEA*, este último no logra una mejora en el resultado obtenido con *LB* aunque sí se insuma tiempo de cómputo. Se observó que al aplicar *LB* primero se genera una convergencia temprana hacia óptimos locales, lo cual limita la capacidad exploratoria de *OSEA* en etapas posteriores. En resumen, se observa que *LB* culmina en un óptimo local, este, al ser utilizado como solución inicial del método *OSEA*, genera que aún modificando los costos del problema no se pueda mejorar la solución. Por otro lado, el método *LB* cuenta con mecanismos para escapar de óptimos locales al poder aumentar el valor del tamaño de la vecindad k_T , mecanismos que *OSEA* no posee haciendo que no pueda escapar de la solución inicial en este caso. Esto sugiere que el orden de aplicación de las heurísticas influye directamente en la calidad final de las soluciones y que la sinergia obtenida en la configuración *OSEA+LB* no se replica al invertir el proceso, sino que tan sólo se insumen más recursos sin mejorar la solución. Entonces, se deduce que la aplicación de *FP+LB+OSEA* es lo mismo que aplicar *FP+LB*.

Al buscar evaluar el mejor método obtenido, en la Figura 14 se pueden observar los valores funcionales obtenidos luego de aplicar cada método heurístico de la concatenación *FP+OSEA+LB* y el valor funcional obtenido de la ejecución de 2 horas del paquete GLPK. Gráficamente se puede observar un correcto y balanceado funcionamiento de los métodos de

mejora, observando una forma escalonada en donde se mejora cada solución luego de cada heurística. En promedio, con la utilización del método *OSEA* se observa una mejora en la función objetivo de aproximadamente 3000 unidades de distancia. Luego, mediante el método *LB* se observa una mejora de aproximadamente 3500 unidades de distancia. Debido a esto y los bajos tiempos de cómputo, se considera un caso de éxito la concatenación de los dos métodos. Se observa que la instancia número cuatro es la que logra mejores resultados, sin ser la que tiene una solución inicial (luego de *FP*) más baja. Agregado a esto, se observa que la solución luego de OSEA es la segunda más alta de todas las corridas y es el método *LB* el gran responsable del buen resultado .

Por otro lado, se observa como la partida de los métodos de mejora inicia con un valor inicial (luego de *FP*) elevado en comparación con la primera solución encontrada en GLPK. Por lo tanto se deseó analizar los tiempos de cómputo de cada una de los métodos heurísticos y por lo que se presentan las Figura 15 y Figura 16. Además, en la Tabla 6 se pueden observar los valores funcionales promedio luego de la aplicación de cada método.

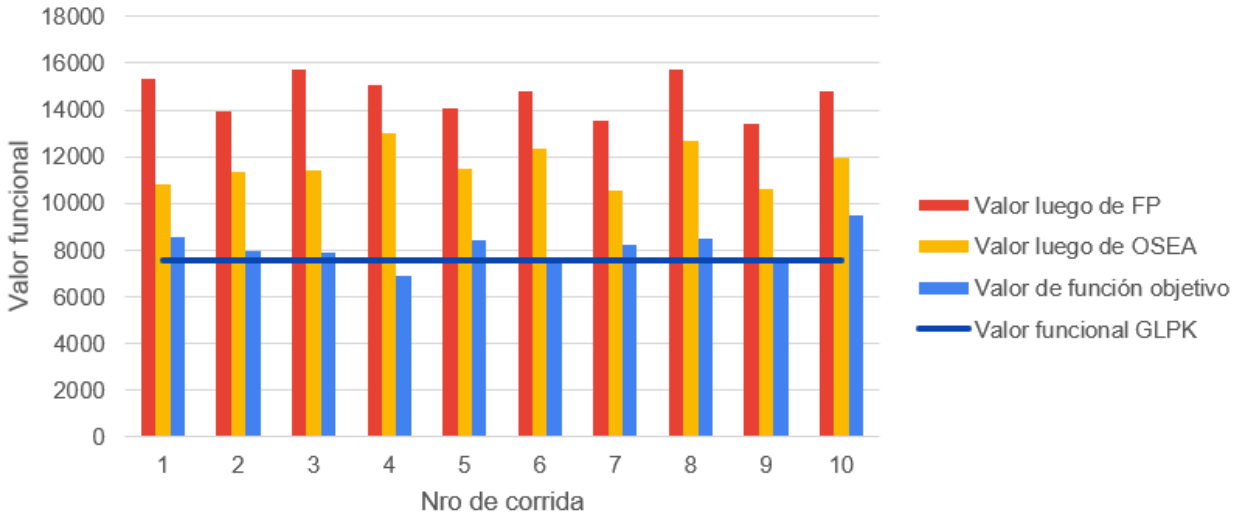

Figura 14 - Valor funcional de las corridas

Tabla 6 - Valores promedio por método de la concatenación de las heurísticas

Valor luego de FP	Valor luego de OSEA	Valor de función objetivo
14639,47	11610,15	8098,405

En la Figura 15 y Figura 16 se muestran los tiempos de cómputo insumidos en la concatenación *FP+OSEA+LB* para cada corrida y la distribución promedio respectivamente. Del análisis de estas gráficas y la de la Figura 14 se observa que el valor funcional de la solución encontrada por el método de *FP* no sólo es alto, sino que es el método que consume la mayor cantidad del tiempo de cómputo. Por otro lado, el tiempo promedio insumido por *FP* de 900 segundos (15 minutos) sigue estando muy por debajo de los 43 minutos que necesita GLPK, por lo que no es posible afirmar que sea una mala forma de obtención de solución inicial. Por otro lado, haciendo un análisis de tiempo y resultados se observa como mayoritariamente (el 80

% de los casos), los tiempos que insume la heurística de OSEA son mayores a los de LB en la concatenación pero estos bajos tiempos de cómputo de *LB* son posibles gracias a la aplicación de *OSEA*. Además se observó cómo, en promedio, la heurística LB logra mejorar la solución en mayor medida en comparación con OSEA.

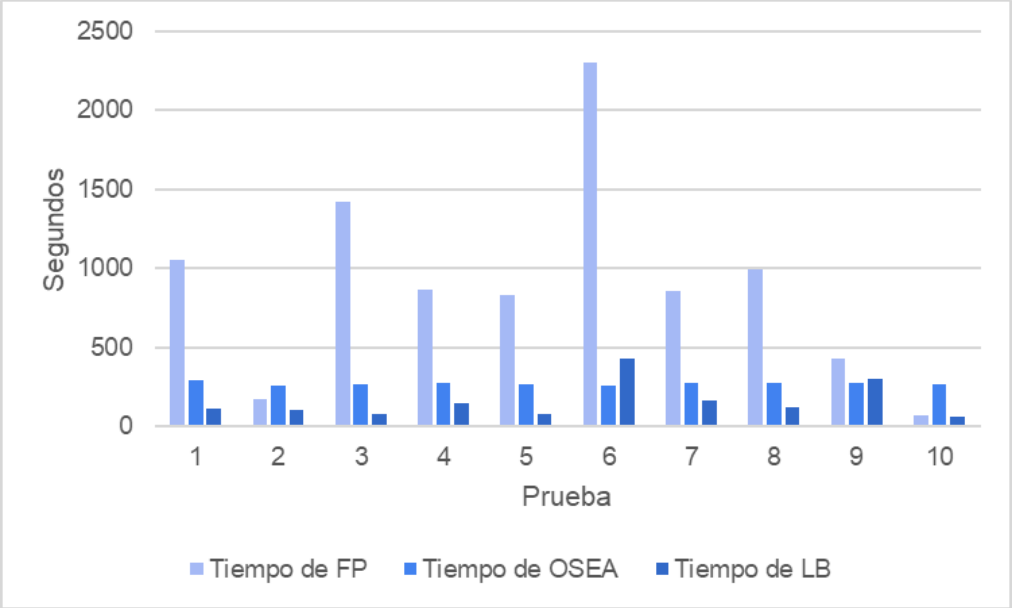

Figura 15 - Tiempos de cómputo por método en las corridas

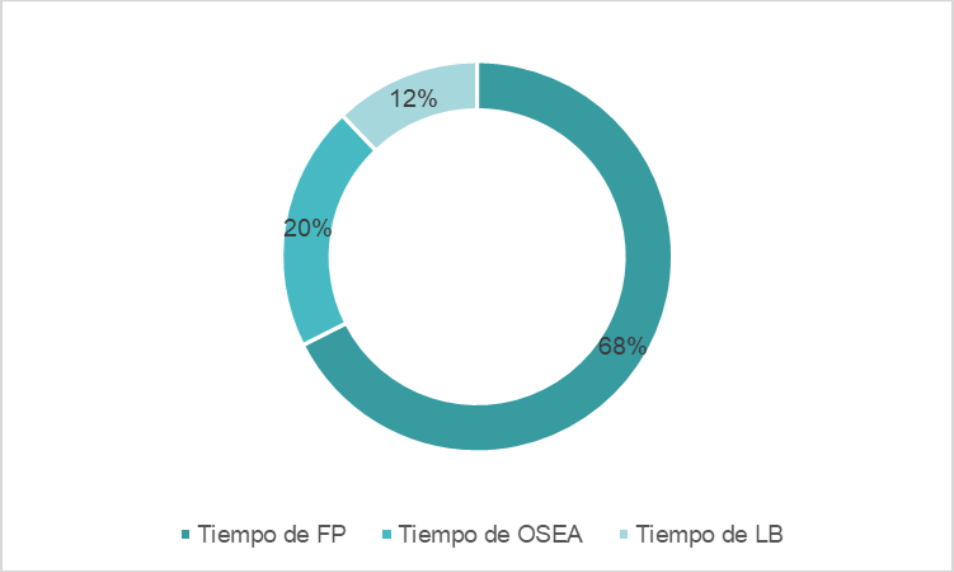

Figura 16 - Tiempo de cómputo por método

Se concluye que el mejor método heurístico es *FP+OSEA+LB* el cual logra resultados que se asemejan al del paquete GLPK en la mitad del tiempo. Además, se determina como una debilidad del método el valor funcional de la heurística de solución inicial *FP*. Esta es la que

más tiempo de cómputo conlleva y es una solución inicial el doble de grande en valor funcional que la primera obtenida mediante GLPK, aunque es lograda en un tercio del tiempo de cómputo.

6.3 Resultados alcanzados en Gurobi y SCIP

Para mostrar la mayor eficiencia de los paquetes comerciales se presentan los resultados de la solución del problema del PTIC conseguida a través de la ejecución del código presentado en la Figura 17. Este fue ejecutado en el paquete de *software* comercial Gurobi Optimizer versión 12.0.3 para Windows 64 bits. Podemos observar cómo en tan sólo 45 segundos una solución factible es encontrada y a los 60 segundos se encuentra una solución ampliamente mejor a las encontradas en la sección anterior, con un valor funcional de 5433 unidades de distancia. Esto, si bien es experimentado sólo para un problema, dada la magnitud de diferencia tanto en el valor funcional como en tiempos de cómputo se aporta como un ejemplo para asegurar que existe una sustancial diferencia en cuanto a la calidad de las soluciones entre paquetes comerciales y no comerciales.

```
   Nodes    |    Current Node    |     Objective Bounds      |     Work
Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1255.49667    0  106          - 1255.49667      -     -    1s
     0     0 1255.49667    0  105          - 1255.49667      -     -    2s
     0     0 1255.49667    0  115          - 1255.49667      -     -    2s
     0     0 1255.49667    0  109          - 1255.49667      -     -    4s
     0     0 1255.49667    0  118          - 1255.49667      -     -    4s
     0     0 1255.49667    0  121          - 1255.49667      -     -    5s
     0     0 1255.49667    0  126          - 1255.49667      -     -    5s
     0     0 1255.49667    0  120          - 1255.49667      -     -    7s
     0     0 1255.49667    0  107          - 1255.49667      -     -    7s
     0     2 1255.79667    0  107          - 1255.79667      -     -   10s
   216   233 1262.00044   57  109          - 1255.79667      -  46.4   15s
   606   648 1296.49277  210   99          - 1255.79667      -  39.5   20s
  1177  1235 1300.75626  428  111          - 1255.79667      -  27.8   25s
  1817  1905 1308.77272  715  128          - 1255.79667      -  22.2   30s
  2403  2511 1320.95017  944  136          - 1255.79667      -  20.1   35s
  2960  3086 1635.62812 1151   94          - 1255.79667      -  19.6   41s
  3425  3511 1666.95420 1355   80          - 1255.79667      -  19.2   45s
H 3492  3511                    12891.440000 1255.79667  90.3%  19.1   45s
H 3548  3513                    12440.480000 1255.79667  89.9%  19.2   46s
H 3549  3513                    6707.1000000 1255.79667  81.3%  19.2   46s
H 3550  3515                    5907.7400000 1255.79667  78.7%  19.2   48s
H 3596  3568                    5565.9300000 1255.79667  77.4%  19.7   50s
H 3618  3391                    5475.6500000 1255.79667  77.1%  20.0   55s
H 3619  3221                    5433.6300000 1255.79667  76.9%  20.0   56s
  3628  3227 1323.74962  983  225 5433.63000 1258.26429  76.8%  19.9   60s
```

Figura 17 - Resultados alcanzados por Gurobi

De todas formas, con el objetivo de una comparación justa entre paquetes de *software* de optimización no comerciales, se muestran los resultados de la ejecución en el paquete de *software* SCIP. Este último es considerado el de mejor rendimiento dentro de los paquetes no comerciales. Para obtener los resultados de la Figura 18 se ejecutó el código presentado en la Figura 10. Se puede observar en estos resultados que efectivamente se obtienen mejores

resultados comparados con los de GLPK pero que sigue habiendo una relevante diferencia con los presentados de la ejecución de paquetes comerciales. La primera solución se obtiene a los 350 segundos con un valor de 8492,3. En comparación, GLPK lo logra a los 2584 segundos pero con un valor de 7800,7. Sin embargo, a los 544 segundos el paquete SCIP logra mejorar la solución de GLPK por lo que en SCIP se obtienen mejores resultados iniciales.

a)

```
 time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons |rows |cuts |sepa|confs|strbr|  dualbound   | primalbou
nd   |  gap   | compl.
  4.6s|     1 |     0 |  9934 |     - |   339M |   0 |  20k|  21k|  21k|   0 |  0 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
  9.2s|     1 |     0 | 26304 |     - |   346M |   0 |  20k|  21k|  21k| 162 |  1 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 10.0s|     1 |     0 | 27219 |     - |   349M |   0 |  20k|  21k|  21k| 275 |  2 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 10.5s|     1 |     0 | 27697 |     - |   351M |   0 |  20k|  21k|  21k| 347 |  3 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 10.9s|     1 |     0 | 27903 |     - |   353M |   0 |  20k|  21k|  21k| 372 |  4 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 11.2s|     1 |     0 | 28042 |     - |   355M |   0 |  20k|  21k|  21k| 380 |  5 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 11.4s|     1 |     0 | 28063 |     - |   358M |   0 |  20k|  21k|  21k| 385 |  6 |   2 |   0 | 1.269985e+03 |      --
    |    Inf | unknown
 36.6s|     1 |     2 | 46428 |     - |   387M |   0 |  20k|  21k|  21k| 385 |  7 |  32 |  20 | 1.269985e+03 |      --
    |    Inf | unknown
  177s|   100 |    91 |139231 |1124.2 |   552M |  15 |  20k|  21k|  21k|2052 |  1 | 104 | 351 | 1.269985e+03 |      --
    |    Inf | unknown
  267s|   200 |   191 |248242 |1107.1 |   689M |  21 |  20k|  21k|  21k|4559 |  1 | 192 | 440 | 1.269985e+03 |      --
    |    Inf | unknown
d 350s|   294 |   285 |349242 |1096.6 |distribu|  21 |  20k|  21k|  21k|   0 |  1 | 318 | 543 | 1.269985e+03 | 8.492310e
+03 | 568.69%| unknown
  361s|   300 |   291 |360570 |1112.5 |   798M |  21 |  20k|  21k|  21k|7124 |  9 | 318 | 563 | 1.269985e+03 | 8.492310e
+03 | 568.69%| unknown
```

b)

```
 time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons |rows |cuts |sepa|confs|strbr|  dualbound   | primalbou
nd   |  gap   | compl.
 6641s|  3300 |  3205 |  8289k|1133.5 |  1843M |  41 |  20k|  23k|  21k|  37k|  1 |6322 |6532 | 1.451296e+03 | 7.375090e
+03 | 408.17%| unknown
 6730s|  3400 |  3303 |  8408k|1134.3 |  1844M |  41 |  20k|  23k|  21k|  38k|  1 |6435 |6622 | 1.454492e+03 | 7.375090e
+03 | 407.06%| unknown
 6800s|  3500 |  3403 |  8512k|1133.0 |  1847M |  41 |  20k|  23k|  21k|  39k|  1 |6586 |6671 | 1.454492e+03 | 7.375090e
+03 | 407.06%| unknown
 6862s|  3600 |  3499 |  8585k|1127.7 |  1849M |  41 |  20k|  23k|  21k|  40k|  5 |6661 |6763 | 1.454492e+03 | 7.375090e
+03 | 407.06%| unknown
 6896s|  3700 |  3591 |  8641k|1120.4 |  1852M |  41 |  20k|  23k|  21k|  40k|  1 |6744 |6797 | 1.454492e+03 | 7.375090e
+03 | 407.06%| unknown
 6979s|  3800 |  3691 |  8754k|1120.5 |  1870M |  41 |  20k|  23k|  21k|  41k|  1 |6849 |6873 | 1.454492e+03 | 7.375090e
+03 | 407.06%| unknown
 7034s|  3900 |  3787 |  8828k|1115.6 |  1871M |  41 |  20k|  23k|  21k|  41k|  1 |6962 |6929 | 1.454994e+03 | 7.375090e
+03 | 406.88%| unknown
 7096s|  4000 |  3885 |  8923k|1113.5 |  1875M |  41 |  20k|  23k|  21k|  42k|  1 |7058 |6973 | 1.455866e+03 | 7.375090e
+03 | 406.58%| unknown
 7163s|  4100 |  3985 |  9004k|1109.7 |  1880M |  41 |  20k|  23k|  21k|  43k|  1 |7241 |7081 | 1.455866e+03 | 7.375090e
+03 | 406.58%| unknown

SCIP Status        : solving was interrupted [time limit reached]
Solving Time (sec) : 7200.00
Solving Nodes      : 4135 (total of 8104 nodes in 2 runs)
Primal Bound       : +7.37509000000000e+03 (8 solutions)
Dual Bound         : +1.46203645557587e+03
```

Figura 18 - a) Primeros resultados alcanzados por SCIP. b) Resultados finales alcanzados por SCIP

Luego de dos horas de ejecución, los resultados alcanzados por SCIP son mejores que los alcanzados mediante GLPK. SCIP alcanza un valor funcional de 7375 y GLPK 7529. Se observa que dentro de los paquetes no comerciales, SCIP muestra mejores rendimientos que

GLPK, principalmente para los resultados iniciales ya que a las dos horas de ejecución los resultados son del mismo orden. Por otro lado, se muestra que Gurobi, como paquete comercial, logra mejores resultados como fue visto en la Sección 2.2. Por lo tanto, se concluye que el resto de las soluciones alcanzadas para el problema del PTIC son de baja calidad al compararse con el paquete comercial Gurobi.

6.4 Resultados con el resto de las instancias

En esta sección se presentan los resultados obtenidos para las instancias generadas a partir de bibliotecas de datos obtenidas mediante Chaire de recherche en distributique – HEC Montréal (s.f.). Desde la Figura 19 hasta la Figura 24 se muestran los valores promedios para los métodos implementados en este documento, comparados con la solución inicial encontrada por GLPK. Se decide presentar estos datos debido a que en la gran mayoría de las instancias, la solución inicial de GLPK es una solución de mejor calidad que las encontradas por los métodos seleccionados y en tiempos menores de cómputo. Por ello, se consideran malos los resultados de los métodos implementados para estos problemas, poniendo foco en la creación de hipótesis de por qué se dan estos resultados.

El análisis se divide en dos tipos de instancias. En primer lugar, las primeras cuatro instancias son de tamaño idéntico al del problema original (31 empresas en un mes) en donde se varían los datos introducidos. Además, se considera un factor α que varía para cada instancia, con $2 > \alpha > 0$, que multiplica la capacidad del vehículo. Se busca estudiar cómo afecta al resultado la diferencia en la dificultad del problema al variar α. Si al aumentar la capacidad del contenedor la región factible aumenta de tamaño, entonces la búsqueda de una solución factible se vuelve más sencilla de encontrar. Por otro lado, se evalúan instancias a las que se varía el tamaño del problema desde 10, 20, 30 y 40 la cantidad de empresas dentro de un mes. Con esto se busca evaluar la eficiencia de los métodos dependiendo del tamaño de las instancias.

El método GLPK presenta un buen funcionamiento que supera los resultados tanto en tiempo como en valor funcional en la mayoría de los métodos heurísticos. Este cambio, en comparación con la primera instancia donde los resultados alcanzados son más equiparables entre los diferentes métodos, se puede explicar básicamente porque los métodos heurísticos están orientados y destacan cuando se enfrentan a problemas difíciles de resolver. Los problemas generados con esta biblioteca no son considerados difíciles para el método de resolución de GLPK. Esta consideración se fundamenta en que el paquete de *software* de GLPK nunca insumió más de 600 segundos (10 minutos) de cómputo para hallar una solución factible, caso completamente contrario al visto con los datos generados del PTIC. Además, la forma de definición de la capacidad es muy similar en todas estas instancias por lo se considera que todas las instancias comparten una similitud en la dificultad. También, se aclara que todas los resultados del método GLPK son la primera solución obtenida a través del *branching* pero que en todos los casos, tras 2 horas de ejecución del método, no se demuestra la optimalidad.

Para la evaluación se divide esta sección en dos sub secciones. La Sección 6.4.1 evalúa los resultados alcanzados por las instancias donde se varía el parámetro α. En la Sección 6.4.2 se evalúan los resultados alcanzados por las instancias donde se varían los tamaños.

6.4.1. Instancias tipo 1: Variando parámetro α

En la Figura 19, presentada para la instancia 2, se observa como la solución obtenida a través de GLPK es de calidad superior en tiempo y valor funcional que el resto de métodos heurísticos. De los métodos heurísticos, el que logra mejores tiempos es *FP+LB* y el mejor valor funcional es *FP+OSEA+LB.* Se observa cómo para este caso los resultados de *FP+LB* son mejores que *FP+LB+OSEA* tanto para tiempos como para valores funcionales. Este fenómeno puede explicarse por la componente estocástica presente en los métodos utilizados. En particular, la ejecución del enfoque *FP+LB* presentó mejores resultados en el primer método, posiblemente debido a una combinación favorable de soluciones de los métodos *FP* y *LB*. En contraste, en el segundo método se vuelve a evidenciar la ineficiencia del uso de OSEA posterior a la aplicación de LB, lo que sugiere que la calidad de las soluciones generadas por *LB* anula el margen de mejora de *OSEA.*

La obtención de soluciones iniciales se logra en tiempos menores en comparación con la primera instancia. Esto se adjudica a una menor dificultad del problema sumado a que el factor α utilizado para este problema es *1,2,* lo que haría que dentro aumente el tamaño de la región factible facilitando la obtención de la solución inicial. En la Figura 19 se observa un gráfico de barras en donde se observa el promedio del valor funcional y los tiempos de las 5 corridas. Se observa que el método *FP+OSEA* insume un tiempo de cómputo mayor al de *FP+LB*, sucediendo lo contrario a lo observado en la primera instancia. Analizando los resultados se observa que el tiempo de cómputo del método *OSEA* es del mismo orden al insumido en la primera instancia. Por otro lado, el tiempo de cómputo insumido en *FP* y *LB* se ven disminuidos. Esto puede ser explicado por la menor complejidad del problema para un mismo tamaño. Debido a la menor complejidad, se plantea que existe una mayor cantidad de soluciones factibles, lo que impacta en una menor dificultad de los métodos *FP* y *LB* que se basan en la evaluación de soluciones en una región vecina. Por otro lado, la heurística de *OSEA* resuelve problemas LP en su núcleo y culmina con la resolución de un problema MILP reducido, el cual tiene muchas variables fijadas. Dado que debe resolver una gran cantidad de problemas LP, se determina que variar el tamaño del problema es lo que tendría un mayor impacto en los tiempos de cómputo.

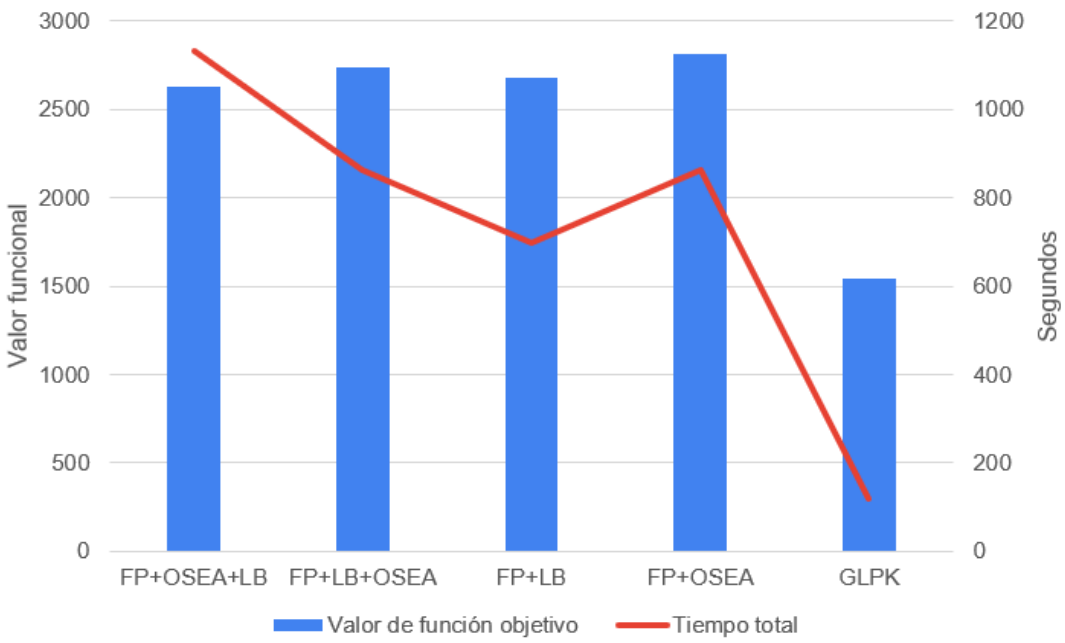

Figura 19 - Tiempos y valores funcionales promedio instancia 2

En la instancia 3 los tiempos de cómputo de todos los métodos aumentaron como se puede observar en la Figura 20. Esto se debe a que es la única instancia donde α es menor a *1*, específicamente $\alpha = 0,8$, por lo que se considera un problema de mayor dificultad (menor tamaño de la región factible). Se puede observar en el Apéndice IV: Resultados Tablas como para la mayoría de los métodos heurísticos el tiempo de cómputo de *FP* aumentó en comparación con la anterior instancia, y lo mismo ocurrió con el método *LB*. El mejor valor funcional alcanzado se logra a través del método *FP+LB+OSEA*. Los menores tiempos de cómputo son logrados por *FP+OSEA*, esto es explicado por una invarianza en los tiempos de cómputo de *OSEA* con el tamaño del problema pero la mayor dificultad afectando los procesos *FP* y *LB*.

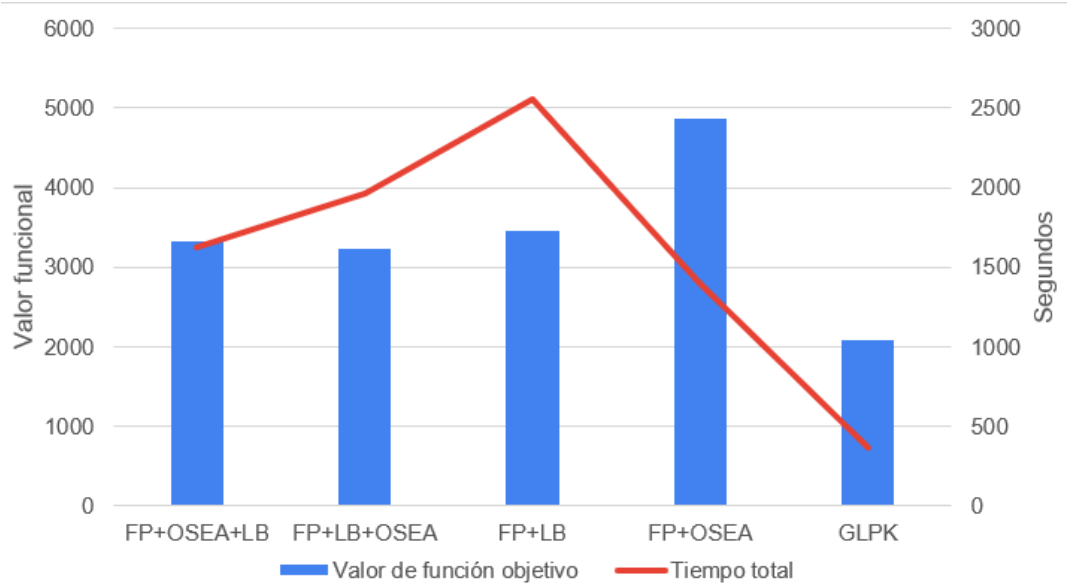

Figura 20 - Tiempos y valores funcionales promedio instancia 3

En la Figura 21 se observa el mismo tipo de gráfico para la instancia número 4. Se obtiene un análisis muy similar a la instancia 2 ya que se trabaja con un $\alpha > 1$ y el resultado de GLPK es el que brinda los mejores resultados y tiempos de resolución. En el caso de las heurísticas, los tiempos de cómputo menores son los de *FP+LB*, adjudicado a una baja dificultad del problema. El factor de α utilizado es *1,7* y se obtiene el resultado esperado pues los tiempos promedios de *FP* son menores, incluso que la instancia 2 donde α es *1,2*, implicando que el espacio factible es mayor y la obtención de un resultado inicial es aún más sencillo. Para el método *LB*, donde también se esperaba una reducción de los tiempos de cómputo, se observan casos favorables y desfavorables, pero los tiempos son del mismo orden que para la instancia 2. El mejor resultado del valor funcional se obtiene mediante la heurística de *FP+LB+OSEA* aunque lo logre en tiempos mayores que *FP+LB,* lo que implica que aumentó más veces el valor del tamaño del vecindario. Además, nuevamente no se observan mejoras en los resultados por la aplicación de *OSEA* luego de *LB* y es un caso mínimamente favorable (8% en el valor funcional) de la componente estocástica que insume un tiempo de cómputo innecesario.

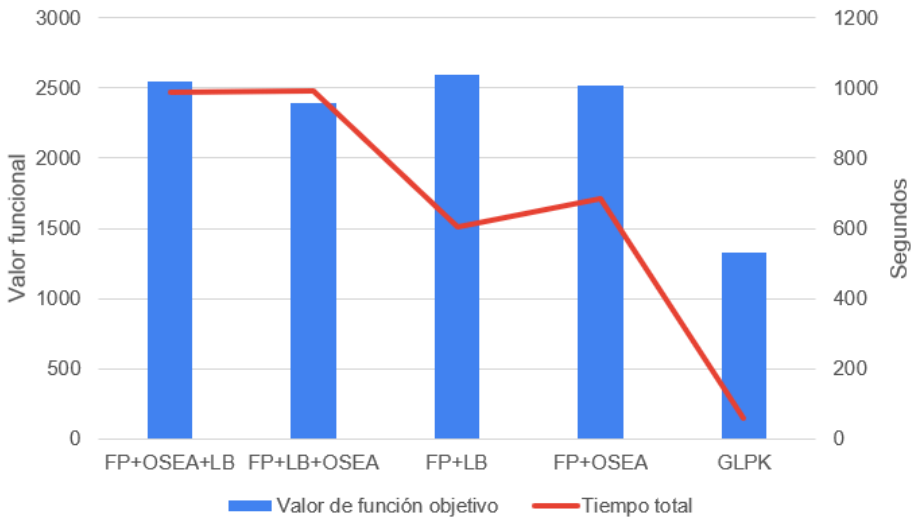

Figura 21 - Tiempos y valores funcionales promedio instancia 4

La instancia 5 es la última en donde se mantiene el tamaño del problema original. En esta instancia el factor α es igual a *1* al igual que en la primera instancia. Observando los métodos heurísticos se puede hacer una analogía a la primera instancia. Respecto a los tiempos de cómputo, al utilizar un $\alpha \leq 1$ se observa un beneficio de la utilización del método *OSEA* pero es *LB* el método de mejora más relevante en términos de valor funcional como se observa en la Figura 22. Por otro lado, en cuanto a la resolución a través del paquete GLPK se observan nuevamente soluciones ampliamente mejoradas a la conseguidas a través de los métodos heurísticos implementados. Estas se siguen adjudicando a una baja dificultad para el paquete para la resolución de los problemas generados a través de la biblioteca de datos.

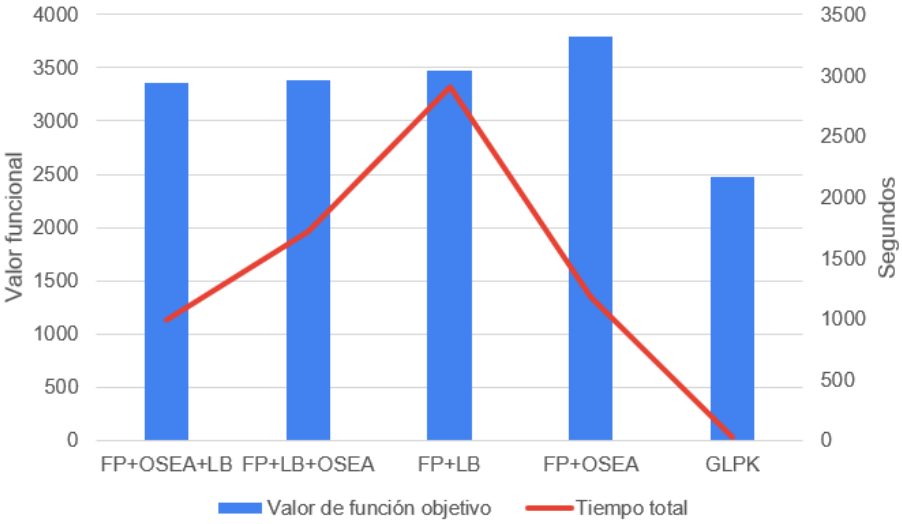

Figura 22 - Tiempos y valores funcionales promedio instancia 5

De estas instancias se concluye una correcta identificación del relacionamiento entre los parámetros del problema y el funcionamiento de los métodos heurísticos. Se observa una

relación inversamente proporcional entre el tamaño de la región factible y la velocidad de resolución del método *FP* y *LB,* pues al aumentar el factor α los tiempos de culminación de los métodos disminuyen. Por otro lado, se plantea una relación entre el tiempo de cómputo de la aplicación de *OSEA* y el tamaño del problema (cantidad de empresas), esta misma podrá ser comprobada en la siguiente sección. Por otro lado, se concluye que el factor estocástico favoreció en términos de valor funcional a la concatenación *FP+LB+OSEA*, y en ciertas ocasiones también en términos de tiempo de cómputo, pero si se observa el Apéndice IV: Resultados Tablas se podrá observar que en todos los casos la implementación de *OSEA* luego de *LB* no mejora los resultados.

6.4.2. Instancias tipo 2: Variando el tamaño

A partir de esta instancia se siguen obteniendo los datos de bibliotecas pero se modifica el tamaño del problema manteniendo el factor α en *1*. La instancia 6 cuenta con 10 empresas, siendo el problema más pequeño evaluado. Los tiempos en los que se resuelve son muy pequeños, no llegando a los 20 segundos. De todas formas, la solución inicial mediante el método GLPK es aún más rápida y mejor. Los resultados alcanzados tanto en tiempos y valores funcionales son muy similares para todos los métodos como se observa en la Figura 23. De esta forma se observa que los métodos *LB* y *FP* también están relacionados con el tamaño del problema. Esto último tiene sentido ya que para *LB* el tamaño del vecindario aumenta. Respecto a *FP,* al tener que iterativamente resolver problemas LP de mayor tamaño los tiempos de cómputo son mayores. Entre los métodos heurísticos, se observa como *FP+LB* es quien obtiene los mejores resultados en promedio. Respecto al método *FP+OSEA+LB,* se observa que en solo 1 de las 5 corridas se da una mejora a través de *LB* lo que indicaría que tras la solución de OSEA se está alcanzando mínimos locales y que el método de escape no es efectivo. Además, se considera que modificar algunos de los parámetros de entrada de los métodos heurísticos pudo tener relación con la mejora de los resultados alcanzados en los métodos heurísticos en comparación con otras instancias, por ejemplo, por la modificación del tamaño del vecindario en el método *LB*.

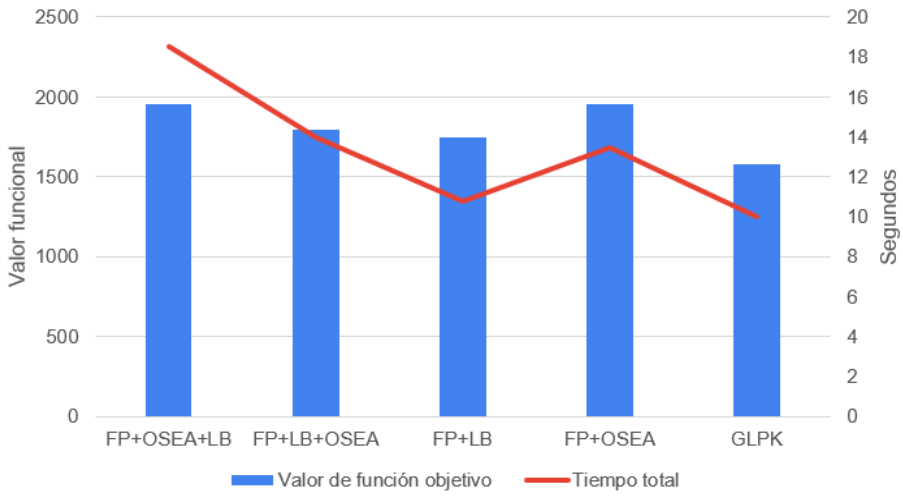

Figura 23 - Tiempos y valores funcionales promedio instancia 6

En la instancia 7, cuya cantidad de empresas son 20, el mejor resultado alcanzado en valores funcionales es obtenido mediante un método heurístico, más específicamente el método *FP+OSEA+LB.* Por otro lado, los mejores tiempos se obtienen a través del método GLPK, adjudicado a la facilidad del mismo para resolver los problemas generados con la biblioteca. La mejora en los resultados puede deberse a la modificación en los parámetros tanto para *FP* como para *LB.* Esto se refuerza al observar como los resultados alcanzados para *OSEA*, donde no se modifica ningún parámetro de entrada por el tamaño del problema, presenta resultados muy similares en tiempos de cómputo, al de instancias anteriores. Estos resultados se observan en la Figura 24 y el Apéndice IV: Resultados Tablas.

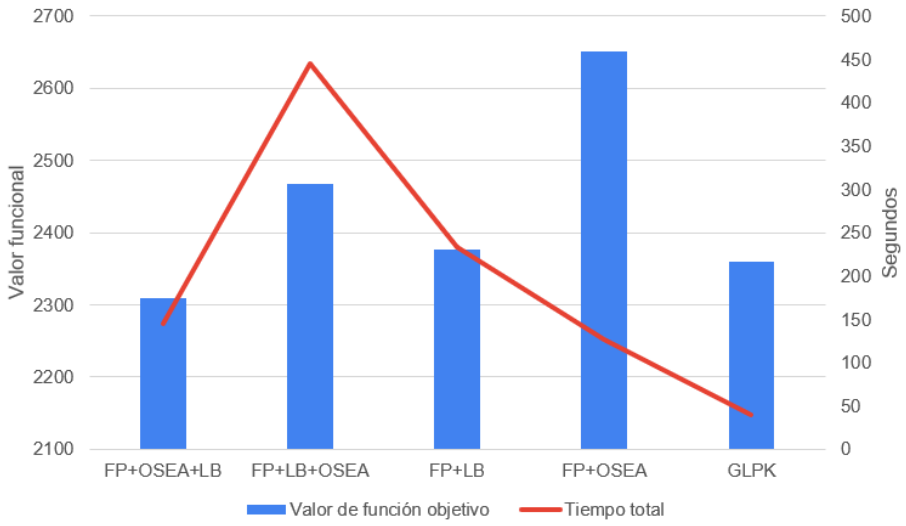

Figura 24 - Tiempos y valores funcionales promedio instancia 7

En la Figura 25 se presentan los resultados alcanzados para la instancia 8. En materia tiempo de cómputo *OSEA* vuelve a mostrar buenos resultados pero se hace aún más notorio que el método más relevante en cuanto a mejora de la solución inicial es el de *FP+LB* ya que es el que muestra los mejores resultados funcionales. No obstante, el mejor resultado alcanzado es mediante el método GLPK. tanto en tiempos de cómputo como valor funcional. Se empieza a observar que al aumentar el tamaño empieza a ser mayor el esfuerzo computacional para los métodos heurísticos por el tiempo de cómputo insumido, pero sigue siendo muy fácil para GLPK encontrar buenas soluciones en comparación.

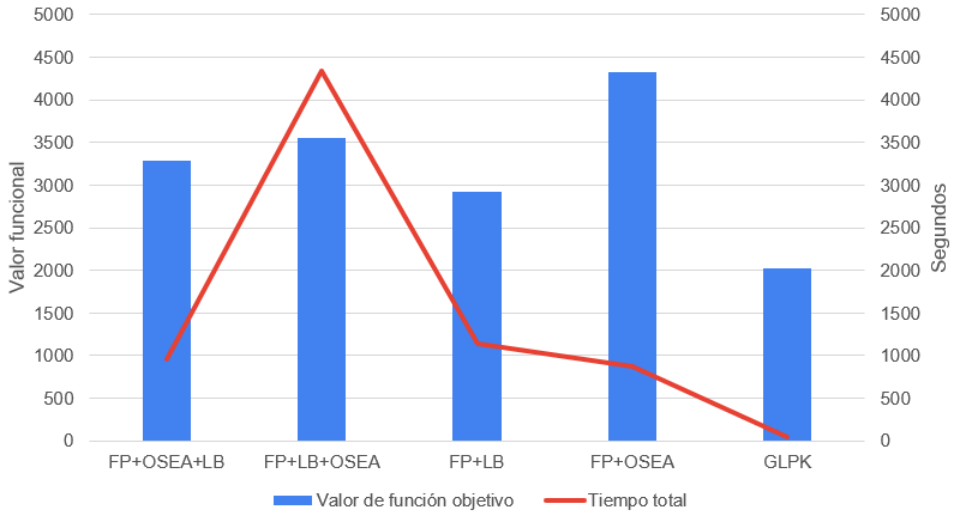

Figura 25 - Tiempos y valores funcionales promedio instancia 8

La instancia 9 es la instancia con mayor número de empresas (40) y por ende la de mayor tamaño. Esta tiene la particularidad de que en tiempos de cómputo mayores a cinco horas, en algunas instancias no se logra un resultado mediante los métodos heurísticos. Este es el único caso en el que sucede esto y comprueba la relación con el tamaño del problema que tienen todos los métodos heurísticos implementados. En estos casos se detuvo la corrida y se paso a la siguiente, pues se consideró que se supera un tiempo eficiente para la aplicación real. De las 20 corridas tomadas, en 11 casos no se llegó a una solución, incluyendo todas las ejecuciones del método *FP+LB+OSEA*, por lo que no es presentado en la Figura 26. Es de destacar que para este método, así como se vio en las instancias anteriores, la ejecución de *OSEA* luego de *LB* sería innecesaria por lo que sólo suma tiempo de cómputo haciendo que no se encuentren los resultados finales. Igualmente, el principal motivo de no culminar el método en menos de 5 horas es la obtención de la solución inicial mediante *FP.* Se observa como baja el rendimiento de las heurísticas en cuanto al tiempo de resolución siendo los métodos de *FP* y *LB* los principales motivos.

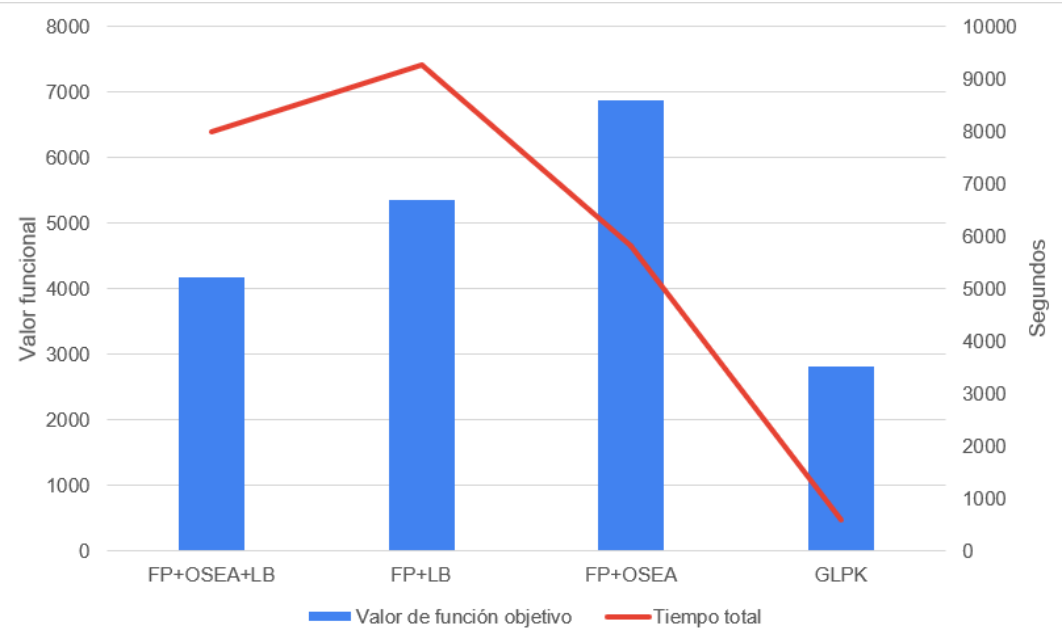

Figura 26 - Tiempos y valores funcionales promedios instancia 9

De estos resultados se puede afirmar que teniendo en cuenta los tiempos tan bajos para encontrar soluciones por el paquete de *software* GLPK, los problemas son de baja dificultad para demostrar su factibilidad. A medida que aumenta el tamaño del problema, los métodos heurísticos insumen mayor tiempo de cómputo, se considera que el método *OSEA* tiene una relación directa con este factor pero que son *FP* y *LB* más sensible a este factor.

Evaluando los resultados obtenidos de los métodos según el tamaño de la instancia se obtiene que mientras aumenta el tamaño del problema se requieren mayores tiempos de cómputo como era de esperarse. Se destaca que la principal demora de los métodos hasta cierto tamaño es *FP*, insumiendo la mayor cantidad del tiempo. Luego empeora el rendimiento LB conforme aumenta el tamaño del problema ya que empeora sus tiempos con alta tasa, algo que fue observado en los resultados del Apéndice I: Estado del Arte al ser LB un método de *neighborhood*. En cuanto a *OSEA,* aunque se identifica que aumenta los tiempos de cómputo con el tamaño, no se observa gran variación comparado con el resto de métodos heurísticos.

6.5. Solución al problema de ruteo PTIC

En esta sección se presentan y analizan los resultados obtenidos tras la formulación y resolución del problema MILP correspondiente a la primera instancia, con especial atención a su evaluación en un contexto real. Una vez alcanzada la solución del modelo para el problema de ruteo de vehículos, utilizando los datos de la instancia del PTIC, se procedió a la representación gráfica de los recorridos asignados para uno de los días. Se procede a mostrar los resultados de algunas de las rutas obtenidas mediante la solución del mejor resultado alcanzado en la implementación. El resto de recorridos del problema del PTIC se pueden acceder en Apéndice V: Representación visual de los recorridos.

En este tipo de visualización se busca validar, interpretar y comunicar de manera clara los resultados generados por el modelo. Además, dado que se emplearon métodos heurísticos en la obtención de las soluciones no garantiza la optimalidad del resultado. Por lo tanto, las rutas obtenidas podrían ser mejoradas mediante técnicas complementarias de refinamiento o ajuste, que permitan mejorar aún más los recorridos propuestos. Estas técnicas buscan ser algo sencillo y aplicable o consejos de buenas prácticas extraídos de Artucio E. *et al.* (2023). Se destacan dos consejos para la creación de ruteos de vehículos. Primero, se aconseja construir rutas comenzando con la parada más lejana del depósito. Esta lógica tiende a minimizar la distancia total recorrida y evita solapamientos o trayectos innecesarios. Otro consejo es que la secuencia de paradas en una ruta por carretera debería formar una figura de lágrima, sin cruces de caminos. El vehículo se aleja gradualmente del depósito realizando entregas o visitas, y luego retorna por un camino distinto sin necesidad de retroceder o cruzarse con tramos ya recorridos.

Teniendo esto en mente, en la Figura 27 se muestra el plano del PTIC en donde se ubican las diferentes empresas clientes del servicio de recolección de residuos, el centro de clasificación (“depósito”) y las aristas que representan los recorridos que se hacen. Además, se identifica el orden del recorrido según el número que esté colocado al lado del nodo. Se distingue las empresas por las letras C (del inglés *customer*) y a su lado el número en el que se atenderá. Específicamente, se representa el recorrido obtenido para el primer día del mes.

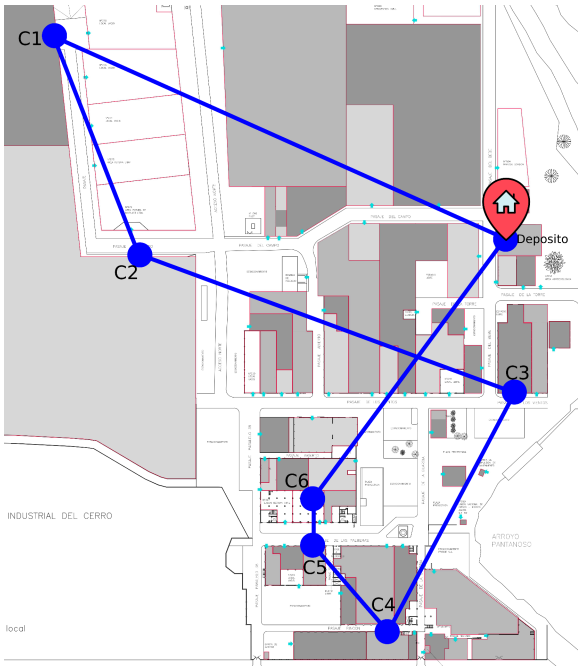

Figura 27 - Recorrido para el primer día del mes de la mejor solución

En color azul se observan los resultados del recorrido. Se puede observar que el primer consejo relacionado con comenzar con el punto más alejado al depósito parece cumplirse. Por otro lado, se observa que, como hay un cruce en el recorrido, cambiando el orden del nodo C3 por el C6 se lograría una disminución en el trayecto total. Con el cambio propuesto se lograría un cambio en el valor de la función objetivo de 6870,9 a 6660,8 unidades de distancia. Extendiendo este razonamiento al resto de los recorridos, como se observa en Apéndice V: Representación visual de los recorridos, se obtiene que el valor de la función objetivo decrece a

5852,8 unidades de distancia logrando una mejora de aproximadamente 15% en la solución y que mejora aún más la solución obtenida mediante GLPK. Comparando este escenario con la realidad, en donde se recorren 6643,5 unidades de distancia sin cumplir con la demanda de todos los clientes, el resultado obtenido mejora en 790,6 unidades de distancia cumpliendo con la recolección de todos los residuos valorizables.

7. Conclusiones

En este documento se presentó el proyecto realizado sobre la aplicación de métodos heurísticos en problemas MILP y los resultados obtenidos tras su uso en un problema real de ruteo de vehículos dentro del PTIC. A partir del Apéndice I: Estado del Arte se determinaron tres heurísticas llamadas *Feasibility Pump, Objective Scaling Ensemble Approach* y *Local Branching* como objeto de estudio del proyecto. Los resultados alcanzados mediante las heurísticas se compararon con los obtenidos mediante métodos GLPK. Se prestó especial interés en la complejidad del problema, la calidad de los resultados y los tiempos de resolución.

Como primera conclusión, afirmamos el cumplimiento del primer objetivo del proyecto, al lograr la elaboración del Apéndice I: Estado del Arte. En este se relevaron métodos de resolución eficientes de la literatura de los últimos cinco años para mejorar los tiempos de resolución de problemas modelados a través MILPs. Se analizaron un total de 85 métodos heurísticos en donde la metodología que mejor se adapta a las distintas características del problema son los métodos de *neighborhood*. De esta forma se tomaron de esta categoría los métodos heurísticos a implementar y analizar.

Con respecto al segundo objetivo, el cual fue trabajar en el caso de estudio del PTIC, se logró desarrollar, implementar y resolver una formulación MILP de ruteo de vehículos con capacidad y ,multi período. Se implementaron y evaluaron cuatro métodos heurísticos resultantes de la aplicación de *FP, OSEA* y *LB* en distintas secuencias. Cada una comenzó por la obtención de una solución inicial mediante *FP* y luego, mediante *OSEA, LB* o ambas se mejora la solución inicial, obteniendo los cuatro métodos estudiados. Estos se compararon con la resolución mediante el *software* de optimización GLPK. Ambos métodos de mejora (*OSEA* y *LB*) logran mejorar el resultado inicial concluyendo la correcta implementación de las mismas y en particular la fortaleza de *OSEA* es la reducción en los tiempos de cómputo mientras que de *LB* se destaca la capacidad de mejora de una solución en términos de su valor funcional.

El mejor método obtenido fue *FP+OSEA+LB* el cual, en el mejor de los casos, logra la resolución del problema del PTIC obteniendo una mejor solución y en la mitad del tiempo requerido para encontrar una solución con GLPK. Por otro lado, se determina gracias a la generación de otras instancias que la eficiencia de los métodos cambia según el tamaño de los problemas y el tamaño de la región factible. En estos casos, se concluye que los tiempos de cómputo aumentan para todos los métodos conforme aumenta el tamaño del problema o disminuye el tamaño de la región factible afectando en mayor medida a *FP* y *LB*.

Una vez obtenidos los mejores resultados a través del método *FP+OSEA+LB* para el caso real, se logró, aplicando dos técnicas complementarias de mejora, un resultado mejorado en un 15%. Este resultado representa una mejora en 600 unidades de distancia a los recorridos actuales del PTIC por lo que se considera un caso de éxito la aplicación de las heurísticas.

Finalmente, se determina ventajosa la utilización de métodos heurísticos frente a métodos exactos cuando se enfrentan problemas MILP difíciles, o sea, donde los tiempos de cómputo para hallar soluciones son altos. Se afirma esto ya que se comprobó que, con métodos heurísticos, se puede lograr mejores soluciones en menores tiempos. Por otro lado, se determina que para problemas de baja dificultad, la utilización de métodos heurísticos, al menos de fijación y *neighborhood* representan una desventaja tanto en tiempos de cómputo como valor funcional. Se destaca la generalidad de los métodos pero principalmente su buen

funcionamiento frente a problemas con variables binarias. La posibilidad de utilizar métodos heurísticos para obtener insumos cuantitativos para la toma de decisiones en tiempos razonables se considera de gran valor.

Se considera que una posible mejora a futuro puede radicar en la implementación de un nuevo método de resolución inicial o en la mejora de *FP* a través de algunos de los métodos abordados en la literatura y presentados en el Apéndice I: Estado del Arte. Esto permitiría mejorar los tiempos y calidad de la solución, ya que *FP* es el método heurístico que más tiempo de cómputo insume en la mayoría de los casos, obteniendo una solución de baja calidad. Por otro lado, la literatura relacionada con los problemas de ruteo de vehículos suelen aprovechar propiedades específicas de estos problemas. Surgen otras posibilidades de trabajo a futuro como incurrir en uno de estos métodos. Además, existen métodos heurísticos más complejos como pueden ser métodos de aprendizaje automático o de descomposición que podrían ser utilizados y comparar la eficiencia con los métodos implementados en el documento.

Por otro lado, resulta interesante implementar estas heurísticas en formulaciones y problemas diferentes con el objetivo de evaluar los métodos en una mayor cantidad de problemas y evaluar su generalidad. Además, se podría adaptar y generalizar el modelo VRP propuesto para el problema del PTIC para que permita evaluar otros problemas similares, permitiendo una mayor cantidad de vehículos de recolección y mayor cantidad de centros de recolección. Esto permitiría la resolución de problemas de recolección de residuos a nivel barrial o departamental, facilitando una planificación más eficiente y localizada de las rutas de recolección y la asignación óptima de recursos. Por último, a partir de la comparación de distintos *software* de optimización, la utilización o la implementación de las heurísticas en paquetes de *software* comerciales podría representar una gran ventaja para problemas de alta complejidad.

Referencias

Achterberg, T., & Wunderling, R. (2013). Mixed integer programming: Analyzing 12 years of progress. En *Facets of combinatorial optimization* (pp. 449–481). Springer. https://doi.org/10.1007/978-3-642-38189-8_18

AMPL. (s.f.). A mathematical programming language. Recuperado el 10 de junio de 2025, de https://ampl.com

Artucio, E., Alsó, J., & Bertón, G. (2023). Transporte: Ruteo de vehículos. En *Elementos de Gestión Logística* (curso). Instituto de Ingeniería Mecánica y Producción Industrial, Facultad de Ingeniería, Udelar.

Bertacco, L., Fischetti, M., & Lodi, A. (2007). A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization, 4*(1), 63–76. https://doi.org/10.1016/j.disopt.2006.10.001

Berthold, T., Lodi, A., & Salvagnin, D. (2018). Ten years of feasibility pump, and counting. *EURO Journal on Computational Optimization, 7*(1), 1–28. https://doi.org/10.1007/s13675-018-0109-7

Boschetti, M. A., Letchford, A. N., & Maniezzo, V. (2023). Matheuristics: Survey and synthesis. *International Transactions in Operational Research, 30*(6), 2840–2866. https://doi.org/10.1111/itor.13301

Clarke, G., & Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

Chaire de recherche en distributique – HEC Montréal. (s.f.). Data sets. Recuperado el 12 de abril de 2025, de http://neumann.hec.ca/chairedistributique/data/

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–84. https://doi.org/10.1287/mnsc.6.1.80

De Prado, J., Moscatelli, S., Piñeyro, P., Tansini, L., & Viera, O. (2022). Solving the multi depot vehicle routing problem with limited supply capacity at the depots with a multi phase methodology. En *Metaheuristics and Nature Inspired Computing. Communications in Computer and Information Science* (pp. 198–211). Springer. https://doi.org/10.1007/978-3-030-94216-8_15

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming, 104*(1), 91–105. https://doi.org/10.1007/s10107-004-0570-3

Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming, 98*(1), 23–47. https://doi.org/10.1007/s10107-003-0395-5

Fischetti M. & Lodi A., T. (2014). *Heuristics in mixed integer programming*. Springer.

Goeke, D., Roberti, R., & Schneider, M. (2019). Exact and heuristic solution of the consistent vehicle-routing problem. *Transportation Science, 53*(1), 1–20. https://doi.org/10.1287/trsc.2018.0864

Guasque, A., & Balbastre, P. (2022). Evaluation and comparison of integer programming solvers for hard real-time scheduling. *IEICE Transactions on Information and Systems, 105*(D10), 1726–1733. https://doi.org/10.1587/transinf.2022EDP7073

Gurobi Optimization, LLC. (s.f.-a). Academic program and licenses. Recuperado el 10 de abril de 2025, de https://www.gurobi.com/academia/academic-program-and-licenses/

Gurobi Optimization, LLC. (s.f.-b). Gurobi Optimizer Reference Manual – Command-line interface. Recuperado el 10 de abril de 2025, de https://docs.gurobi.com/projects/optimizer/en/current/reference/misc/commandline.html

Gurobi Optimization, LLC. (s.f.-c). Gurobi 8 performance benchmarks. Recuperado el 10 de abril de 2025, de https://www.gurobi.com/resource/gurobi-8-performance-benchmarks/

Gurobi Optimization, LLC. (s.f.-d). How does presolve work? Gurobi Support Center. Recuperado el 10 de abril de 2025, de https://support.gurobi.com/hc/en-us/articles/360024738352

Gurobi Optimization, LLC. (s.f.-e). How do I install Gurobi Optimizer? Recuperado el 10 de abril de 2025, de https://support.gurobi.com/hc/en-us/articles/4534161999889

Gurobi Optimization, LLC. (s.f.-f). Mixed-integer programming (MIP): A primer on the basics. Recuperado el 10 de abril de 2025, de https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/

Havet, F. (s.f.). Polynomiality of linear programming. Recuperado el 12 de junio de 2025, de https://www-sop.inria.fr/members/Frederic.Havet

Hillier, F. S., & Lieberman, G. J. (2010). *Introduction to operations research* (9.ª ed.). McGraw-Hill Higher Education.

Huang, T., Ferber, A., Tian, Y., Dilkina, B., & Steiner, B. (2023). Local branching relaxation heuristics for integer linear programs. En *Lecture Notes in Computer Science* (Vol. 14100, pp. 273–289). Springer. https://doi.org/10.1007/978-3-031-33271-5_7

IBM. (s.f.). Solving mixed integer programming problems (MIP). Recuperado el 10 de abril de 2025, de https://www.ibm.com/docs/en/icos/22.1.1?topic=optimization-solving-mixed-integer-programming-problems-mip

Jafarian-Moghaddam, A. R., & Shahpoori-Arany, S. (2024). How did VRP grow? Towards sustainable models by developing a novel classification. *Sustainable Futures, 8*, 100194. https://doi.org/10.1016/j.sftr.2024.100347

Jaikishan, T. S., & Patil, R. (2019). A reactive GRASP heuristic algorithm for vehicle routing problem with release date and due date incurring inventory holding cost and tardiness cost. *International Journal of Computer Applications, 182*(39), 28–32. https://doi.org/10.5120/ijca2019918689

Joncour, C., Kritter, J., Michel, S., & Schepler, X. (2023). Generalized relax-and-fix heuristic. *Computers & Operations Research, 149*, 106190. https://doi.org/10.1016/j.cor.2022.106038

Klotz, E., & Newman, A. (2013). Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science, 18*(1–2), 18–32. https://doi.org/10.1016/j.sorms.2012.12.001

Løkketangen, A., Jornsten, K., & Storøy, S. (1994). Tabu search within a pivot-and-complement framework. *International Transactions in Operational Research, 1*(3), 305–317.

Lüer, A., Benavente, M., Bustos, J., & Venegas, B. (2009). El problema de rutas de vehículos: Extensiones y métodos de resolución, estado del arte. En *Actas del Workshop Internacional EIG 2009 – 3er Encuentro Informática y Gestión* (Vol. 558). Temuco, Chile.

Makhorin, A. (2009). *GNU Linear Programming Kit: Reference Manual* (Version 4.38, Draft). GNU Project.

Mexi, G., Berthold, T., & Salvagnin, D. (2023). Using multiple reference vectors and objective scaling in the feasibility pump. *EURO Journal on Computational Optimization, 11*(1), 1–14. https://doi.org/10.1016/j.ejco.2023.100066

Noriega González, C. (2023). El algoritmo Branch & Cut en programación lineal entera. *Trabajo de Fin de Grado, Universidad de Valladolid*. UVaDOC. Recuperado el 11 de junio de 2025, de https://uvadoc.uva.es/handle/10324/63200

Pochet, Y., & Wolsey, L. A. (2006). *Production planning by mixed integer programming*. Springer.

SCIP Optimization Suite. (s.f.). Modules – Versión 9.2.1. Recuperado el 10 de junio de 2025, de https://scipopt.org/doc-9.2.1/html/modules.php

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. En *Lecture Notes in Computer Science, 1520*, 417–431. https://doi.org/10.1007/3-540-49481-2_30

Spielman, D. A., & Teng, S.-H. (2004). Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM, 51*(3), 385–463. https://doi.org/10.1145/990308.990310

Riccardi, A., Minisci, E., Akartunalı, K., Greco, C., Kershaw, A., & Hashim, A. (2021). Introduction to optimisation. En *Optimization under uncertainty with applications to aerospace engineering* (pp. 223–268). Springer. https://doi.org/10.1007/978-3-030-60166-9

Vrieswijk, M. (2023). *Comparing solvers: Piecewise linear & logarithmic approximations* (Master's thesis, Wageningen University & Research).

Wolsey, L. A. (1998). *Integer programming*. Wiley-Interscience

Zhang, J., Liu, C., Li, X., Zhen, H.-L., Yuan, M., Li, Y., & Yan, J. (2022). A survey for solving mixed integer programming via machine learning. *Neurocomputing, 519*, 205–217. https://doi.org/10.1016/j.neucom.2022.11.024

Zhang, W., & Nicholson, C. D. (2019). Objective scaling ensemble approach for integer linear programming. *Computers & Operations Research, 105*, 1–13. https://doi.org/10.1007/s10732-019-09418-9

Apéndices

Apéndice I: Estado del Arte

Métodos de resolución eficientes para problemas de producción complejos

Estado del arte

Facultad de Ingeniería - Universidad de la República

Autores:
Felipe Bocca
Francisco Ferriolo

Tutor:
Pedro Piñeyro

Agosto 2025

Índice:

Siglas y acrónimos

- B&B: Branch & Bound
- B&P: Branch & Price
- CCG: Column & constraint generation
- CVRS: Constraint violation reduction search
- DW: Dantzig–Wolfe
- FFM: Feasible Finder Model
- FP: Feasibility Pump
- GED: Distancia de edición de gráfico
- GNN: Graph neural network
- HD: Distancia de Hamming
- ILP:Problemas de programación lineal
- IPS: Inner parallel set
- LB: Local Branching
- LNS: Large Neighborhood Search
- LS: Búsqueda local
- MILP: Programación lineal entera mixta
- MNA: Modified Newton Algorithm
- MP: Problema maestro
- OSEA: objective scaling ensemble approach
- RD: Distancia de relajación
- RENS: Relaxation Enforced Neighborhood Search
- RINS: relaxation induced neighborhood search
- RL: Relajación Lagrangiana
- SO: Software de optimización

- SP: Problema secundario
- TSRO: Two-stage robust optimization
- VPLS: Variable Perturbation Local Search

1. Introducción

El objetivo de este documento es brindar información sobre distintos métodos heurísticos aplicables a problemas de programación lineal entera mixta (MILP). Estos métodos son utilizados con el objetivo de disminuir el esfuerzo computacional necesario para la resolución de los problemas MILP. Los problemas MILP son herramientas utilizadas para modelar y resolver problemas complejos de toma de decisiones, principalmente en áreas de optimización industrial y científica. La importancia de esta herramienta se fundamenta en las aplicaciones del modelado en la actualidad, entre ellas se destacan la planificación y programación de la producción, enrutamiento de vehículos, optimización de cadenas de suministro, diseño de redes y telecomunicaciones, optimización del diseño industrial y optimización en la industria energética. Dada la alta complejidad computacional de este tipo de problemas *NP-hard*, los algoritmos de *software* de optimización matemática, basados generalmente en métodos exactos y completos como *Branch & Bound* (B&B), no siempre son capaces de encontrar la solución en tiempos razonables para la aplicación práctica. Justamente, los problemas *NP-hard* son problemas cuya dificultad crece exponencialmente con el tamaño de la entrada, lo que significa que el tiempo necesario para resolverlos puede volverse inmanejable incluso con pequeños aumentos en los datos. Es debido a esta característica que el uso de herramientas como las heurísticas se convierte en una necesidad ya que una heurística es un enfoque o algoritmo que busca una solución aproximada a un problema difícil (como los NP-hard), sin garantizar una solución óptima. En lugar de explorar todas las posibles soluciones, una heurística emplea reglas prácticas o estrategias para reducir el espacio de búsqueda y encontrar buenas soluciones en un tiempo razonable.

En las décadas de 1960 y 1970, el término "heurística" comenzó a utilizarse ampliamente en el campo de la investigación operativa y la optimización. Como se introdujo, las heurísticas son métodos no exactos que no garantizan la obtención de una solución óptima pero tienden a ser más rápidos que otros métodos exactos y logran la obtención de soluciones con una calidad aceptable. Se recomienda para la introducción del lector en el campo investigado, comenzar la revisión de la literatura por documentos generales descriptivos del funcionamiento e introducción de algunas heurísticas básicas como por ejemplo los referenciados en Fischetti & Lodi (2010) y Shoja & Axehill (2023).

Para la búsqueda bibliográfica revisada en este documento se utilizó la plataforma Timbó, a cargo de la Agencia Nacional de Investigación e Innovación (ANII), que permite acceder a bibliografía y literatura científica-tecnológica a nivel mundial. En esta plataforma se accedió a colecciones de suscripción como Scopus, SpringerLink y ScienceDirect que se utilizaron para la búsqueda bibliográfica. Estas colecciones reconocidas a nivel mundial proporcionan acceso a artículos científicos e investigaciones y publicaciones académicas de alta calidad. En primera instancia, para la revisión de la literatura, se optó por la búsqueda a través de palabras claves utilizadas en el tipo de documento buscado, pero este método se consideró inefectivo. Los artículos encontrados mayoritariamente estaban enfocados en la formulación del modelo y el aprovechamiento de particularidades del mismo, sin enfocarse en la heurística utilizada y su forma de implementación que era el principal interés de la revisión. Por lo tanto, una vez revisados los documentos Fischetti & Lodi (2010) y Shoja & Axehill (2023), se optó por expandir la revisión a través de la búsqueda de artículos que citan a alguno de los documentos que destacamos como fundacionales de los métodos heurísticos estudiados, estos son los nombrados en las Referencias con años anteriores a 2006. Este método se consideró exitoso ya que los artículos encontrados tenían las características deseadas y debió acotar la búsqueda en los últimos 5 años (2019 - 2024), debido a la cantidad elevada de artículos y para

favorecer el entendimiento actual prestando atención en nuevos enfoques e innovaciones que se han desarrollado.

El resto de este documento se organiza de la siguiente manera. En la Sección 2 se toman conceptos claves para comprender y profundizar diferentes definiciones y métodos relacionados con la resolución de MILP. La Sección 3 define la clasificación de las heurísticas que se utilizan en este documento y una breve descripción de cada una de ellas. La Sección 4 y sus subsecciones analizarán las diferentes propuestas de heurísticas, dividiendo cada subsección según la clasificación propuesta en la Sección 3. Por último, se brindan tendencias sobre el futuro de las heurísticas, lecciones aprendidas y conclusiones del documento en la Sección 5.

2. Conceptos claves

Esta sección ofrece una visión integral de los principios de MILP, incluyendo su formulación matemática y las diversas estrategias de solución disponibles. Mediante este análisis, se pretende establecer una base sólida para la comprensión de problemas MILP y sus derivaciones. Además, se discutirán diferentes algoritmos, heurísticas y metodologías con el objetivo de introducir al lector en diferentes heurísticas para conocer su funcionamiento. La sección se divide en varias subsecciones en donde la Subsección 2.1 introduce conceptos de programación matemática, la Subsección 2.2 especifica qué es un MILP y, luego de definirlo, se comienza a profundizar en algunos métodos de resolución. Para ello se empieza por la Subsección 2.3 en la que se define y explica la metodología exacta de *B&B*, luego se pasa a la Subsección 2.4 donde se define el método heurístico *Feasibility Pump* (FP) y a la Subsección 2.5 donde se define otra heurística llamada *Local Branching* (LB). Por último, se definen métodos de descomposición en la Sección 2.6.

2.1 Programación matemática:

Un problema de programación matemática involucra elementos, definidos en Islas & Testuri (2023), que se deben de tener en cuenta para su comprensión, entre ellos se destacan:

- Variables de decisión: son aquellos valores que determinaremos a través de la resolución del problema ya que están bajo el control del tomador de decisiones. Estas variables son fundamentales en el proceso de optimización ya que su meta es maximizar o minimizar ciertos objetivos, expresados a través de la función objetivo, y sujetos a un conjunto de condiciones o restricciones específicas.

- Parámetros: son valores que surgen del análisis de la realidad y que no están bajo el control de quien toma las decisiones, por ejemplo, la capacidad productiva de una fábrica en la actualidad.

- Función objetivo: es una expresión que se quiere optimizar y que vincula las variables de decisión con algunos de los parámetros. Para determinado problema puede existir más de un objetivo posible a seleccionar y se dirá que es un problema multiobjetivo.

- Restricciones: son expresiones que involucran variables y parámetros que representan las exigencias que debe cumplir la solución. El espacio determinado por el conjunto de restricciones se define como la región factible del problema.

La programación matemática se entiende como el proceso de modelar, analizar y resolver problemas utilizando modelos matemáticos. El objetivo es ajustar las variables de decisión dentro de los límites impuestos por las restricciones, con el fin de optimizar la función objetivo. Se considera solución óptima a aquella que optimiza la función objetivo cumpliendo con todas las restricciones y valor óptimo al valor que alcanza la función objetivo en la solución óptima. La programación matemática se clasifica según la manera en que se modela el problema y el tipo de variables utilizadas. A continuación, se listan los diferentes tipos de problema de programación matemática:

- Programación lineal (LP): se denomina problema de programación lineal cuando tanto la formulación de la función objetivo como las restricciones son funciones lineales, además, las variables de decisión toman valores continuos. En este tipo de problemas la región factible es convexa por lo que permite la resolución mediante algoritmos con tiempos razonables de cómputo.
- Programación lineal entera (ILP): son problemas de programación lineal en donde las variables de decisión deben tomar valores discretos. Se trabaja en una región factible no convexa por lo que su resolución suele requerir mayor esfuerzo computacional con respecto al LP. Dentro de estos se destacan los casos en que la variable de decisión toma valores binarios.
- Programación lineal entera mixta (MILP): se denomina un problema de programación lineal entera mixta cuando las variables de decisión toman valores discretos y lineales. Este tipo de formulación de problemas es el más amplio de los tres y permite representar una mayor cantidad de problemas.

En la elaboración de este estado del arte se presta principal atención a los ILP y MILP. La elección de la priorización, como ya fue introducido, se debe en primer lugar a la gran aplicabilidad del modelado y, en segundo lugar, debido a que frecuentemente, debido al tamaño de las instancias, se vuelve inviable la resolución a través de métodos de optimización convencionales por el elevado tiempo de cómputo requerido.

2.2 Programación lineal entera mixta

Un problema de programación lineal entera mixta busca optimizar una función objetivo lineal, sujeta a restricciones lineales, con variables de decisión discretas y continuas. Generalmente, las variables utilizadas son continuas y las variables discretas aparecen por estricta necesidad de las mismas. A continuación, se expone una formulación general de un problema MILP en donde se busca minimizar una función objetivo, sujeta a restricciones, y se separan las variables discretas de las continuas.

Minimizar $c^T_1 x + c^T_2 y$ (1)

Sujeto a $A_1 x + A_2 y \leq b$ (2)

$x \geq 0$ (3)

$x_j \in \mathbb{Z} \quad \forall j \in J$ (4)

$y_i \in \mathbb{R} \quad \forall i \in I$ (5)

Donde:

- J define al conjunto de índices tal que $J = \{1,\dots n\}$.
- I define al conjunto de índices tal que $I = \{1,\dots, o\}$.
- $x = (x_1, x_2,\dots, x_{n-1}, x_n)$ es el vector de las variables de decisión discretas de tamaño *n*.
- $y = (y_1, y_2,\dots, y_{o-1}, y_o)$ es el vector de las variables de decisión continuas de tamaño *o*.

- $c_1 = (c_{1,1}, c_{1,2}, \ldots, c_{1,n-1}, c_{1,n})$ y $c_2 = (c_{2,1}, c_{2,2}, \ldots, c_{2,o-1}, c_{2,o})$ son los vectores coeficientes de la función objetivo.

- $A_1 \in \mathbb{R}^{mxn}$ y $A_2 \in \mathbb{R}^{mxo}$ son matrices que definen las restricciones lineales.

- $b = (b_1, b_2, \ldots, b_{m-1}, b_m)$ es el vector de términos independientes de las restricciones.

Un concepto muy utilizado en este tipo de problemas es la relajación (*relaxation*). Se dice que se relaja el dominio de una variable discreta cuando se expande su dominio permitiendo tomar valores continuos dentro de un rango específico. Este mecanismo se logra modificando la restricción (4) del modelo y extendiendo el dominio al conjunto de los reales. Este concepto es común en problemas de optimización, ya que simplifica el problema original facilitando su resolución mediante técnicas de programación lineal.

Otra herramienta utilizada es la de fijar donde una variable toma un valor específico para pasar a tratarla como un parámetro en el problema. Este proceso es utilizado para disminuir el tamaño del problema tras añadir una (o varias) restricciones para indicar estas nuevas igualdades. Otro concepto muy utilizado en MILP es el de Dualidad. La Dualidad indica que para todo problema MILP se puede definir un problema dual, distinguiendo el problema primal (P) del dual (D) que se definen de la siguiente manera:

$$\text{(P)} \quad \begin{aligned} &\text{mín } c^T x \\ &\text{s.a} \\ &Ax = b \\ &x \geq 0 \end{aligned} \qquad \text{(D)} \quad \begin{aligned} &\text{máx } \lambda^T b \\ &\text{s.a} \\ &\lambda^T A \leq c^T \\ &\lambda \in \Re^m \end{aligned}$$

.

Se conoce debido al teorema de dualidad, que si P tiene una solución óptima, entonces D tiene solución óptima y el valor óptimo de ambos problemas es coincidente definiéndose su solución como:

$$\lambda^T = c_B^T B^{-1} \quad (6)$$

Donde λ^T es una solución óptima y donde B se define como la matriz básica óptima del problema P. La extensión de este concepto se debe a la gran utilidad que presenta. En primer lugar, el problema dual es utilizado para proporcionar cotas de la solución del problema original, permitiendo el funcionamiento de diversos métodos. También ofrece información relevante sobre la sensibilidad del problema a cambios en las restricciones, esto puede ser útil para identificar restricciones críticas o diseñar estrategias de ajuste de parámetros en problemas complejos.

Existen diversos métodos para la resolución de los problemas MILP, estos se dividen en métodos exactos y métodos heurísticos. Los métodos exactos garantizan una solución óptima una vez concluido y, entre ellos, se encuentran el método de *Branch & Bound* (B&B) apoyado en el método Simplex. Este último se utiliza específicamente para resolver problemas lineales continuos, y forma la base para resolver la relajación lineal de un MILP en el método B&B. En cuanto a los métodos heurísticos, estos son técnicas que buscan encontrar soluciones aceptables en tiempos razonables, sin garantizar su optimalidad. Se destacan métodos como *Feasibility Pump* (FP) introducido en Fischetti *et al.*, (2005) y *Local Branching* (LB) en Fischetti

& Lodi (2003) como métodos clásicos de suma relevancia, entre otros. A continuación, se decide explicar tres metodologías de resolución de problemas, debido a su alta utilización en varios de los documentos de la literatura.

2.3 *Branch & Bound*

La resolución de problemas de formulación MILP mediante *software* de optimización (SO) matemática se fundamenta en la aplicación del método *B&B* ya que ha sido el método base de los SO más utilizados a nivel mundial como CPLEX, Gurobi, y SCIP, entre otros. Este método es un procedimiento iterativo que comienza relajando el dominio de las variables discretas del problema original y a partir de la solución obtenida se ramifica en subproblemas relajados. En el método *B&B* el proceso de ramificación consiste en identificar una variable que, en la solución óptima del problema relajado, no cumpla con sus restricciones de integridad (es decir, toma un valor fraccional). Cuando esto ocurra se debe dividir la región factible en dos sub regiones disjuntas, añadiendo restricciones excluyentes para dicha variable. Este proceso se repite en cada subproblema generado, mientras permanezcan nodos activos en el árbol de búsqueda. La poda de ramas o condiciones de parada de ramificación se realiza bajo los siguientes criterios:

- La solución óptima del subproblema relajado es discreta y, por tanto, válida.
- El subproblema relajado no tiene solución factible.
- El valor objetivo del subproblema relajado no mejora la mejor solución discreta encontrada hasta el momento.

Para favorecer el entendimiento de la metodología, a continuación se presenta un pequeño ejemplo del procedimiento y la representación gráfica del mismo.
Sea P_0 el siguiente problema MILP:

Maximizar	z= 4x+y	(7)
Sujeto a	$4x + 2y \leq 11$	(8)
	$2x + 4y \leq 13$	(9)
	$x, y \geq 0$	(10)
	$x, y \in Z$	(11)

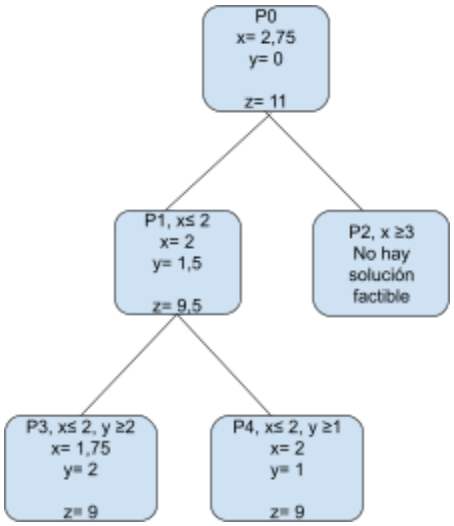

Figura 1 - Ejemplo de utilización del método B&B, extraído de Islas & Testuri (2023)

El caso de la Figura 1 ejemplifica la utilización de los tres criterios de poda, luego de la primera ramificación, la poda de la rama derecha se debe al criterio de no factibilidad. En el siguiente nivel de ramificación, por un lado observamos que la solución óptima del subproblema generado del lado derecho es entera y por tanto válida. Por otro lado, observamos que la solución óptima del lado izquierdo no es mejor que la solución discreta ya encontrada por lo que se procede a la poda de la misma. Resulta claro que a través del método se ha recorrido todo el espacio factible, encontrando que (*x=1, y=1)* es la solución óptima del problema.

2.4 Feasibility Pump

Feasibility Pump (FP) es un método iterativo utilizado para obtener una solución inicial en problemas de programación entera, su objetivo es lograr la convergencia entre dos trayectorias de puntos. La primera trayectoria se genera mediante el redondeo de las soluciones obtenidas tras relajar al dominio de las variables discretas. Esta solución redondeada, sin embargo, no garantiza estar dentro de la región factible. La segunda trayectoria se obtiene resolviendo un nuevo problema lineal, cuyo objetivo es encontrar el punto dentro de la región factible relajada que minimice la distancia al punto generado en la primera trayectoria. El proceso se repite de manera iterativa hasta que ambas trayectorias convergen en un mismo punto, o bien, se alcance un criterio de terminación predefinido. La principal ventaja que se obtiene del método es que el nuevo problema que genera la trayectoria 2 es un problema con variables continuas, que como ya se explicó, facilita su resolución. FP busca la obtención de factibilidad de la solución de dos formas distintas. La primera trayectoria busca la factibilidad de las variables de forma que pertenezcan a valores discretos logrando satisfacer las restricciones de integralidad y la segunda trayectoria busca la factibilidad cumpliendo el resto de restricciones del problema. De esta forma, se obtienen puntos discretos y pertenecientes al área delimitada por la región factible del problema.

El método fue introducido en Fischetti *et al.*, (2005) donde fue planteado para variables exclusivamente binarias. Se determina la función de distancia (Δ) como la norma L_1 y la forma de redondeo queda determinada por la función de redondeo escalar ($[\cdot]$) al valor entero más cercano $\tilde{x}$. Por último, la función objetivo de la segunda trayectoria de puntos queda

determinada por la ecuación (12).

$$min_{x \in \mathbb{R}^n} \{\Delta(x, \tilde{x}) \ : \ Ax \geq b\} \quad (12)$$

El método comienza con la obtención de un punto discreto $\tilde{x}$ al redondear la solución del problema relajado. Luego, se busca hallar un nuevo punto *x** minimizando la distancia entre este y el discreto hallado en la parte anterior, bajo las restricciones del problema original. Si se encuentra una solución al problema, pero se cumple que Δ(x*,x̃) ≠ 0, entonces se redondea el valor de *x** y se obtiene un nuevo $\tilde{x}$ para luego usar en la ecuación (12). Repitiendo este proceso se define el método iterativo. El método culmina cuando $\Delta(x, \tilde{x}) = 0$ donde ambos puntos son iguales y por lo tanto se logró obtener un punto discreto que cumple las restricciones lineales. Para evitar *loops* en el proceso iterativo se propone un cambio estocástico en algunas entradas de variables, pues un *loop* sucede cuando $[x\ ^*] = \tilde{x}$ (el valor obtenido al redondear la variable de la ecuación (12) es el mismo que el número discreto de la anterior iteración).

El método ha demostrado ser muy eficaz en la resolución de problemas MILP, por lo que diferentes autores a lo largo del tiempo han variado el método creando nuevas heurísticas y expandiendo el ámbito de ejecución, permitiendo usar FP en MILP e ILP tanto para problemas binarios como no binarios. Según Fischetti *et al.*, (2005), el método de FP se define por dos factores característicos, el primero es la forma de redondear las variables a las que se aplica la relajación, pues según la forma de realizar este redondeo el método puede cambiar el rendimiento. El otro factor evalúa el cambio en la función de distancia, existen otras opciones a considerar además de la norma 1. Además, existen otros factores que pueden afectar el rendimiento de FP como los parámetros para evitar *loops,* por ejemplo, la cantidad de variables a efectuar un cambio estocástico. Para todos estos factores no hay una metodología de asignación de valores que demuestre ser más efectiva para la mayoría de los casos, y, en general, son definidos a partir de la experimentación con distintos valores.

2.5 Local Branching

La heurística *Local Branching* es una técnica iterativa diseñada para mejorar la búsqueda de soluciones en problemas de optimización binaria, introducida en Fischetti & Lodi (2003). Esta es una heurística de mejora basada en la selección de un vecindario en el cual sea posible mejorar las soluciones halladas. Por tanto, se busca un vecindario que, al resolver el problema MILP, en este se pueda encontrar una mejor solución a la previamente obtenida. En particular, en esta heurística, el vecindario estará definido por las potenciales soluciones que sólo difieren en una cantidad acotada de entradas o variables. Partiendo de la solución factible inicial, iterativamente un conjunto de variables serán seleccionadas para ser destruidas. Destruir, dentro de un problema MILP, se refiere a fijar todas las variables, exceptuando aquellas que pertenezcan al conjunto seleccionado para destruir. Al fijar el resto de las variables y resolver el problema se optimiza un problema MILP reducido del problema original, únicamente teniendo en cuenta el conjunto de variables previamente destruidas. A continuación, se determina los pasos de la heurística LB:

1. Obtener una solución inicial: LB asume la disposición de una solución inicial. Esta solución se puede obtener por ejemplo con una resolución rápida o con alguna heurística inicial.

2. Definir la vecindad: Con la solución inicial obtenida, se define una “vecindad” (o área limitada de búsqueda alrededor de la solución de referencia). Esta vecindad se establece a través de una restricción adicional, llamada "corte local", que acota el espacio factible a las soluciones que difieren en un número específico (*k)* de variables binarias con respecto a la solución inicial. La restricción que forma esta vecindad está basada en la distancia de Hamming (HD):

$$\sum_{i \epsilon S_1} (1 - x_i) + \sum_{i \epsilon S_0} x_i \leq k \qquad (13)$$

En esta restricción, S_0 y S_1 representan los conjuntos de las variables que son 1 o 0 respectivamente en la solución inicial.

3. Explorar dentro de la vecindad: Con esta nueva restricción, se resuelve el MILP en un espacio acotado de búsqueda permitiendo al SO concentrarse en una pequeña región del espacio de búsqueda y posiblemente encontrar mejoras sin tener que explorar todo el espacio factible.

4. Iteración o ajuste de la vecindad: Si se encuentra una mejor solución en esta vecindad, se convierte en la nueva solución base, y el proceso se repite desde el segundo paso iterativamente. Si por el contrario no se encuentra una mejor solución, entre otros métodos, se puede aumentar el valor de *k* para ampliar la vecindad y permitir una mayor diversificación, buscando en una región más amplia. El criterio de terminación del método radica en un tiempo límite o un número de iteraciones sin encontrar una mejor solución.

LB es particularmente útil en problemas complejos o de gran tamaño, ya que reduce la cantidad de soluciones a considerar, enfocándose en aquellas cercanas a una solución prometedora inicial. Así, se pueden encontrar buenas soluciones en menos tiempo que si se explorara el espacio de soluciones completo como hacen los métodos exactos.

2.6 Métodos de descomposición

Estos métodos se fundamentan en el principio de dividir y conquistar, que consiste en descomponer un problema grande y complejo en subproblemas más pequeños y manejables. Al hacerlo, se busca simplificar la resolución de cada subproblema, reduciendo la complejidad general y, en muchos casos, el tiempo necesario para encontrar una solución. Este enfoque permite abordar problemas grandes y/o complejos de manera más eficiente, dividiendo el trabajo en partes más sencillas y, en algunos casos, obteniendo resultados más rápidos. Los métodos de descomposición enfrentan dificultades en relación con la formulación de los subproblemas a resolver (Clímaco *et al.*, 2019), así como con su elevado costo computacional en términos de tiempo (Salvagnin *et al.*, 2024). Sin embargo, su principal ventaja radica en la capacidad para abordar problemas de gran tamaño, caracterizados por un elevado número de variables y restricciones (Salvagnin *et al.*, 2024). Entre los métodos más comunes de descomposición se encuentra la Descomposición de Benders (DB) (Mo *et al.*, 2019), Relajación

Lagrangiana (RL) (Luteberget & Sartor, 2023), Descomposición de Dantzig–Wolfe (DW) (Sadykov *et al.*, 2017) y la Generación de Columnas (Zhang *et al.*, 2023). Dado que estos métodos son ampliamente utilizados, a continuación se procederá a describir cada uno de ellos, para luego en la Sección 4.2 observar implementaciones en diversas heurísticas.

2.6.1 Descomposición de Benders

El método introducido por Benders (1962) es un enfoque exacto ampliamente utilizado, diseñado para aprovechar la estructura del problema. Su propósito es resolver problemas con variables complejas, cuya fijación reduce significativamente el tiempo de resolución. Se entiende por variables complejas a las variables discretas en términos generales, aunque también pueden incluirse otros conjuntos de variables, dependiendo de las características del problema específico. La propuesta es descomponer el problema original en dos subproblemas, separando las variables complejas del resto de variables y creando un problema maestro (MP) y un problema secundario (SP). El MP surge de considerar proyecciones, cortes en las restricciones y relajaciones y contiene las variables complejas y las restricciones relacionadas a estas variables. El SP contiene información del resto de variables, conteniendo el resto de restricciones del problema. Lo que propone el algoritmo es resolver el MP y luego utilizar el resultado para fijar variables en el SP. De esta manera, el SP identifica posibles violaciones en el resto de restricciones del problema, encontrando nuevas restricciones para el MP según se encuentre una infactibilidad. De esta forma, se define un proceso iterativo que comienza con la resolución del MP y su resultado se evalúa en el SP. En caso de obtener una solución factible termina el algoritmo, en otro caso, se vuelve a resolver el MP con nuevas restricciones. Se llega a la demostración de que existe convergencia de este método a la solución del problema original, aunque los tiempos de convergencia son en general lentos.

2.6.2 Relajación Lagrangiana

El método permite realizar una descomposición a problemas que poseen la característica de ser separables. Un problema es separable si en la función objetivo y en las restricciones están separadas las variables complejas del resto de las variables. La relajación Lagrangiana consiste en trasladar algunas de las restricciones del problema original a una nueva función objetivo, la cual se multiplica por constantes denominadas multiplicadores de Lagrange (λ_i). Las restricciones que se colocan en la función objetivo se llaman restricciones "*dualizadas*". Se busca que las restricciones sobrantes posean una estructura adecuada para una resolución sencilla. A continuación se observa el problema con todas las restricciones *dualizadas* (Bragin , 2023).

$$\text{Minimizar} \quad c^T{}_1 x + c^T{}_2 y + \lambda^T \ (b - A_1 x - A_2 y) \qquad (14)$$

$$\text{Sujeto a} \quad A_1 x + A_2 y \leq b \qquad (15)$$

$$x_j \in \mathbb{Z} \qquad \forall\, j \in J \qquad (16)$$

$$y_i \in \mathbb{R} \qquad \forall\, i \in I \qquad (17)$$

$$\lambda_m \in \mathbb{R} \qquad \forall\, i \in M \qquad (18)$$

Esta formulación permite hallar cotas inferiores del problema original (modelado como en la Sección 2.2). Esta afirmación da lugar a un nuevo método de resolución que surge del problema de maximizar los multiplicadores de Lagrange en la misma función objetivo. Para determinar la mayor cota inferior se resuelve el problema (19).

$$\text{Maximizar} \quad z(\lambda) \qquad (19)$$
$$\lambda_m \in \mathbb{R} \qquad \forall\, m \in M \qquad (20)$$

Partiendo de una solución inicial conocida, se resuelve el problema (19) y se calculan los valores de los multiplicadores de Lagrange, en donde se fijan el resto de las variables según la solución obtenida del primer problema. De esta manera, se determinan los multiplicadores de Lagrange, los cuales se fijan en el problema (14) para crear el proceso iterativo. Este proceso finaliza cuando se alcanza una solución de calidad adecuada, determinada a partir de las cotas obtenidas. Este método está vinculado a un modelo económico que resulta útil para comprender su funcionamiento ya que los valores de Lagrange pueden ser vistos como el precio sombra de las restricciones. Al incumplir una restricción, el valor del multiplicador de Lagrange asociado a dicha restricción aumenta, perjudicando el valor de la función objetivo. Por lo tanto, se regulan los valores de las variables y se busca el valor óptimo de la función objetivo.

2.6.3 Dantzig–Wolfe (DW)

Es un método exacto ampliamente utilizado para la resolución de problemas MILP de gran escala (Tonbari & Ahmed ,2023). Al igual que DB, se propone la descomposición del problema original en un MP y un SP. El método está diseñado para problemas cuyas restricciones definen un espacio limitado por un poliedro, lo que da lugar a una envolvente convexa. DW va a calcular una secuencia de puntos extremos de la envolvente convexa del problema a través del SP y trasladar esta información al MP. El MP evalúa los puntos extremos obtenidos a través de restricciones relajadas, más sencillas de resolver y actualiza la envolvente convexa para que este encuentre nuevos puntos. De esta forma se detalla un proceso iterativo en el que el SP obtienen nuevos puntos y el MP los evalúa dando nueva información para la búsqueda de nuevos puntos. Este proceso culmina cuando no existen nuevos cortes. Poder aplicar heurísticas en este método permite la resolución del MP sin tener que trabajar con una gran cantidad de datos ni largos tiempos de resolución. La combinación de DW con heurísticas trae diversos problemas que se deben de tener en cuenta para poder aplicar estos métodos de forma correcta. Según la teoría de dualidad de Lagrange, la reformulación de Dantzig Wolfe puede proponer una relajación más fina y que obtenga mejores resultados.

2.6.6 Column Generation

Nuevamente, este método propone descomponer el problema en un problema maestro y uno secundario. Es un método cuyo tiempo por iteración aumenta al aumentar el número de variables y, en mayor medida, al aumentar el número de restricciones. Al igual que los problemas anteriores, el método funciona para problemas que son “separables”, donde se crea un MP aplicando relajación a las variables del problema original para resolverlo en un conjunto

reducido de variables. Luego, en el SP se busca evaluar nuevas variables con sus respectivos parámetros a agregar en la función objetivo del problema maestro. Este subproblema guarda relación con el problema dual y su solución determina si se acepta o no la nueva variable. Este proceso se hace iterativamente hasta que se alcance una cierta calidad en la solución o que suceda algún otro criterio de culminación. El método nunca logró ser competitivo como método exacto, salvo para algunos problemas específicos como se muestra en Lübbecke (2011) y suele estar relacionado con el método de DW debido a que presenta sinergia al usarse en conjunto.

A partir de estos métodos de descomposición se han desarrollado nuevas heurísticas al cambiar partes del procedimiento. Lo que caracteriza a dichas heurísticas es tomar el cuerpo de algún método anteriormente descrito, aplicarle alguna heurística para facilitar su cálculo o cortar la ejecución de alguna forma y quedarse con un resultado intermedio.

3. Clasificación de Heurísticas para Problemas MILP

Para una mejor comprensión de las diferentes heurísticas, se propone un método de clasificación de tres niveles en donde se definen según su ámbito de ejecución, propósito y método utilizado. Cada una cuenta con 2, 2 y 7 subcategorías respectivamente obteniendo un total de 28 categorías. A continuación se desglosan las mismas. Según el ámbito de ejecución, las heurísticas pueden estar planteadas para un problema genérico o para un problema específico. Las heurísticas genéricas se pueden utilizar en cualquier problema MILP, mientras que las heurísticas específicas están planteadas para problemas particulares. Este trabajo presenta interés en heurísticas genéricas, en las que no se aprovecha información relativa a la estructura y características específicas de un problema, sino que sirven para cualquier problema MILP planteado. En cuanto al propósito de las heurísticas, estas pueden clasificarse en dos grandes grupos. Por un lado, se encuentran las heurísticas de solución inicial, que tienen como finalidad generar una solución factible de manera rápida para un problema dado. Por otro lado, se encuentran las heurísticas de mejora, las cuales buscan a partir de una solución inicial obtener soluciones mejoradas de un problema dado. Estas categorías logran agrupar los dos principales focos que puede tener una heurística, la optimalidad y la factibilidad.

El último nivel de clasificación se basa en el método utilizado por la heurística. Las diferentes heurísticas propuestas por los autores utilizan herramientas cuya forma de utilización varía. Observando qué herramientas se utilizan y la forma, se permite crear una clasificación por método. Si bien una heurística puede utilizar varias herramientas en el cuerpo del algoritmo, generalmente predomina un tipo de estas. A partir de una revisión de la literatura se proponen 7 métodos para la clasificación de heurísticas: heurísticas de aprendizaje automático, de descomposición, *relax*, *rounding*, estocásticas, *neighborhood* y *fixing*.

Las heurísticas de aprendizaje automático utilizan información de una base de datos para poder definir, ajustar y fijar variables. Utilizan información brindada por algún método o información de problemas relacionados como base. Estas heurísticas han tomado mayor relevancia en los últimos años con el aumento de prácticas relacionadas con el *machine learning* y la inteligencia artificial. Un método basado en descomposición divide al problema original en otros subproblemas más sencillos de resolver, los cuales van a permitir hallar una solución al problema original. Las heurísticas basadas en *relax* hacen uso de la herramienta de relajar el dominio de las variables para poder aplicar algoritmos basados en variables continuas. Como ya se observó, los algoritmos para problemas con variables continuas permiten la obtención de la solución óptima de forma más sencilla y estas heurísticas tienen diversas formas de volver a instalar la restricción de integralidad. Las heurísticas de *rounding* se basan en redondear determinados valores a medida que se resuelve el problema propuesto. Se obtiene una solución del problema relajado y luego se redondea a través de algún método, la principal limitante es la obtención de la factibilidad. Las heurísticas estocásticas son técnicas de resolución de problemas que utilizan elementos de aleatoriedad o procesos probabilísticos para encontrar soluciones, razón por lo que no son estudiadas al detalle en este documento. Las heurísticas basadas en *neighborhood* exploran una región factible alrededor de una solución base que se irá actualizando, acotando así la región y acelerando la búsqueda. Por último, las heurísticas basadas en *fixing***,** fijación en español y como el nombre lo indica, van fijando variables del problema utilizando diferentes métodos. Esto genera un menor número de variables a optimizar, disminuyendo el tamaño del problema y así facilitando la resolución. En la Figura 2 se presentan de forma gráfica las categorías planteadas.

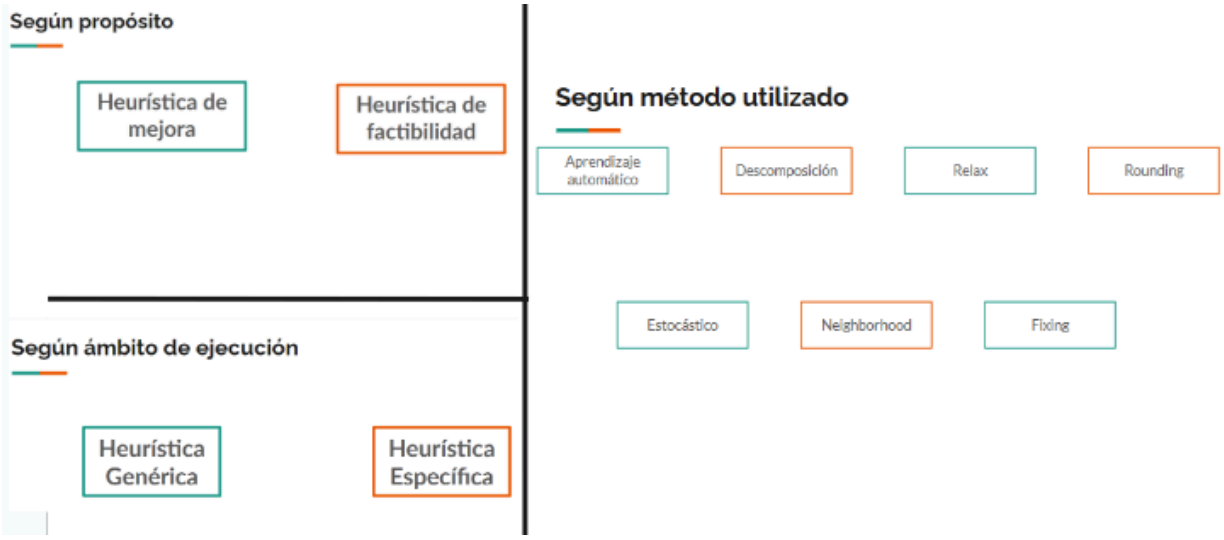

Figura 2- Clasificación propuesta según propósito, ámbito de ejecución y método utilizado de elaboración propia.

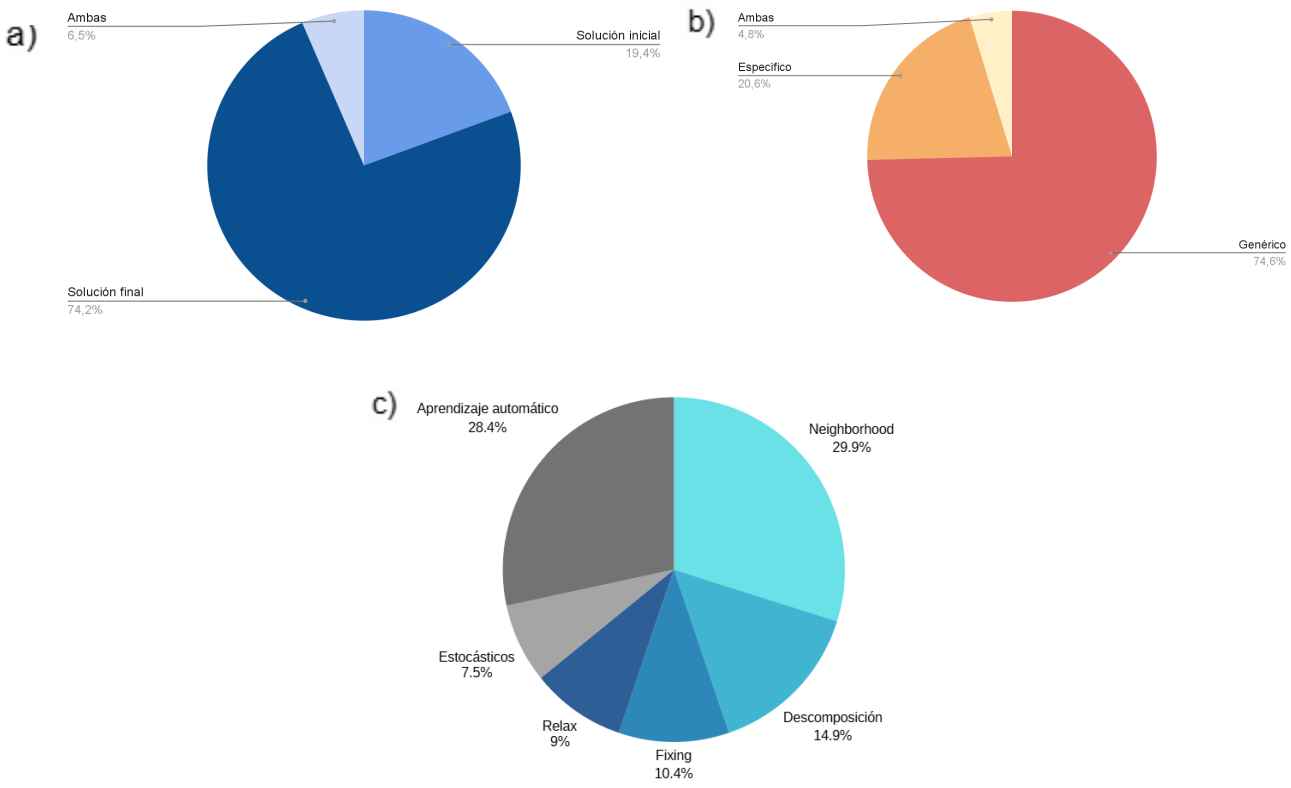

Figura 3: a) Distribución de los artículos citados según propósito. b) Distribución de los artículos según ámbito de ejecución. c) Distribución de los artículos citados según el método utilizado

A partir de la búsqueda de literatura, y teniendo en cuenta el sistema de clasificación propuesto, se organizó la bibliografía de forma que todo documento quede identificado según los tres criterios de clasificación. Si bien muchos documentos plantean la implementación de heurísticas que incluyen distintos métodos, se clasificaron según el que se considera como predominante. Se observa que la mayoría del ámbito de ejecución de los documentos son del tipo genéricos y su propósito es mayoritariamente la obtención de soluciones finales. Los resultados obtenidos se pueden observar en la Figura 3. La mayor cantidad de documentos presentan metodologías de *neighborhood* según método utilizado, seguida por aprendizaje automático, *descomposición*, *fixing*, *relax* y finalmente, estocásticos. Los métodos estocásticos, si bien aparecen en última posición, esto se debe principalmente a que no fueron el enfoque de la búsqueda de literatura.

4. Revisión de la literatura

En esta sección se presenta la revisión de literatura sobre la aplicación de distintas heurísticas en problemas de optimización modelados como MILP. Esta sección estará dividida por la agrupación presentada en la clasificación de las heurísticas, más específicamente según el método utilizado. Además, se especificará el propósito para el que es aplicada cada heurística (solución inicial y de mejora) así como el ámbito de ejecución para cada método propuesto (general o específico). Teniendo esto en cuenta, la presente sección se divide en 7 subsecciones donde en la Sección 4.1 se tratan las heurísticas *neighborhood*, en la Sección 4.2 se exponen diversas propuestas de heurísticas de descomposición. En la Sección 4.3 las heurísticas basadas en *fixing* y en la Sección 4.4 heurísticas basadas en métodos de *relax.* En la Sección 4.5 se presentan las heurísticas basadas en métodos estocásticos, la Sección 4.6 trata de las heurísticas de aprendizaje automático y se culmina con métodos de rounding en la Sección 4.7. Vale destacar que esta clasificación es utilizada en la literatura e invitamos al lector a observar artículos como Boschetti *et al.*, (2023) que son una introducción en el tema y enfocan la clasificación de las heurísticas desde otros puntos de vista.

4.1 *Neighborhood*

El primer desafío que se puede afrontar en la resolución de un problema es encontrar una solución factible inicial. Con el fin de encontrarla se pueden aplicar distintos métodos heurísticos para reducir la complejidad computacional para el SO y así aumentar las probabilidades de encontrar una solución factible en tiempos razonables. Además, estos métodos pueden resultar en un buen límite superior o inferior del problema (dependiendo de la función objetivo) que permitan podar una mayor cantidad de nodos en el posterior método *B&B* aplicado por el SO. Para hallar una solución inicial factible una de las heurísticas es FP que fue introducida en la Sección 2.4, FP ha sido implementada y se han realizado varios estudios de la misma buscando mejoras en la performance del algoritmo como en Berthold *et al.*, (2018). El método FP se considera como uno de *neighborhood* debido a que alrededor del punto encontrado se busca un punto que minimice la distancia a este y que se encuentre dentro de las restricciones del problema relajado.

4.1.2 *Neighborhood* de solución inicial

Una de las principales críticas al método FP se menciona en Fischetti *et al.*, (2005) donde se demuestra que dado que FP busca la obtención de una solución inicial que sólo está relacionada con la solución relajada del problema original, las soluciones obtenidas suelen ser de baja calidad. Para resolver este problema se propone, una vez obtenida la solución factible a través de FP, agregar al MILP una restricción de límite inferior o superior para la solución hallada en anteriores iteraciones para repetir el proceso de FP una cantidad limitada de veces y así cada vez encontrar una mejor solución inicial que la anterior. En Frangioni *et al.*, (2012) se vuelve a resaltar este problema pero con un enfoque distintos y se propone obtener mejores resultados a partir de un cambio en la función distancia (Δ) de FP. Para ello, primero realizan una analogía entre la función distancia del FP clásico y un paso del método de Frank-Wolfe aplicado a la minimización de una función no diferenciable y cóncava. La analogía viene de la

redefinición de la función distancia como $\Delta(x; x^*) = \sum_{j: x_j^* < 1/2} w(x_j^*) x_j - \sum_{j: x_j^* \geq 1/2} w(x_j^*) x_j$, obteniendo la función distancia clásica utilizando $w(x_j) = |\varphi'(x_j)|$, siendo $\varphi(x_j) = mín\{x_j, 1 - x_j\}$. A partir de este punto se proponen buscar la función $\varphi(x_j)$ con mejores resultados experimentales, concluyendo que esta es la definida como $\varphi(x_j) = mín\{\frac{1}{1+e^{x_j}}, \frac{1}{1+e^{x_j - 1}}\}$. Si bien argumentan mejores resultados, exponen que la función distancia empuja las variables a los enteros que parecen más prometedores sin tener en cuenta las restricciones activas cerca de la solución relajada. Es por ello que proponen otro enfoque para modificar la función de distancia, basados en la idea de que las mejores soluciones estarán cerca de los bordes de la región factible. Por lo tanto, la búsqueda se basa en la minimización de una nueva función distancia mín $\{ \phi(x, s): s = Ax + b, s \geq 0, x \in [0, 1]^n \}$ donde el peso de decisión lo tendrán las variables activas en la restricciones pues son las que conforman el borde.

Por otro lado, en Achterberg & Berthold, T. (2007) se propone una variación en la búsqueda del punto de la segunda trayectoria en FP al agregar en la función objetivo un término estrictamente relacionado con la calidad de la solución del MILP original en vez de con las restricciones. En Mexi *et al.*, (2023), abordan este problema a través de un enfoque complementario a FP con funcionalidad para las variables discretas no binarias. Primero separan FP en 3 etapas siendo la primera la ejecución de FP para encontrar una solución factible relajando las variables discretas no binarias. La etapa 2 refiere a la ejecución de FP partiendo de la solución encontrada en la etapa 1 sin relajar variables. Si FP no logró encontrar una solución factible en los pasos anteriores, se asume que se está "cerca" de una solución, y se activa una búsqueda de proximidad en la etapa 3 tomando el mejor vector entero encontrado hasta el momento en términos de proximidad a la factibilidad y modificando la función distancia con la presentada en Achterberg & Berthold, T. (2007).

Es en este punto donde el documento propone una modificación en la etapa 3. En lugar de seguir con la búsqueda originalmente planteada, se sugiere una nueva estrategia: a partir de la solución relajada del MILP original, se fijan las variables enteras y se plantea la resolución de un subproblema MILP. En este subproblema, la función objetivo se mantiene sin cambios, pero las variables se restringen a tomar valores cercanos a los obtenidos en la solución de la relajación. Este método está basado en el método *Relaxation Enforced Neighborhood Search* (RENS). Además, como método novedoso en el artículo citado proponen restringir las variables en la etapa 3 no por los valores discretos más cercanos a la solución relajada sino con información extraída durante la ejecución de FP, para así tratar dificultades que se pueden encontrar en la ejecución de RENS como la infactibilidad por restricciones duras. Por lo tanto, durante la ejecución de FP se guardan los valores intermedios máximos y mínimos de cada variable luego de su redondeo y son estos valores a través de los que se restringe el sub-MILP. A este método se le denomina *Multi-Reference* RENS. Vale destacar que, en general, RENS es utilizado para encontrar soluciones finales y no tanto así para una solución inicial como en este caso.

Por otro lado, se identifica como un problema recurrente en la heurística FP el fenómeno de *cycling*, el cual radica en consecutivas soluciones repetidas en el paso de redondeo y por lo tanto, a partir de allí se genera un ciclo sin salida. Para tratar este problema, los métodos

suelen modificar la solución a través del método de fijación alterando el número de alguna/s de sus entradas y así escapar del ciclo. También es muy usual la utilización de métodos estocásticos. Cabe destacar que, si bien incrementar la perturbación puede ayudar a escapar de ciclos, también puede alejar la solución del espacio factible original, lo que podría afectar negativamente su calidad. Por este motivo, suele ser beneficioso complementar estas perturbaciones con métodos de mejora que busquen recuperar o incluso mejorar la calidad de la solución obtenida.

En Baena & Castro (2023), se propone como solución al *cycling* trazar una recta entre la solución relajada obtenida antes del *cycling* (que estará al borde de la región factible relajada) y un centro (se proponen el centro analítico y el centro de Chebyshev). Sobre esa recta se eligen una cantidad finita de puntos (serán interiores a la región factible aplicando relajación a sus variables) y se ejecuta el paso de redondeo, esperando que algún punto sea factible al MILP original o al menos caiga dentro de la región factible aplicando relajación a las variables para seguir FP desde el mismo. Si más de un punto cae dentro de la región se elegirá el más cercano al punto de la solución, con variables a las que se apliquen relajación, inicialmente priorizando la calidad. Ya que el centro analítico es definido a través de las restricciones que limitan la región factible, las restricciones redundantes desviarán el mismo del verdadero centro analítico. Como resolución para este problema se aconseja el uso del centro de Chebyshev que geométricamente es el centro de la bola más grande que puede ser construida dentro del poliedro que define la región factible. El método logra mejores resultados que otros métodos de cycling como algunos estocásticos y de *rounding*.

Vistos estos problemas, en Yarahmadi *et al.*, (2023), enfocados en problemas de soluciones binarias, se propone optar por un nuevo modelo de resolución que no es guiado por métodos de redondeo. Por el contrario, se basan en encontrar una solución inicial del MILP original a través de la resolución de un *Feasible Finder Model* (FFM) basado en *ratio programming*, que se refiere a una técnica o enfoque de programación en el que se busca maximizar o minimizar una relación (*ratio*) entre dos funciones o términos. El modelo está relacionado con el MILP original a través de la definición de la misma región factible y añadir una restricción en relación a un valor límite (EL) para el valor de la función objetivo, controlando así la calidad de la solución inicial encontrada. Este nuevo modelo basado en *ratios* se convertirá en problemas lineales de resolución (mucho más eficientes) gracias a la adaptación del *Modified Newton Algorithm* (MNA) para la resolución del FFM. Una vez obtenida la solución inicial se ejecutará nuevamente el algoritmo con el valor EL actualizado en función de esta última. Los resultados computacionales, en comparación con la ejecución en CPLEX del MILP, mostraron que si bien el método propuesto demora más en encontrar la solución, lo hace en mayor cantidad de oportunidades y con una calidad mayor.

Otra heurística que busca la obtención de una solución inicial y que toma ideas de neighborhood es la vista en Balas *et al.*, (2001), que se enfoca en problemas exclusivamente binarios aprovechando la correspondencia uno a uno entre puntos binarios representados en $\mathbb{R}^n$ y las facetas de un octaedro de *n* dimensiones. El vecindario queda restringido por un octaedro de la dimensión correspondiente que representa el espacio de soluciones factibles del problema binario. Tomando el punto que se obtiene de la solución del problema relajado y una dirección establecida, se determina una semirrecta. Las intersecciones entre el octaedro y esa semirrecta definen soluciones factibles del problema original. Es un método que logra un resultado positivo en una amplia gama de problemas.

4.1.2 Neighborhood de solución final

Tras haber discutido las metodologías para la obtención de soluciones iniciales, el siguiente desafío al aplicar las metodologías *neighborhood* es hallar soluciones mejoradas para el MILP. Las heurísticas revisadas a continuación tratan este problema de diferentes formas aunque todas coinciden en la idea de reducir la región factible y encontrar una solución de alta calidad en tiempos reducidos. Vale recordar que no se va a garantizar que la solución obtenida sea la solución óptima del problema.

Large Neighborhood Search (LNS) es un método muy utilizado y se basa en dos operaciones claves, destruir y reparar. Esta es una heurística de mejora basada en la selección de un vecindario en el cual sea posible mejorar las soluciones halladas. Por tanto, se busca un vecindario que, al resolver el problema MILP, en este se pueda encontrar una mejor solución a la previamente obtenida. Dada la mejor solución factible encontrada hasta el momento, la heurística de destrucción se basa en fijar un conjunto de variables y optimizar un subproblema con las restantes variables. El número de iteraciones del procedimiento va a estar directamente asociado al tiempo límite que se digite. El uso de esta heurística genera, en muchos casos, mejoras de la calidad de la solución otorgada por los SO, aún basándose en una elección trivial de las variables a fijar.

Local Branching (LB) es un método desarrollado en Huang *et al.*, (2023) y se basa en LNS. El objetivo de la heurística es mejorar la solución encontrada en tiempos de cómputo acotados al explorar una región limitada. Para ello se acota el espacio factible alrededor de la solución previamente obtenida y se encuentra un conjunto de variables a destruir. En esta región potencialmente se encontrarán mejores soluciones en menores tiempos de cómputo que resolviendo el problema completo. Para definir la vecindad se crean nuevas restricciones que fijan la cantidad de variables que pueden diferir de la mejor solución encontrada hasta el momento. En particular, en esta heurística, el vecindario estará definido por las potenciales soluciones que sólo difieren en una cantidad acotada de entradas o variables. Partiendo de la solución factible inicial, iterativamente un conjunto de variables serán seleccionadas para ser destruidas.

Así como es aplicable en este tipo de búsqueda la heurística RENS, la cual ya fue definida, otro método parecido es el de *Relaxation induced neighborhood search* (RINS) enfocado en las variables discretas, no binarias. En este último, así como en RENS, luego de relajar el problema se fijan las variables que resultaron ser discretas y luego se opta por ejecutar *B&B* con el sub-MILP, resultando una solución incumbente donde nuevamente se realizará una fijación con las variables que coincidan con la solución relajada del sub-MILP. Este proceso se repite iterativamente disminuyendo la región de búsqueda en cada ejecución B&B hasta haber completado la misma o por otro criterio de parada. En este último método se basa Zhang & Nicholson (2019) llamado *Objective Scaling Ensemble Approach* (OSEA) que ataca los problemas desde el punto de partida que al menos el 95% de las variables tomarán un valor nulo en la solución óptima. Este punto de partida es apropiado para los problemas de selección de conjunto (*Set Covering*) y problemas de cobertura, problemas de ubicación (*Facility Location*), entre otros. El método se basa en fijar las variables para que tomen valor nulo, lo que es análogo a eliminar esas variables de la formulación. Para identificar qué variables fijar a cero existen varias alternativas, pero todas comparten la idea de observar conjuntos de posibles soluciones y observar patrones, evaluando las que generalmente son cero y fijar estas para posteriormente acelerar el método de B&B con un procedimiento similar al de RINS.

Un posible problema con la idea presentada es que las soluciones factibles obtenidas para la posterior fijación de variables sean de mala calidad. En este caso, existe un alto riesgo de fijar variables que perjudican la función objetivo. Para mitigar este riesgo lo que se propone es evaluar el mismo problema con variaciones en los coeficientes de la función objetivo. Haciendo esto, en Zhang & Nicholson (2019) argumentan que si una variable toma cero en la solución, aún cuando el costo asociado (coeficiente de la función objetivo) se redujo a fracciones del costo original, entonces dicha variable probablemente no sea útil en el problema. Además, proponen que el costo absoluto asociado a una variable no es tan importante como el costo relativo respecto a otras. En ello basan el método iterativo por lo que se fundamenta que sólo se están eliminando las variables que no aportan a la optimalidad de la solución.

En Ma *et al.*, (2020), y basado en la *Hamming Distance* (HD) del método LB, se propone dirigir la búsqueda del *B&B* incorporado en los SO a través de la *Relaxation Distance* (RD), inspirada en HD, con el objetivo de reducir la región factible. En específico, la diferencia con la HD es el agregado de coeficientes de peso, haciendo pesar más la distancia medida en una entrada de la solución donde la tendencia de la solución del problema relajado es fuerte (por ejemplo una entrada cuya solución aplicando relajación es *0,9*, se dice que tiende fuertemente a *1*) que donde no hay una tendencia de parte de la solución relajada (por ejemplo cuando vale *0,5* la entrada, ya que no tiene ni a *1* ni a *0*). Luego, simplemente se acota el MILP bajo estas restricciones por lo que se resuelve un sub-MILP de un tamaño considerablemente menor al MILP.

En Ghirardi & Amerio (2019) se propone un método heurístico de resolución para problemas complejos de creación de cronograma de producción, cuando los tiempos y costos de *setup* son significativos. Se propone *Variable Perturbation Local Search* (VPLS) que, en base a la resolución de subproblemas, actualizan la solución hasta que el número de iteraciones sin encontrar una solución mejorada llegue a un máximo preestablecido o el tiempo máximo de ejecución se cumpla. En cada iteración del algoritmo se define aleatoriamente *i* productos libres y *j* máquinas libres, permitiendo que la variable de decisión de producir en la máquina *j* el producto *i* (cualquiera sea el período considerado) quede liberado a la modificación y se resuelva el MILP. Este método es aplicable a diversos contextos de asignación y planificación.

Otro método propuesto para acelerar el hallazgo de la solución final es el expuesto en Beasley (2024), donde dada una solución inicial, se define un vecindario al igual que en LB a través de la HD y se restringe el espacio de búsqueda a las posibles soluciones que mejoren el valor objetivo del MILP original. Por lo tanto, no sólo se reduce la región factible por la HD sino también por las que den un mejor valor objetivo. Este método es una variación de LB, enfocado en problemas de cobertura de conjuntos no unicosto ($c_i \neq 1$ en la formulación MILP) donde se han observado mejores resultados para problemas específicos. Dado que las restricciones son aún más fuertes que en LB, es fácil entender que es más propenso a encontrar infactibilidades y para solucionar esto es que se varía el parámetro *k,* tamaño de la vecindad de la HD, que es lo que permite a la heurística aumentar su espacio de búsqueda.

En Darwiche *et al.*, (2018) se plantea una posible mejora para el método LB con este último concepto, en el documento proponen métodos de intensificación y diversificación del espacio de búsqueda por la variación del parámetro *k*. Primero se plantea la búsqueda en el vecindario determinado por LB, pero si en el tiempo especificado no se logra mejorar la solución encontrada, se plantea la hipótesis de que es porque el vecindario es demasiado grande. Por ello se plantea la disminución del parámetro *k*, para que luego, si no se encuentran mejores

soluciones, se adjudica a la existencia de un óptimo local. Para salir de este se plantea la resolución de un MILP con la restricción contraria a la de LB, siendo la diferencia entre las soluciones mayor a un parámetro *k´.*

Otro tipo de heurística de *neighborhood* es la llamada *Tabu Search*, desarrollada en Koguma (2024). En el documento se definen 3 formas en las que se puede definir una región de *neighborhood*: *flip*, *shift* y *jump*. Un vecindario definido por *flip* es para variables binarias y comprende movimientos que invierten el valor de un único componente $\bar{x}$ (este es el aplicado para la HD). Los vecindarios definidos por *shift* y *jump* son para variables discretas no binarias, el vecindario creado por *shift* comprende movimientos que incrementan o decrecen el valor de un único componente $\bar{x}$ dentro de su rango, mientras que el vecindario de *jump* comprende movimientos que alteran el valor de un único componente $\bar{x}$ en un valor igual a la mitad del margen hacia los límites definidos por la región factible. Una vez definida la forma a utilizar para definir el vecindario, se procederá a la evaluación de las posibles soluciones, elección de la mejor y nueva definición del vecindario. Lo novedoso del método radica en la utilización de una lista tabú en la que se guarda soluciones o movimientos que no se deben considerar durante un número determinado de iteraciones, evitando así repetir espacios de búsqueda ya explorados. En Koguma (2024) se puede profundizar en los conceptos de la heurística, además de observar como es perfectamente aplicada en conjunto con otras heurísticas mejorando el comportamiento de la búsqueda y así minimizando el tiempo requerido.

En Bansal & Uzsoy (2021) se desarrolla otra heurística de *neighborhood* llamada *Constraint violation reduction search* (CVRS) y parte de la idea de permitir una violación de las restricciones (a través de una variable *s* tal que $Ax \leq b + s$) para relajar el espacio de búsqueda a través de una reformulación del MILP con la utilización de variables auxiliares. La suma de estas variables representan cuántas restricciones son violadas y la función objetivo del nuevo MILP será minimizar la suma de estas variables auxiliares. Esta se evaluará con la restricción de que la nueva solución encontrada sea mejor que la solución actual en una cantidad controlada por un parámetro Θ. Por lo tanto, si esta suma es nula significa que se ha encontrado una mejor solución y se actualiza. Si esta suma es positiva hay dos opciones: no se encuentra mejor solución y hay que redefinir algún parámetro o hay una mejor solución pero violando restricciones. Si el último fuera el caso, entonces se procede a realizar una búsqueda a través de algún criterio de *neighborhood* alrededor de la solución encontrada, esperando que el valor objetivo de la nueva solución sea nulo. Si el criterio de búsqueda no fue efectivo se propone optar por la reducción del parámetro Θ y la repetición del proceso. Si una mejor solución es encontrada se actualiza a esta solución y se repite el proceso aplicando una modificación del parámetro Θ. A modo de dar un cierre al procedimiento CVRS pero también como ejemplos aplicables en general, se listan a continuación distintos criterios de terminación de los métodos heurísticos, en particular aplicables a heurísticas de *neighborhood:*

- Límite de tiempo: finaliza la búsqueda tras un tiempo predefinido.
- Número de iteraciones sin mejora: termina si no se observa mejora en el valor de la función objetivo en un número específico de iteraciones.
- Tolerancia de convergencia: se detiene si la mejora entre iteraciones es menor a un umbral pequeño, lo que indica la posibilidad de que la solución está cerca de ser óptima.

En base al contenido desarrollado en esta sección, se recomienda que en la utilización de

alguna de estas heurísticas sea aplicado algún criterio para escapar de óptimos locales antes del cumplimiento del criterio de terminación. Por otro lado invitamos al lector a profundizar en estudios para elegir correctamente la heurística *neighborhood* para cada problema y sus limitaciones a través de Hendel (2021).

4.2 Heurísticas basadas en descomposición

Se comienza el análisis en Mo *et al*., (2019) donde se resuelve un problema específico con el objetivo de cargar sensores inalámbricos a través de cargadores móviles generando una red de sensores inalámbricos. Se plantea un problema en el que se busca minimizar el consumo de energía de los cargadores, garantizando que nunca se queden descargados, lo que da lugar a una formulación compleja y de difícil resolución. El método propuesto para su resolución se inspira en la descomposición de Benders. Se va a dividir el problema en un MP que representa la planificación de los cargadores móviles y un SP que representa el tiempo para mover los cargadores y los tiempos de carga de los sensores. Se va resolviendo ambos problemas iterativamente y se van obteniendo cotas para el problema en general. En cada una de las iteraciones se observa la posibilidad de incluir una nueva restricción al problema haciendo que el método converja a una solución. En el documento se estudia la convergencia y se logra mejores resultados que en los métodos propuesto por la literatura.

Otro método que utiliza la DB es Farkiani *et al*., (2019) para la resolución de un problema específico, fácilmente extendible a un problema general. El paper propone dividir el problema original en dos subproblemas, uno que englobe las variables discretas (MP) y otro las continuas (SP). Para la resolución de este problema se propone utilizar FP e información relacionada al problema dual valiéndose del teorema de holgura, que de otra manera incluiría la solución de un problema discreto en cada iteración. El método comienza buscando una solución inicial al problema original para luego fijar alguna de las entradas de esta solución en la resolución del problema dual del MP. Se fija esta solución en el SP y de la solución obtenida se agrega nuevas restricciones al MP, como se propone en DB. Vale la pena aclarar que en todo momento se evalúa la solución obtenida buscando mejoras en la función objetivo original y en caso de existir infactibilidades plantea varios mecanismos para retroceder en restricciones nuevas. El método propuesto logra obtener resultados 30% mejores que un B&B y lo logra hacer en tiempos 10 veces menores.

DB también se utiliza en Mo *et al*., (2019) donde los autores buscan determinar la solución de un problema con una cierta arquitectura llamada *Asymmetric multicore processors* la cual permite la resolución de una larga serie de problemas. En este paper se demuestra que la metodología de descomposición DB logra converger a la solución del problema original. El problema que resulta de esta metodología es que su uso no es viable para la resolución de problemas con alta cantidad de variables y restricciones debido a que el MP sigue siendo un problema con variables discretas o binarias y que su cantidad de restricciones aumenta con la cantidad de iteraciones debido a las nuevas restricciones surgidas por optimalidad y por violación. Para poder pasar estas dificultades se propone una metodología heurística que plantea el método de FP para hallar una solución al MP y se culmina cuando el SP obtiene una solución acotada. De esta forma, se logra obtener una solución al problema original en un tiempo despreciable y perdiendo solamente un 29,8% de la calidad de la solución en comparación al método sin heurística.

En Sadykov *et al.*, (2017) se utiliza la descomposición de DW en conjunto con *Branch & Price* (B&P). El documento busca brindar solución a problemas genéricos en donde se explota la reformulación propuesta por DW en conjunto con heurísticas de *diving*. Para la resolución del MP se proponen métodos de *diving* en el arbol del B&P donde se evalúan propuestas para la elección de ramas de búsqueda. El primer método de selección, propone un método de *rounding* para la selección de qué rama tomar, seleccionando un nodo dependiendo la forma de redondeo de solución del problema MP aplicando relajación. Luego, se propone este mismo método y se agrega la posibilidad de volver un paso atrás con el objetivo de explorar otro nodo diferente. Para asegurarse de elegir un nodo diferente se utiliza una lista tabú en la que se indica qué ramas ya fueron exploradas. Otra forma propuesta para la elección de la ramificación es elegir según cual rama va a tener un mayor impacto en el *dual bound* pero como su cálculo es computacionalmente costoso se propone el cálculo aproximado a través de heurísticas tomando algunas ramas para evaluar por medio de redondeo. De esta forma, se resuelve el problema relajado y luego se aplica redondeo a la solución y así calcular su *dual bound* y elegir el que menos empeore el resultado. Otro método propuesto es la posibilidad explorar a fondo un nodo específico del problema con la posibilidad de volver a él luego de profundizar en una rama específica. Otro método propuesto consiste en limitar un conjunto de variables del MP que fueron generadas a partir de la solución de algún nodo. También se puede aplicar LB o FP para la obtención de resultados manteniendo calidad y lograrlo de forma rápida. Todos estos métodos propuestos logran un buen desempeño en la resolución del problema en comparación con otras heurísticas de la literatura. Se destaca el funcionamiento de la heurística que permite volver pasos atrás según se vaya desarrollando.

Para poder considerar problemas con incertidumbre se propone un método llamado t*wo-stage robust optimization* (TSRO) el cual en Zhang *et al.*, (2024) se busca determinar a partir de CG. Se define una variación de CG llamada *column & constraint generation* (nCCG) que consiste en dos loops iterativos en donde uno busca minimizar un MP en el que se considera el peor de los casos (como existe incertidumbre se considera el peor caso) y otro en donde el SP busca maximizar y minimizar un subproblema para obtener el peor de los casos. El método consiste en la implementación de FP en el SP y así mejorar el tiempo de resolución. Con esto en cuenta se genera el algoritmo que comienza con la resolución del MP para agregar su solución al SP. Se resuelve el problema dual del SP y se utiliza FP para comparar factibilidad. Dependiendo del resultado se actualiza el MP con nuevas restricciones y vuelve a comenzar el ciclo. Se termina cuando la diferencia entre la cota superior e inferior sea suficiente. Luego, realizando ensayos computacionales se llega a que el método nCCG es más eficiente computacionalmente que otros métodos y logra mantener la misma robustez en la solución.

Una heurística que aplica la relajación Lagrangiana es Luteberget & Sartor (2023). A través de heurísticas estocásticas se busca guiar la relajación Lagrangiana planteando una heurística llamada *Feasibility Jump* (FJ), un método que produce de manera rápida una solución inicial para un problema MILP. No recurre al cálculo del problema relajado, por lo que facilita la resolución en instancias que presentan un gran tamaño. El método consiste en seleccionar iterativamente variables y minimizar una suma ponderada de restricciones. El método propuesto toma la idea de la relajación Lagrangiana y minimiza una función objetivo similar a la función Lagrangiana la cual va a ser evaluada en vecindarios en donde solo se cambia el valor de una variable a la vez. Lo que se va a hacer es fijar un conjunto de variables y obliga a modificar una sola variable a la vez para definir hasta donde se puede modificar sin romper alguna restricción y también se define el “salto” que puede dar la variable. De esta forma los multiplicadores de lagrange pasan a ser un peso en cada restricción forzando la búsqueda en los bordes de la región factible.

En Boutarfa *et al.*, (2024) se estudia el problema de una cadena de manufactura que busca incorporar materia prima surgida de productos fallados o desechados luego de fabricado. Para su resolución se propone una descomposición que no tiene en cuenta la disposición del problema específico. Se crean tres subproblemas en donde se fijan diferentes conjunto de variables del sistema generando tres subproblemas con una menor cantidad de variables y, por ende, con resolución más sencilla cuyo tamaño de variables fijadas cambia en el correr del proceso iterativo. Se plantea una metodología de trabajo tipo caja negra en donde la resolución de estos subproblemas queda a cargo de un paquete de optimización. El primer conjunto de variables se fija según iteraciones anteriores. El segundo, toma un valor entre *0* y *1* arbitrariamente. El tercer conjunto obliga a las variables ser *0* o *1*. Se obtienen varias soluciones y se busca quedar con la que de mejor resultado. Por otro lado, se propone otro método de descomposición el cual toma las variables binarias y las separa en dos subconjuntos. Un primer subconjunto el cual fija los valores a 0 o 1 y un segundo conjunto que se obtiene evaluando valores anteriores de soluciones. Se obtiene un conjunto de tamaños para cada subconjunto el cual logra mejores resultados en las pruebas computacionales. Además, ambas heurísticas logran una solución muy cercana a la óptima en escasos segundos.

En Chitsaz *et al.*, (2024) se centra en la resolución de problemas de planeacion de produccion y ruteo de la salida de los productos (*Production Routing Problem*), problemas donde la cantidad a producir es un dato y se centra en cómo llegan los insumos a la planta (*Inventory Routing Problem*) y problemas que integran la llegada de los insumos a planta y la planificación de la producción (*Assembly routing problem*). En este documento se propone aplicar una heurística de descomposición aprovechando la estructura específica compartida de estos tres tipos de problemas. No se utiliza ninguno de los métodos nombrados anteriormente sino que se proponen tres nuevos subproblemas que luego son resueltos utilizando alguna heurística.

Otro método que aprovecha la estructura específica del problema es Benavent *et al.*, (2023) en donde se busca resolver problema *periodic rural postman problem with irregular services*. El problema consiste en diseñar un conjunto de recorridos con un mínimo costo, logrando cumplir con el servicio y en cada período de tiempo. Puede aplicarse a limpieza de calles, recolección de residuos, mantenimiento de calles y vigilancia en carreteras. Para su resolución se utiliza una descomposición que va a dividir el problema en un problema de asignación de los *links* requeridos de un día determinado para los grupos de días. El otro problema consiste en la construcción de rutas para cada día en las que al menos un recorrido solicitado se hace. Para la resolución de cada problema se proponen métodos heurísticos como FP y heurísticas estocásticas. El rendimiento del algoritmo es muy bueno considerando el problema complejo obteniendo un porcentaje de diferencia menor al 10% en instancias cercanas a 20 minutos.

Otro método catalogado como descomposición es el desarrollado en Cereser *et al.*, (2022), el mismo se basa en el método simplex utilizado en diferentes subproblemas que se descomponen según variables básicas y no básicas. Este método define las *m* (cantidad de restricciones) variables básicas como las activas no nulas, y busca qué vértice de la región factible es el del mejor valor funcional. Luego, analiza el intercambio de una de las variables básicas por una no básica y evalúa los valores funcionales en los nuevos vértices adyacentes, actualizando la solución encontrada si lo amerita. La recta de un vértice a otro está direccionada por las direcciones simplex, por lo tanto, el artículo plantea la relajación lineal de las variables para la aplicación del método simplex y recorrer el borde de la región factible a través de la recta dada por la dirección simplex en la búsqueda de una buena solución entera. Si esto no es factible buscará cerca del borde pero dentro de la región factible. Por lo tanto, se resolverán *n-m* sub problemas, siendo *n* la cantidad de variables en el problema y *n-m* la

cantidad de variables no básicas. A continuación se elige la solución de mejor valor funcional y se repite el proceso hasta que en los *n-m* subproblemas generados no se encuentre una solución mejor a la anterior.

4.3 Heurísticas basadas en Fixing

Las heurísticas de fijación son un conjunto de métodos que simplifican la obtención de una solución al fijar ciertas variables a valores específicos, reduciendo el tamaño del problema original y por ende, el tiempo de resolución. Estas heurísticas resultan valiosas para abordar problemas de gran escala o con estructuras complejas, ya que facilitan la búsqueda al concentrarse en subespacios dentro del espacio de soluciones. El concepto central de estas heurísticas es identificar un conjunto de variables que se consideran prometedoras para fijar y la forma para determinar a qué valor se fijan. Este enfoque puede mejorar la eficiencia computacional al transformar el problema original en un subproblema más pequeño o menos complejo que puede resolverse de manera más rápida. Una vez fijadas ciertas variables, las heurísticas de *fixing* permiten explorar combinaciones de las variables no fijadas, lo cual ayuda a encontrar soluciones factibles en menos tiempo que los métodos exactos convencionales.

Comenzando la revisión de literatura, en Joncour *et al*., (2023) se propone un método iterativo en el que se divide el espacio de variables en *K* subespacios, agrupando las variables según las restricciones a las que están asociadas. A estas variables se las denomina locales. Cada subespacio de variables locales genera un subproblema, que se enfoca exclusivamente en el conjunto de variables locales correspondientes a ese subespacio. En el caso general, no se divide en *K* conjuntos, pues existen las variables que están en varias restricciones a la vez. A este subconjunto de variables se denomina variables globales. El algoritmo propuesto resuelve, en cada iteración, el subproblema definido por el conjunto de variables *K* y las variables del conjunto global.

El método comienza resolviendo el subproblema definido por el primer subgrupo de variables locales. Para abordar los siguientes subproblemas se utilizan los valores fijados de los subconjuntos anteriores, lo que reduce significativamente la cantidad de variables activas facilitando así su resolución. Además, esta metodología ofrece flexibilidad en la resolución de cada subproblema, permitiendo utilizar tanto enfoques heurísticos como métodos exactos. Esto contribuye a una reducción adicional en los tiempos de computo, adaptándose a las necesidades específicas de cada subproblema. El método culmina cuando todas las variables locales estén fijadas y se resuelva el subproblema generado por las variables globales. Un problema que surge del método es la incapacidad de garantizar la factibilidad de la solución. Por este motivo se propone agregar una búsqueda a través del método de *Local Search* (LS) para evaluar nuevos resultados permitiendo retroceder en la fijación de las variables. Este proceso tiene un alto costo computacional pero aumenta significativamente la capacidad de brindar soluciones factibles al algoritmo. Además, un estudio computacional del método determina la capacidad de obtención de resultados aceptables en comparación con metodologías como FP o RINS. El algoritmo funciona especialmente bien en problemas con una gran cantidad de variables y restricciones.

En Dupin & Talbi (2018) se plantea un método específico para la minimización de costos en el contexto de brindar energía eléctrica cumpliendo con una demanda y bajo las restricciones para la generación de la misma. La heurística plantea la resolución del problema relajado y determina varias formas de fijar las variables. La combinación de un método de fijación con un

método de relajación permite la elaboración de diferentes métodos que se comparan mediante ensayos computacionales. De estos surge la relación de que al fijar más variables, mayor es el *gap* con respecto a la solución óptima pero la solución se obtiene en menores tiempos de cómputo. Se aclara que varias de las propuestas de fijación y de relajación no son genéricas sino que aprovechan estructura o propiedades específicas del problema. Las pruebas computacionales se hacen en el mismo problema y se obtienen resultados superiores que otras heurísticas de la literatura. Además, se llega a que las mejores estrategias son aquellas en las que se aprovechan las propiedades específicas del problema.

En Gamrath *et al.*, (2019) se utilizan estructuras globales de los problemas MILP en donde se fijan iterativamente variables discretas, proceso denominado propagación de variables. Se fija una variable y luego se va a propagar esta fijación a un conjunto de variables relacionadas con la variable fijada. Para poder propagar estas fijaciones se recurre al conocimiento brindado por la estructura del problema, donde el proceso de detección de la estructura se suele llamar fase de pre resolución y previo a este se suele hacer un proceso de pre procesamiento que busca remover redundancias del modelo, también conocido como presolución. Al utilizar la estructura global del problema, se va a poder predecir el efecto de las fijaciones en un conjunto de variables. Este proceso se llama fijar y propagar (*fix & propagate* en inglés) y las formas de propagación estudiadas en el documento son 3: *clique table*, *variable bound graph* y *variable blocks*. A continuación, se describe cada una de estas.

- Las estructuras *clique table* refieren a aquellas restricciones comprendidas por variables binarias en las que se restringe que a lo sumo una de estas variables puede valer *1*. Este tipo de restricciones son muy comunes en varios tipos de problemas y permiten extender la fijación una vez que se determine la no nulidad de alguna variable binaria que forme parte de la restricción. De esta forma, el resto de variables quedan automáticamente fijadas a *0.*

- Las estructuras de *variable bound graph* describen inecuaciones que contienen exactamente solo dos variables, permitiendo establecer relaciones llamadas relaciones de acotación de variables. Estas muestran la dependencia de una variable con otra guardando la información en una matriz para poder evaluar de forma dinámica los efectos de fijar alguna de las dos variables.

- Las estructuras globales de *variable locks* buscan determinar el número de restricciones a las que pertenece cada variable y se evalúa si se puede bloquear, aumentar o disminuir su valor. En caso de que una variable no esté restringida para una cierta dirección, significa que se puede modificar, sin recurrir a infactibilidades.

Basándose en estas ideas, el método propone fijar un conjunto de variables que luego serán propagadas utilizando la información obtenida del análisis estructural del problema. En todo momento se evalúa la factibilidad y, en caso de romperse, se da un paso atrás en la fijación para evitar demoras, esta opción se restringe a un máximo de 10 repeticiones. Luego de esta fase se evalúa el efecto del método observando la cantidad de variables que fueron fijadas. Se realiza un estudio computacional y se demuestra que las tres estructuras por separado brindan una mejora y tienen una tasa de éxito superior al 50%. Además, se determina que implementando los tres tipos de propagación juntos se logran aún mejores resultados, mostrando la sinergia que existe entre estas.

Utilizando las mismas estructuras globales, Salvagnin *et al.*, (2019) propone otro método de fijar y propagar en donde se profundiza la posibilidad de dar pasos hacia atrás en el método.

Plantea diferentes estrategias para la elección de qué variables fijar. Se propone un método estático, que jerarquiza las variables a fijar antes de resolver el problema. A su vez, se estudia la opción de jerarquización dinámica, la cual fija las variables mientras se está ejecutando el método. En el documento se profundizan varias estrategias estáticas que, en caso de encontrar infactibilidades, se recurre a un algoritmo que va a modificar variables siguiendo criterios *greedy*, *random* y *walk,* según el método propuesto en *WalkSAT.* El método consiste en modificar variables hasta que el problema logre ser factible modificando variables individualmente, con el objetivo de ajustar las restricciones violadas de manera que estas alcancen su valor límite. Al hacerlo, se consideran las estructuras globales del problema y se propaga el resto de variables.

En Zhang *et al*., (2018) se estudia la optimización del funcionamiento de generadores de energía geotérmicas teniendo en cuenta su mantenimiento. Se plantea un algoritmo basado en el método de *objective scaling ensemble approach* (OSEA) el cual funciona en dos etapas (Zhang & Nicholson, 2019). Basándose en experimentos con diferentes heurísticas, se observa que la solución óptima del problema original tiende a ser similar a la solución óptima de su versión relajada. Con esto en cuenta, se fija al valor *1* una variable del problema general, según el resultado del problema aplicando relajación. En la primera fase se resuelve una secuencia de subproblemas relajados del problema original, propuesto por Kim & Pardalos (1999), fijando variables las cuales son cercanas a los valores *0* o *1*. La segunda etapa consiste en fijar las variables obtenidas en la etapa anterior y resolver los subproblemas. Se propone que el método logra mejores resultados que el OSEA original.

4.4 Heurísticas basadas en Relax

Las heurísticas basadas en relajación ofrecen un enfoque práctico y directo que aprovecha la eficiencia computacional y la diversidad de los métodos de resolución de problemas LP. El concepto de relajación se refiere a permitir al dominio de variables discretas tomar valores continuos. Una vez que se amplía el dominio de las variables se dice que las variables están relajadas. Al permitir la expansión del dominio permite evadir las restricciones de integridad las cuales causan la no convexidad del problema impidiendo la utilización de algunos métodos exactos del estado del arte. Una limitación que presentan estos métodos es que para problemas con gran cantidad de variables y restricciones, aplicar relajación tampoco permite una mejora significativa en tiempo de cómputo en la resolución debido a que los problemas LP de gran tamaño tampoco poseen un algoritmo que permita su resolución de forma eficiente. Otro problema que surge directamente de este tipo de heurísticas es que no garantiza que la solución obtenida cumpla las restricciones de integridad que fueron quitadas, por lo que para solventar esto se suelen ver heurísticas de *fixing* que están acompañadas por otro tipo de heurística que permita la factibilidad de las variables en origen discretas.

En Joncour *et al*., (2023) se utiliza un método llamado Restringir y Fijar basado en otro método llamado Relajar y Fijar (*Relax & Fix* en inglés). Estos métodos permiten mitigar la complejidad de los problemas de gran escala al descomponerlos en varios subproblemas, según las variables involucradas. A cada subproblema se le aplica una relajación que facilita su resolución al reducir su tamaño. A su vez, se tiene en cuenta la posibilidad de no cumplir con la factibilidad por lo que se tiene en cuenta mecanismos para asegurar que las variables sean discretas. Otro método que mezcla fijación con relajación es Dupin & Talbi (2018), en donde se proponen diversas formas de relajar y se hace un análisis para determinar cuál es la que da mejor resultado. Entre las opciones de relajación se destaca un método que considera el

tiempo de cómputo de la resolución de la relajación para poder decidir la cantidad de variables a fijar. En el documento, el método determinó que los mejores resultados se obtienen por una relajación que se hace por partes, siguiendo un procedimiento similar al de RINS.

En Ma *et al.*, (2020) se combina la relajación con búsquedas locales en vecindarios para problemas binarios. La idea consiste en proponer un método *neighborhood* combinado con la relajación del dominio de algunas variables, creando una nueva forma de determinar el tamaño del vecindario mediante la *"relaxation distance"* (RD). En la RD se toma en cuenta un parámetro denominado como "peso" que busca representar la tendencia que sigue las entradas de la solución del problema relajado. Al trabajar con variables binarias las dos posibles tendencias de las variables es acercarse a *1* o *0* al resolver el problema relajado. Entonces el vecindario queda limitado según la RD máxima elegida entre la solución del problema relajado y la variable sin aplicar relajación. A su vez, este método propone un forma de relajación denominada *Relaxation Inducement* (RI) que introduce un nuevo término en la función objetivo haciendo que los valores de las variables binarias sigan la tendencia de la se hablo anteriormente. Por último, en Boutarfa *et al.*, (2024) se plantea una heurística de fijar *y* optimizar en donde se separa al problema en varios subproblemas y se aplica relajación en determinado conjunto de variables. Este método ya fue analizado y se puede consultar en la Sección 4.2.

En Celaya *et al.*, (2022) se combina un teorema exacto relacionado con *neighborhood* y el método de relajación. El teorema, propuesto por Cook *et al.*, (1986), busca dar una relación entre la solución del problema relajado y la solución del MILP. El teorema define un vecindario alrededor de la solución del problema relajado con un radio donde se tiene en cuenta la matriz de restricciones A de tamaño $m \times n$y la cantidad de variables del problema, en la ecuación (21) se puede observar la definición del vecindario propuesto. Lo que se busca demostrar en el documento es la posibilidad de reducir el radio de exploración del *neighborhood* para poder mejorar la búsqueda y luego realizar la búsqueda en un espacio de búsqueda más pequeño. De este teorema surge un algoritmo trivial para, obteniendo la solución del problema relajado, asegurar a su alrededor un radio de exploración en donde se encontrará la solución del problema original.

$$\left|\left|x^* - z^*\right|\right| \leq n\Delta \qquad (21)$$

En esta ecuación, x^* es la solución del problema relajado, z^* la solución del problema original, *n* la cantidad de variables y Δ es el subdeterminante de la matriz A con mayor magnitud.

4.5 Heurísticas basadas en métodos estocásticos

Las heurísticas estocásticas, en contraposición a los métodos deterministas, utilizan elementos aleatorios para explorar soluciones de manera eficaz al introducir componentes estocásticos. Estos generalmente son creados para escapar de óptimos locales y tener mayores probabilidades de encontrar una solución óptima, especialmente en espacios de búsqueda amplios y complicados (Riccardi *et al.*, 2021). Al tener un enfoque en la aleatoriedad, estos métodos distan de los anteriores vistos por lo que, en base al alcance del documento no son analizados extensivamente. De todas formas, se procede a explicar algunos métodos que resultaron interesantes de la búsqueda bibliográfica.

Se comienza por explicar un funcionamiento estocástico dentro del método FP. Para evitar bucles característicos del método (que la solución redondeada del problema sea igual al valor obtenido en la anterior iteración), en Fischetti *et al*., (2005) ya se contemplaba la idea de un cambio aleatorio en algunas variables. El objetivo es evitar repetir el proceso indefinidamente escapando de óptimos locales y comenzando una nueva búsqueda con el objetivo de encontrar el óptimo global. Esta variación conlleva un riesgo importante pues se debe considerar que no se debe alejar la solución lo suficiente como para perder el mínimo al que se buscaba converger. Se trata de un compromiso entre convergencia y la búsqueda de otras soluciones. Este mecanismo de perturbación tuvo diversos aportes a lo largo de los años (Berthold *et al*., 2018). En el caso binario se considera la posibilidad de cambiar un número aleatorio de entradas de la solución siguiendo alguna función estocástica. En casos de variables con dominio continuo, cambiar un número aleatorio de entradas permite la variación antes de violar alguna restricción. A su vez, se puede variar el factor aleatorio de forma tal que sea dinámico según la perturbaciones detectadas.

En Zaozerskaya (2021) se buscan resolver *Generalized assignment problem* (GAP) los cuales está demostrado que son generalmente NP-hard y tienen varias aplicaciones. En este documento se crea una heurística que busca encontrar soluciones aproximadas buscando de forma aleatoria para encontrar una solución factible inicial del problema relajado. Luego, se fijan parte de las variables binarias y se busca encontrar una solución aproximada utilizando un paquete de optimización. Si se encontraba una solución óptima, entonces se buscaba una solución similar a ella. Por último, se utiliza un procedimiento de mejora local para obtener una solución factible.

En Khalil *et al*., (2022) se introduce el concepto de *backdoors* que se define como un grupo de variables discretas a través de las que se logra resolver B&B ramificando únicamente estas. Para poder hallar este conjunto se propone la búsqueda en un árbol de Monte Carlo, método que guarda relación con la teoría de juegos. El método propuesto fija el largo del *backdoor* y comienza un método de selección estocástica de los diferentes candidatos. Luego, se determina las ramas a explorar a las cuales se les realiza una simulación aleatoria en donde se determina si la exploración dará un resultado positivo o negativo, definiendo qué nodos son buenos candidatos según cuales son más explorados. El método obtiene un mejor desempeño que otros de la literatura y además lo logra en menores tiempos.

En Boukhari & Hifi (2024) se busca resolver *Knapsack problems*, un conjunto de problemas aplicables a varios casos de la realidad. En este documento se utilizan variantes para métodos *neighborhood,* utilizando métodos estocásticos para su resolución. Por un lado se emplea una estrategia estocástica para mejorar la solución actual con el objetivo de converger a un óptimo local, y por otro una perturbación estocástica con el objetivo de explorar otros vecindarios. El método logra mejores resultados en general que el paquete de *software* Cplex y al ser un algoritmo estocástico se necesita de varias corridas a la hora de la experimentación computacional.

4.6 Heurísticas de aprendizaje automático

La alta relevancia actual de métodos de aprendizaje automático ha llevado a que se utilice en varios ámbitos de conocimiento. Una de las áreas que aprovechó esta tecnología insurgente fue el estudio de problemas MILP, específicamente el uso de heurísticas. Los métodos de

aprendizaje automático han sido aplicados exitosamente a otro tipo de problemas NP-hard (Zhang *et al*., 2023) por lo que fue lógico la incursión de investigadores para poder utilizar esta tecnología en problemas MILP. Aprendizaje automático, o *Machine Learning*, es una rama de la inteligencia artificial que se centra en desarrollar algoritmos y modelos que permiten aprender de datos y hacer predicciones o toma de decisiones sin ser programados explícitamente para cada tarea. En lugar de seguir reglas fijas, los modelos de aprendizaje automático identifican patrones en los datos a través de distintas técnicas. A continuación, se explican tres de ellas que resultan interesantes en el contexto de MILPs:

1. Aprendizaje supervisado: Se entrena el modelo con un conjunto de datos etiquetados donde cada entrada tiene una salida correspondiente.

2. Aprendizaje por imitación: Se basa en la idea de que un agente puede aprender a realizar tareas observando y replicando el comportamiento de un experto o de otros agentes.

3. Aprendizaje por refuerzo: Este enfoque implica que un agente aprende a tomar decisiones mediante prueba y error, recibiendo recompensas o penalizaciones por sus acciones.

En cuanto a las heurísticas basadas en aprendizaje automático, según Liu *et al*., (2023), se pueden clasificar en dos tipos:

- De apoyo: Investigar el uso de aprendizaje automático para aprender a tomar decisiones dentro de otras heurísticas MILP. Los métodos pueden ser una versión más económica en tiempo de resolución de algún método más complejo. Consiste básicamente en reemplazar un fragmento de una heurística por una metodología basada en aprendizaje automático.

- Autocontenido: Enseñar a algoritmos para que sean capaces de producir soluciones primales para MILP a partir de aprendizaje automático. Son algoritmos que basándose en el problema y en una base de datos busca dar con la solución óptima del problema.

En Zhang *et al*., (2023) se proponen varias aplicaciones heurísticas de ML donde se destaca el rol de apoyo del algoritmo que utiliza ML en diferentes heurísticas como FP, LNS, LB o LSN. Para FP se implementa aprendizaje por refuerzo para la resolución del método para buscar soluciones factibles más eficientemente. En el segundo, se propone aprendizaje por imitación y por refuerzo para poder elegir la creación y la destrucción de puntos. Con ello se explorará el vecindario creando un modelo en donde se predice una partición posible a explorar por vez y se propone un método para poder predecir el tamaño del vecindario. Otro método propuesto en el documento es el llamado *local rewrite*, este busca hallar una solución inicial y mejorarla modificando valores de las entradas. Además, propone una heurística en la que el uso de ML es el foco principal. La heurística se llama *predict & pick* y consiste en el uso de ML para la elección de qué heurísticas utilizar para la resolución de un problema específico, pues se llega a la conclusión de que el funcionamiento de las heurísticas dependen del problema a resolver. De esta forma, se podrá elegir cuál heurística se ajusta mejor a la situación, recomendando su utilización para la resolución de los nodos en un árbol de búsqueda del método de B&B.

A continuación, se lista otras aplicaciones planteadas en este documento:

- Predecir restricciones redundantes, buenas soluciones iniciales y espacios afines

(espacios en donde es probable encontrar la solución óptima).

- Decidir cuándo parar la búsqueda del método de *branch & bound* sin perder calidad de la solución.

- Elegir los parámetro de paquetes de *software* para obtener una mejor solución o para poner un tiempo de parada aceptable.

- Fijar variables según ML, acelerando el proceso y sin perder calidad de la solución.

El algoritmo propuesto en Wu *et al*., (2022) tiene como base utilizar un "oráculo" para guiar la búsqueda a través de la descomposición de DW y CG. En este método, el "oráculo" va a determinar espacios de la región factible donde es más probable encontrar una solución óptima. Para hacerlo, utiliza métodos estocásticos y un aprendizaje supervisado al que se le enseña a través de la solución del problema relajado, DW y *column generation* determinar patrones en la solución óptima. Este método logra dar con una mejora en tiempos y calidad de solución al compararse con otras heurísticas de la literatura.

Para poder utilizar el aprendizaje automático es común cambiar la estructura del problema MILP para que pueda ser usada como *input* en los procesos de aprendizaje. Una estructura que permite esto se denomina como representación de grafo bipartito. Esta representación busca pasar de una formulación MILP a una representación en grafos, común en ML. Se tiene dos grupos de nodos, un grupo representa las variables mientras que el otro representa la cantidad de restricciones. Los vértices que unen los nodos buscan representar la aparición de una variable en una restricción. Quien utiliza la formulación en grafos bipartitos del problema MILP es Ye *et al*., (2023) a través de la herramienta *Graph Neural Network* (GNN). Este modelo tiene la utilidad de poder analizar problemas representados como grafos y las características del problema. En primer lugar, se pueden asociar variables según la cantidad de restricciones que comparten. Aquellas que comparten muchas restricciones son afines y se sabe que al modificar una variable es probable que el resto también deban ser modificadas. A su vez, se procura agrupar aquellas variables cuya solución óptima presenta comportamientos similares, estas agrupaciones permiten que el algoritmo pueda predecir soluciones óptimas. A través de este modelo y utilizando *Gradient Boosting Decision Tree* (GBDT), el cual es alimentado con una base de datos, se obtiene un método guiado de *neighborhood*. Este método resulta en una de forma eficiente en cuanto a solución óptima y tiempos.

Otra heurística que utiliza un modelo grafo bipartito es Huang *et al*., (2023-a) en donde se usa ML mezclado con algún método de *Large Neighborhood Search.* El método recurre a muestras halladas a partir de otras heurísticas, que permiten asignar pesos a cada variable para determinar qué variables se deben modificar. Por ejemplo, al resolver el problema original mediante LB se obtienen soluciones que logran buenos resultados y estas son almacenadas con un alto peso asociado. Con los pesos asignados se crea una base de datos para acotar la región factible según los valores para los pesos. A través de una idea similar, en Wu *et al*., (2019) se plantea la formulación de problemas MILP similares al original pero disminuyendo su tamaño. Con estas, bajo la hipótesis de que el conocimiento es transferible de los problemas pequeños al grande, se determina las variables a ser fijadas a través de la solución inicial y así reducir la región factible del problema.

Por otro lado, utilizando como base el método de LB, en Liu *et al*., (2023) se propone la implementación de ML con el objetivo de mejorar los tiempos y la calidad de la solución. Originalmente el valor del tamaño del vecindario era un valor dado por el algoritmo elegido. El

método propone que el valor óptimo para el tamaño del vecindario va a depender del MILP y que utilizando ML se puede hallar. En base a las pruebas se obtiene que el algoritmo mejora en todo sentido al LB original.

En Ding *et al*., (2019) se propone que generalmente se trata como problemas diferentes problemas MILP que en su formulación son parecidos. Por lo tanto, se propone un método que utiliza ML en todos estos problemas aunque sean para tareas muy distintas entre sí pero que comparten una estructura en común. Utilizando esta información y entendiendo que fijar una variable de decisión de forma incorrecta puede llevar a la infactibilidad es que el método propone no fijar directamente, sino que se identifican las variables predecibles y se utiliza la información para realizar un corte en el espacio de soluciones. Este método permite buenos resultados para la obtención de problemas con alta cantidad de variables y restricciones.

La heurística planteada en Huang *et al*., (2020) es autocontenida, pues usa aprendizaje automático como motor del método. En este se utiliza como base de datos varios problemas MILP ya optimizados. Se propone un algoritmo que busca patrones y tendencias para determinar en un nuevo problema dónde se encuentran las soluciones óptimas. Para hacerlo, se utiliza el árbol de B&B y se definen probabilidades para la elección de ramas que se espera tengan la solución óptima. Se utilizan 1000 problemas de pequeña escala para poder enseñar el método y se logran resultados rápidos con alta calidad de solución. Un método similar al propuesto es el de Shen *et al*., (2023) en donde se utiliza aprendizaje automático en base a un método B&B. Ambos métodos van a lograr la obtención de probabilidades para guiar al árbol de búsqueda del método de B&B, pero se destaca que Shen *et al*., (2023) funciona mejor para problemas de gran tamaño, mientras que Huang *et al*., (2020) tiene un mejor rendimiento para pequeñas instancias.

Varios algoritmos heurísticos introducen parámetros en su funcionamiento. Una configuración adecuada de los parámetros ayuda a mejorar el proceso de búsqueda en el espacio de soluciones y así reducir el costo computacional. Estos parámetros se fijan o se toman varios y se decide cual obtiene mejores resultados. En Hajiyan & Yaghini (2019) se busca determinar estos valores a través de ML en el contexto de una metodología de LB. Se distinguen dos tipos de casos, parámetros *offline* y *online*, definiendo los parámetros que toman valores fijos o aquellos que se les permite cambiar su valor. El control de parámetros *online* toma retroalimentación durante la ejecución del algoritmo y utiliza esa información para establecer un marco y determinar la dirección del cambio en la tendencia de los parámetros para las siguientes iteraciones. El enfoque de control de parámetros *online* se basa en algoritmos evolutivos, donde la evolución de generaciones sucesivas se utiliza para implementar un proceso auto-adaptativo para la configuración de parámetros. Se interrumpe el algoritmo en cada iteración y se evalúa el valor del parámetro, verificando la velocidad de convergencia, el estado de la solución actual y el valor objetivo para luego ajustar. Estos enfoques ofrecen cambios de alta precisión para adaptarse a la tendencia del algoritmo.

En cuanto a la resolución de problemas específicos, en Wei *et al*., (2019) se busca resolver un problema de determinación de tamaño de lotes. Utilizando como base al método de *relajar y fijar*, se utiliza un algoritmo en base a soluciones anteriores para poder elegir cuáles variables va a fijar el método. Se obtienen buenos resultados del problema específico y se menciona que aunque el método se haya configurado para la resolución de un problema específico, se podría llegar a generalizarlo.

En Clímaco *et al*., (2019) se busca resolver el problema de ruteo con el objetivo de cumplir demandas. Se utiliza la heurística MDM-GRASP-PR del estado del arte y se le aplica *data*

mining, buscando la obtención de datos claves del problema que se utilizan en otras heurísticas. Se logra mejorar el resultado tanto en tiempos como en calidad de la solución y es un método que se puede extender.

4.7 Heurísticas basadas en Rounding

Dentro de las diferentes estrategias heurísticas, el *rounding* es una técnica muy utilizada debido a su sencillez y mejora en los tiempos de solución. La heurística de *rounding* toma una solución del modelo con variables continuas y discretiza los valores de la entrada del vector al redondear, mediante diferentes métodos, buscando mejoras en tiempos de ejecución pero con indeterminación en cuanto a la calidad. Además, surge el problema de asegurar la factibilidad de la solución, pues la actividad de redondear no asegura estar dentro de la región factible. Aunque la estrategia de rounding es ampliamente utilizada, suele combinarse con el resto de métodos. De esta manera, se obtiene únicamente una sola heurística de las relevadas que utiliza métodos de rounding como núcleo de método, la cual se va a detallar a continuación.

En Neumann *et al*., (2019) se define *inner parallel set (*IPS*),* concepto que resulta muy útil para las heurísticas de redondeo, pues define un espacio en el que al redondear las entradas de cualquier punto dentro del IPS no se rompe ninguna restricción del problema. De esta forma se obtiene un punto dentro de la región factible al redondear y se puede determinar que el IPS se encuentra dentro de la región factible. En caso de encontrar este IPS se crea un método trivial que permite resolver cualquier problema con variables discretas como uno de variables continuas en donde se modifica el dominio de las variables y luego se determina el espacio factible como el IPS. En el resto del documento, se define y demuestra teoremas para determinar qué problemas son los que permiten crear un IPS y generan un método utilizando esta idea. La forma de determinar si existe y el cálculo del IPS es costoso computacionalmente pero da mejores resultados (en los casos de existencia de IPS) que otros métodos de la literatura.

5. Conclusiones

A lo largo de este trabajo se buscó dar un acercamiento teórico a la temática de resolución de problemas MILP mediante heurísticas, brindando las principales definiciones y conceptos asociados. Se realizó un estudio de la literatura de los últimos 5 años relacionado a métodos de resolución eficientes para MILP focalizado en métodos heurísticos y poniendo énfasis en la mejora de tiempos de resolución, manteniendo calidad en la solución. Para hacerlo se relevaron 85 métodos que fueron ordenados en un sistema de clasificación formado por métodos de aprendizaje automático, descomposición, *relax*, *rounding*, estocásticas, *neighborhood* y *fixing*, para así estudiar la importancia y determinar tendencias del área de estudio. En particular, se encontró un creciente interés en aprendizaje automático en base a la cantidad de artículos que apuntan a este tema. Este particular interés se puede fundamentar con un alto grado de maduración del resto de las heurísticas, ocasionando la conjunción de las mismas con métodos de aprendizaje automático.

Se determinó que la mayoría de las metodologías estudiadas cumplen el requisito de adaptabilidad a un alto espectro de problemas y que la elección del mejor método dependerá del tipo de problema, tamaño del mismo, características de la región factible, grado de dificultad de los procedimientos y objetivo en base a la relación deseada entre tiempo de ejecución y la calidad de la solución. En base a lo expresado en la revisión de la literatura se concluye que la metodología que mejor se adapta a las distintas características del problema es *neighborhood*, explicando así su alto desarrollo de investigación, pues no poseen restricción en cuanto al tipo del problema y las características de la región factible. Si bien no son las mejores para la resolución de problemas de gran tamaño, siendo superadas por heurísticas de descomposición o de aprendizaje automático, se compensa con una menor dificultad de implementación. Además, permiten una alta flexibilidad para la relación entre los tiempos y la calidad de solución ajustándose al problema que se plantea.

De acuerdo a lo desarrollado en el documento se estima que se logró el relevamiento de un alto espectro de metodologías, permitiendo la resolución de una considerable cantidad de problemas. Sin embargo, durante la revisión de literatura se encontraron una cantidad significativa de heurísticas estocásticas, las cuales no fueron estudiadas en este documento, por lo que se considera un lineamiento pendiente a expandir en el proyecto.

Referencias

Achterberg, T. (2007). Constraint integer programming (Doctoral dissertation). Zuse Institute Berlin.

Achterberg, T., & Berthold, T. (2007). Improving the feasibility pump. *Discrete Optimization*, *4*(1), 77–86.

Baena, D., & Castro, J. (2023). The Chebyshev center as an alternative to the analytic center in the feasibility pump. *Optimization Letters, 17*(8), 2201-2213.

Balas, E., & Martin, C. H. (1980). Pivot-and-complement: A heuristic for 0-1 programming. *Management Science*, *26*(1), 86–96.

Balas, E., & Martin, C. H. (1986). Pivot and shift: A heuristic for mixed integer programming (Technical Report). GSIA, Carnegie Mellon University.

Balas, E., Ceria, S., Dawande, M., Margot, F., Pataki, G. (2001). OCTANE: A new heuristic for pure 0-1 programs. *Operations Research*, *49*(2), 207–225.

Balas, E., Schmieta, S., & Wallace, C. (2004). Pivot and shift - A mixed integer programming heuristic. *Discrete Optimization*, *1*(1), 3–12.

Bansal, A., & Uzsoy, R. (2021). Constraint violation reduction search for 0–1 mixed integer linear programming problems. *Engineering Optimization, 53*(4).

Beasley, J. E. (2024). A heuristic for the non-unicost set covering problem using local branching. *International Transactions in Operational Research, 31*(5).

Benavent, E., Corberán, A., Laganà, D., & Vocaturo, F. (2023). A two-phase hybrid algorithm for the periodic rural postman problem with irregular services on mixed graphs. *European Journal of Operational Research, 307*(1).

Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik, 4*(1).

Bertacco, L., Fischetti, M., & Lodi, A. (2007). A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, *4*(1), 63–76.

Berthold, T. (2007). RENS - relaxation enforced neighborhood search. *Technical Report 07-28*. Berlin: ZIB.

Berthold, T., Lodi, A., & Salvagnin, D. (2018). Ten years of feasibility pump, and counting. *EURO Journal on Computational Optimization, 7*(1), 1-28.

Boschetti, M. A., Letchford, A. N., & Maniezzo, V. (2023). Matheuristics: Survey and synthesis. *International Transactions in Operational Research, 30*(6), 2840-2866.

Boukhari, S., & Hifi, M. (2024). Combining local branching and descent method for solving the multiple-choice knapsack problem with setups. *International Transactions in Operational Research, 31*(1).

Boutarfa, Y., Senoussi, A., Brahimi, N., & Aouam, T. (2024). Integration and substitution in hybrid manufacturing and refurbishing systems. *International Journal of Production Economics, 274.*

Bragin, M. (2023). Survey on Lagrangian Relaxation for MILP: Importance, challenges, historical review, recent advancements, and opportunities. *Annals of Operations Research, 333*(1).

Celaya, M., Kuhlmann, S., Paat, J., & Weismantel, R. (2022). Improving the Cook et al. proximity bound given integral valued constraints. *Lecture Notes in Computer Science,* 84–97.

Cereser, B. L. H., de Oliveira, A. R. L., & Moretti, A. C. (2022). A new strategy to solve linear integer problems with simplex directions. *Sobrapo,* 42.

Chitsaz, M., Cordeau, J. F., & Jans, R. (2019). A unified decomposition matheuristic for assembly, production, and inventory routing. *INFORMS Journal on Computing, 31*(1).

Clímaco, G., Rosseti, I., Simonetti, L., & Guerine, M. (2019). Combining integer linear programming with a state-of-the-art heuristic for the 2-path network design problem. *Wiley*, 26(2), 523-534.

Cook, W., Gerards, A. M. H., Schrijver, A., & Tardos, E. (1986). Sensitivity theorems in integer linear programming. *Mathematical Programming, 34*, 251–264.

Danna, E., Rothberg, E., & Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve MILP solutions. *Mathematical Programming, 102*(1), 71-90.

Darwiche, M., Conte, D., Raveaux, R., & T'Kindt, V. (2018). Graph edit distance: Accuracy of local branching from an application point of view. *Pattern Recognition Letters, 134.*

Ding JY, Zhang C, Shen L., Li S., Wang B., Xu Y., Song L. (2020): Accelerating primal solution findings for mixed integer programs based on solution prediction. AAAI 2020 - 34th AAAI Conference on Artificial Intelligence.

Dupin, N., & Talbi, E.-G. (2018). Parallel matheuristics for the discrete unit commitment problem with min-stop ramping constraints. *International Transactions in Operational Research, 27*(1), 219–244.

Faaland, B. H., & Hillier, F. S. (1979). Interior path methods for heuristic integer programming procedures. *Operations Research*, *27*(6), 1069–1087.

Farkiani, B., Bakhshi, B., & Mirhassani, S. A. (2019). A fast near-optimal approach for energy-aware SFC deployment. *IEEE Transactions on Network and Service Management, 16*(4).

Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming*, *98*(1), 23–47.

Fischetti, M., Polo, C., & Scantamburlo, M. (2004). A local branching heuristic for mixed-integer programs with 2-level variables. *Networks, 44*(1), 61-72.

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, *104*(1), 91–105.

Fischetti, M., & Lodi, A. (2008). Repairing MILP infeasibility through local branching. *Computers & Operations Research, 35*(4), 1436-1445.

Fischetti, M., & Salvagnin, D. (2009). Feasibility pump 2.0. *Mathematical Programming Computation*, *1*(2), 201–222.

Fischetti, M., & Lodi, A. (2010). Heuristics in mixed integer programming. *Encyclopedia of Operations Research and Management Science*. Wiley.

Frangioni, A., Pan, S., Traversi, E., & Wolfler Calvo, R. (2022). A constraints-aware Reweighted Feasibility Pump approach. *Operations Research Letters, 50*(2), 185–191.

Gamrath, G., Berthold, T., Heinz, S., & Winkler, M. (2019). Structure-driven fix-and-propagate heuristics for mixed integer programming. *Mathematical Programming Computation, 11*(4), 675-702.

Ghirardi, M., & Amerio, A. (2019). Matheuristics for the lot sizing problem with back-ordering, setup carry-overs, and non-identical machines. *Computers & Industrial Engineering, 129*, 81–93.

Ghosh, S. (2007). DINS, a MIP Improvement Heuristic. In: Fischetti, M., Williamson, D.P. (eds) Integer Programming and Combinatorial Optimization. IPCO 2007. Lecture Notes in Computer Science, vol 4513. Springer, Berlin, Heidelberg.

Ghosh, S., & Hayward, R. B. (2009). Pivot and Gomory cut: A MILP feasibility heuristic (Technical Report). University of Alberta.

Hajiyan, H., & Yaghini, M. (2020). An adaptive structure on a new local branching algorithm using instantaneous dimensions and convergence speed: A case study for multi-commodity network design problems. *SN Applied Sciences, 2*(6).

Hansen, P., Mladenović, N., & Urošević, D. (2006). Variable neighborhood search and local branching. *Computers & Operations Research, 33*(11), 3034–3045.

Hendel, G. (2021). Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation, 13*(4), 491–526.

Hillier, F. S. (1969). Efficient heuristic procedures for integer linear programming with an interior. *Operations Research*, *17*(3), 600–637.

Huang, L., Chen, X., Huo, W., Wang, J., Zhang, F., Bai, B., & Shi, L. (2020). Improving primal heuristics for mixed integer programming problems based on problem reduction: A learning-based approach. *17th International Conference on Control, Automation, Robotics and Vision, ICARCV 2020*, 181-186.

Huang, T., Ferber, A., Tian, Y., Dilkina, B., & Steiner, B. (2023-a). Searching large neighborhoods for integer linear programs with contrastive learning. *Proceedings of Machine Learning Research*, 202, 1-9.

Huang, T., Ferber, A., Tian, Y., Dilkina, B., & Steiner, B. (2023-b). Local branching relaxation heuristics for integer linear programs. *Lecture Notes in Computer Science.*

Ibaraki, T., Ohashi, T., & Mine, H. (1974). A heuristic algorithm for mixed-integer programming problems. *Mathematical Programming Study*, *2*, 115–136.

Islas F. & Testuri C. (2023): Método Branch & Bound en Optimización de Problemas de Producción. *Instituto de Computación, Facultad de Ingeniería, UdelaR.*

Joncour, C., Kritter, J., Michel, S., & Schepler, X. (2023). Generalized Relax-and-Fix heuristic. *Computers and Operations Research, 149.*

Khalil, E. B., Vaezipoor, P., & Dilkina, B. (2022). Finding backdoors to integer programs: A Monte Carlo tree search framework. *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022, 36.*

Kim, D., & Pardalos, P. M. (1999). A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters, 24*, 195–203.

Koguma, Y. (2024). Tabu search-based heuristic solver for general integer linear programming problems. *IEEE Access, 12.*

Liu, D., Fischetti, M., & Lodi, A. (2023). Learning to Search in Local Branching. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 36, 2152-2160.

Løkketangen, A., Jornsten, K., & Storøy, S. (1994). Tabu search within a pivot-and-complement framework. *International Transactions in Operational Research*, *1*(3), 305–317.

Lübbecke, M. (2011). Column generation. *Encyclopedia of Operations Research and Management Science*, 182-186.

Luteberget, B., & Sartor, G. (2023). Feasibility jump: An LP-free Lagrangian MIP heuristic. *Mathematical Programming Computation, 15*(2).

Ma, Z., Zhong, H., Xia, Q., Kang, C., Wang, Q., & Cao, X. (2020). A unit commitment algorithm with relaxation-based neighborhood search and improved relaxation inducement. *IEEE Transactions on Power Systems, 35*(5).

Mo, L., Kritikakou, A., & He, S. (2019). Energy-aware multiple mobile chargers coordination for wireless rechargeable sensor networks, *6*(5).

Mo, L., Kritikakou, A., & Sentieys, O. (2019). Approximation-aware task deployment on asymmetric multicore processors. *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition*, 1230-1235.

Mexi, G., Berthold, T., & Salvagnin, D. (2023). Using multiple reference vectors and objective scaling in the feasibility pump. *EURO Journal on Computational Optimization, 11*(1), 1-14.

Naoum-Sawaya, J., & Elhedhli, S. (2010). An interior point cutting plane heuristic for mixed integer programming (Technical Report). University of Waterloo.

Nediak, M., & Eckstein, J. (2007). Pivot, cut, and dive: A heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, *13*(5), 471–503.

Neumann, C., Stein, O. & Sudermann-Merx, N. A feasible rounding approach for mixed-integer optimization problems. Comput Optim Appl 72, 309–337 (2019)

Rahmaniani, R., Crainic, T., Gendreau, M., & Rei, W. (2017). The Benders decomposition algorithm: A literature review. *Volume 259*(3).

Riccardi A., Minisci E., Akartunalı K., Greco C., Kershaw A., Hashim A. (2021). Introduction to Optimisation. Optimization Under Uncertainty with Applications to Aerospace Engineering (pp.223-268)

Rothberg, E. (2007). An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing, 19*(4), 534–541.

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., & Uchoa, E. (2017). Primal heuristics for branch and price: The assets of diving methods. *INFORMS Journal on Computing, 31*(2).

Salvagnin, D., Roberti, R., & Fischetti, M. (2024). A fix-propagate-repair heuristic for mixed integer programming. *Mathematical Programming Computation*, 1-28.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science*, Volume 1520, 417–431.

Shen, Y., Sun, Y., Eberhard, A., & Li, X. (2022). Learning primal heuristics for mixed integer programs. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2021*, 3271-3279.

Shoja, S., & Axehill, D. (2023). Exact complexity certification of start heuristics in branch-and-bound methods for mixed-integer linear programming. *IEEE Transactions on Control Systems Technology*.

Tonbari, M., & Ahmed, S. (2023). Consensus-based Dantzig-Wolfe decomposition. *European Journal of Operational Research, 307*(3).

Wei, M., Qi, M., Wu, T., & Zhang, C. (2019). Distance and matching-induced search algorithm for the multi-level lot-sizing problem with substitutable bill of materials. *European Journal of Operational Research, 277*(2), 473-487.

Wu, T., Xiao, F., Zhang, C., Zhang, D., & Liang, Z. (2019). Regression and extrapolation guided optimization for production–distribution with ship–buy–exchange options. *Transportation Research Part E: Logistics and Transportation Review, 130*, 168–182.

Wu, T., Huang, L., Liang, Z., Zhang, X., & Zhang, C. (2022). A supervised learning-driven heuristic for solving the facility location and production planning problem. *European Journal of Operational Research, 301*(2), 347-358.

Yarahmadi, M. N., MirHassani, S. A., & Hooshmand, F. (2023). A heuristic method to find a quick feasible solution based on the ratio programming. *Operational Research, 23*(3).

Ye, H., Xu, H., Wang, H., Wang, C., & Jiang, Y. (2023). GNN&GBDT-Guided Fast Optimizing Framework for Large-scale Integer Programming. *Proceedings of Machine Learning Research*, 202, 1-11.

Zaozerskaya, L. (2021). A heuristic for a special case of the generalized assignment problem with additional conditions. *Journal of Physics: Conference Series, 1791*(1).

Zhang, W., & Nicholson, C. D. (2019). Objective scaling ensemble approach for integer linear programming. *Springer Science+Business Media, LLC, part of Springer Nature.*

Zhang, C., Dong, Z., & Yang, L. (2022). A feasibility pump based solution algorithm for two-stage robust optimization with integer resources of energy storage systems. *IEEE Transactions on Sustainable Energy, 12*(3).

Zhang, J., Liu, C., Li, X., Zhen, H. L., Yuan, M., Li, Y., & Yan, J. (2023). A survey for solving mixed integer programming via machine learning. *Neurocomputing, 519*, 1-14.

Zhang, Z., Liu, M., Xie, M., & Dong, P. (2023). A mathematical programming-based heuristic for coordinated hydrothermal generator maintenance scheduling and long-term unit commitment. *International Journal of Electrical Power and Energy Systems, 147.*

Apéndice II: Validación del Modelo

Con el objetivo de evaluar la robustez y aplicabilidad del modelo propuesto, se plantean distintos escenarios que permiten observar su comportamiento tanto en situaciones límite como en condiciones controladas. A través de estos casos, se pretende verificar si el modelo responde adecuadamente ante restricciones extremas que comprometen su viabilidad, así como también comprobar su correcto funcionamiento en contextos factibles y representativos de su uso esperado. A continuación, se detallan los 8 casos considerados para esta validación.

En los primeros cuatro, se analizan situaciones límite en las que el modelo pierde validez o deja de ser aplicable. Primero se considera el caso en que no hay empresas generadoras de residuos y en el segundo el caso en que no hay días para hacer los recorridos. En el tercer caso se plantea que las empresas no generan residuos y por último, en el cuarto caso se plantea que la capacidad del contenedor es nula. Por lo tanto, para todos estos casos se espera que el resultado sea infactible debido a que se imposibilita la realización de los recorridos.

Para la siguiente validación se busca demostrar un correcto funcionamiento general para el método. Por lo tanto, se crea un caso en donde cuatro empresas generan residuos cuya suma es menor a la capacidad del vehículo. En la figura X se se disponen 3 empresas y el centro de disposición formando un cuadrado de lado 100 unidades de distancia. Considerando que la capacidad del contenedor es 28.8 y la suma del total de las empresas es 18,27 entonces es fácil determinar que el óptimo se encuentra al realizar el recorrido siguiendo los lados del cuadrado, evitando las diagonales. Es facil determinar que el óptimo es 400 pues se obtiene al recorrer las aristas del cuadrado. Esto se obtiene al correr el modelo y se obtiene el resultado esperado como se ve en la Figura X.

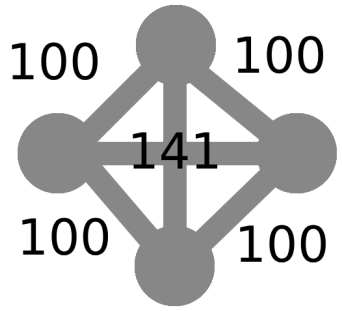

Figura X - Diagrama de caso 1

```
Objective:  ruteo = 400 (MINimum)
```

Figura 1 - Resultado computacional obtenido al ejecutar el caso 5.

Otra validación se creó con el objetivo de evaluar la restricción que obliga a no hacer un recorrido en una misma semana. Por este motivo se creó un caso en donde hay una única empresa que genera más que la cantidad del vehículo contando con un límite de 6 días hábiles. El modelo logra solucionar correctamente este problema haciendo un primer recorrido el primer

día y el otro el último día. El resultado se observa en la figura X donde el día 1 y el día 6 se realizan los recorridos al ser el valor de las variables 1 para estas fechas.

No.	Column name		Activity	L
1	x[1,1,1]	*	0	
2	x[1,1,2]	*	0	
3	x[1,1,3]	*	0	
4	x[1,1,4]	*	0	
5	x[1,1,5]	*	0	
6	x[1,1,6]	*	0	
7	x[1,2,1]	*	1	
8	x[1,2,2]	*	0	
9	x[1,2,3]	*	0	
10	x[1,2,4]	*	0	
11	x[1,2,5]	*	0	
12	x[1,2,6]	*	1	
13	x[2,1,1]	*	1	
14	x[2,1,2]	*	0	
15	x[2,1,3]	*	0	
16	x[2,1,4]	*	0	
17	x[2,1,5]	*	0	
18	x[2,1,6]	*	1	
19	x[2,2,1]	*	0	
20	x[2,2,2]	*	0	
21	x[2,2,3]	*	0	
22	x[2,2,4]	*	0	
23	x[2,2,5]	*	0	

Figura 2 - Valores de las variables del resultado de la validación 6

La siguiente validación busca evaluar la restricción que evita la aparición de subciclos. Para hacerlo se presenta un caso para forzar la generación de subciclos pues se creó pensando una penalización por ir de una empresa a otra. Se creó un caso en donde hay 4 empresas ubicadas de forma tal que están en extremos opuestos y de forma simétricas. De esta forma, en caso de existir subciclos, sería mucho mejor para la solución evitar ir a la otra punta. La figura X se representa un diagrama representativo del caso.

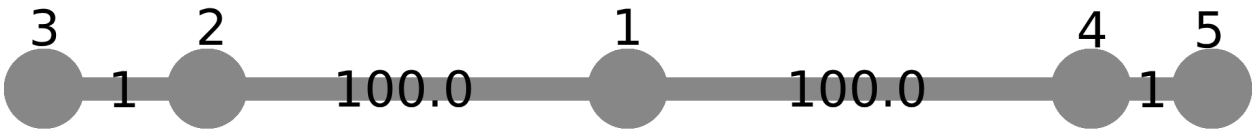

Figura 3 - Diagrama validación 7.

Luego de ejecutar el modelo con los datos de validación se obtiene un resultado como se muestra en la Figura X donde se demuestra que no se generan subciclos y que el recorrido óptimo es 1-4-5-3-2-1.

1	x[1,1,1]	*	0
2	x[1,2,1]	*	0
3	x[1,3,1]	*	0
4	x[1,4,1]	*	1
5	x[1,5,1]	*	0
6	x[2,1,1]	*	1
7	x[2,2,1]	*	0
8	x[2,3,1]	*	0
9	x[2,4,1]	*	0
10	x[2,5,1]	*	0
11	x[3,1,1]	*	0
12	x[3,2,1]	*	1
13	x[3,3,1]	*	0
14	x[3,4,1]	*	0
15	x[3,5,1]	*	0
16	x[4,1,1]	*	0
17	x[4,2,1]	*	0
18	x[4,3,1]	*	0
19	x[4,4,1]	*	0
20	x[4,5,1]	*	1
21	x[5,1,1]	*	0
22	x[5,2,1]	*	0
23	x[5,3,1]	*	1
24	x[5,4,1]	*	0

Figura 4 - Valores de las variables del resultado de la validación 7

Para la última validación se busca demostrar la optimalidad de las distancias. Para hacerlo se crean los datos de un caso como el de la Figura X. Como número 1 se representa al centro de distribución ubicado en el centro del diagrama. En solitario se encuentra la empresa número dos la cual genera 30 unidades de residuos. En el otro extremo, se ubican las empresas 3, 4 y 5 las cuales generan 10 unidades de residuos y forman un cuadrado al unir los puntos con el centro de distribución. La capacidad del contenedor es de 30 unidades y se cuentan con 2 días para realizar los recorridos. Es fácil observar que el resultado óptimo es 6 debido a que en un día se toman las 30 unidades de residuos de la empresa 2 y el otro día las otras 30 de las empresas 3, 4 y 5 siguiendo el orden de las aristas del cuadrado.

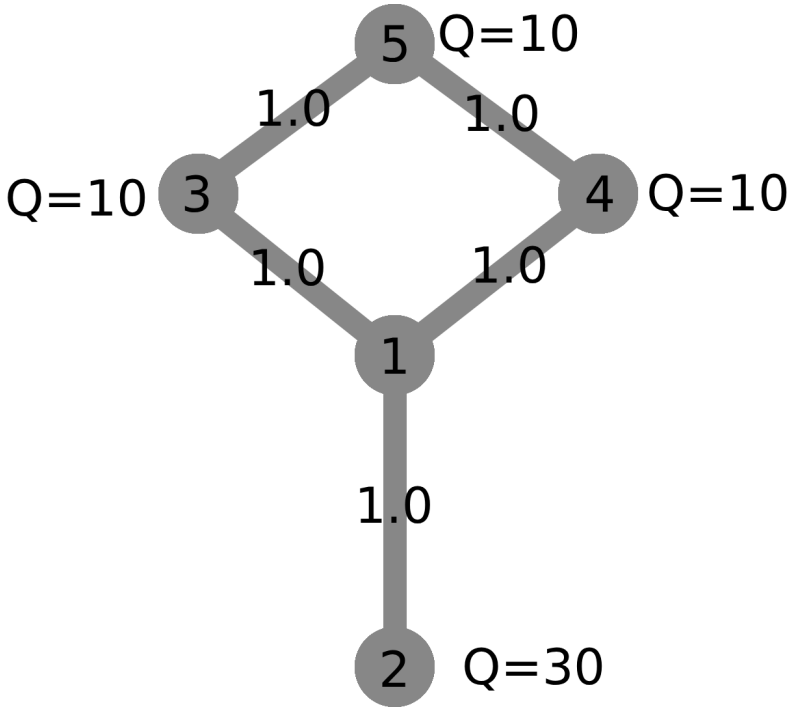

Figura 5 - Diagrama datos de la validación 8

Como se observa en la figura X el resultado alcanzado al correr el modelo con los datos de la instancia es el esperado. Además, las variables toman el valor esperado recomendando el primer día el recorrido 1-2-1 y el segundo día el recorrido 1-4-5-3-1.

```
Problem:    Ruteo
Rows:       178
Columns:    120 (52 integer, 52 binary)
Non-zeros:  642
Status:     INTEGER OPTIMAL
Objective:  ruteo = 6 (MINimum)
```

Figura 6 - Resultado alcanzado al correr el modelo con los datos de la instancia 8

Apéndice III: Modelado del problema en GNU

```
#Funcion objetivo
#

minimize ruteo: sum{e in E, r in E, t in T} x[e,r,t]*distancia[e,r];

#Restricciones
#

s.t. rescap{t in T}: sum{e in E,r in E} y[e,r,t] <= c;

s.t. resgenerado{e in E,t in T}: sum{r in E} y[e,r,t] <= q[e,t];

s.t. resrecolectar{e in E}: sum{t in T,r in E} y[e,r,t] = k[e];

s.t. ressentido{e in E,r in E,t in T}: y[e,r,t] <=  c*x[e,r,t];

s.t. resdemanda{e in E,t in T}: q[e,t] = k[e] - sum{r in E,n in T:n <= t-1} y[e,r,n];

s.t. resinicio{t in T}: sum{r in E} x["Paloma",r,t] = z[t];

s.t. resfinal{t in T}: sum{e in E} x[e,"Paloma",t] = z[t];

s.t. resrecorrido{t in T}: 100000*z[t] >= sum{e in E,r in E} x[e,r,t];

s.t. rescircuito{e in E diff {"Paloma"},t in T}: sum{r in E} x[e,r,t] = sum{r in E} x[r,e,t];

s.t. resevit{e in E,t in T}: x[e,e,t] = 0;

s.t. ressemanal{e in E,r in E,t in T}: sum{n in T: (t <= n) && (n <= t+4)} (x[r,e,n]) <= 1;

s.t. resmtz{e in E diff {"Paloma"}, r in E diff {e,"Paloma"},t in T}: u[e,t] - u[r,t] + m*x[e,r,t] <= m-1;

s.t. respruebа{t in T}:  z[t] <= sum{e in E diff {"Paloma"}} u[e,t];

end;
```

Figura 1 - Modelado del problema en GNU

Una vez especificada la función objetivo, variables y restricciones del problema, sólo resta brindar los datos del problema. Para ello, nuevamente el lenguaje es altamente intuitivo y permite brindar los datos como conjuntos o matrices dependiendo del caso, como se puede observar en la Figura 5 para un caso reducido de prueba. Generado el modelado del problema y de los datos, la resolución se ejecuta en la terminal del sistema operativo para acceder a la solución del mismo. Para correr el código se debe llamar al paquete de *software* de optimización en el interfaz de comandos, a modo de ejemplo se dispone la línea de código a usar en la terminal:

glpsol --model Modelo\Ruteo.mod --data Datos\Ruteo.dat --output Solucion\Ruteo.sol --tmlim 86400

El inicio *glpsol* llama al paquete de *software* de resolución para problemas MILP, *--model* indica cual archivo será utilizado como modelo en la optimización, en este caso, el archivo llamado “Ruteo.mod”, *-- data* indica cual archivo será utilizado para extraer los datos del modelo. En este caso el archivo llamado es “Ruteo.dat”, *--output* indica en dónde y qué archivo se genera para que se guarde la solución del problema una vez que termine la corrida. Este indica el valor de la solución objetivo, el gap de dualidad y el valor tomado por las variables en esta solución.

Por último, *--tmlim* indica la cantidad de segundos máxima delimitada para la ejecución del programa, pudiendo quitarse si se busca que el paquete de *software* funcione hasta lograr determinar la solución óptima. En la Figura 6 se presenta a continuación una versión reducida de los datos de entrada para el modelo, la cual fue utilizada como parte de la validación de la implementación del problema.

```
data ;

set E := 1  2   3   4   5   6   7   8   9   10;

set T := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21;

param distancia :       1   2   3   4   5   6   7   8   9   10 :=
1   0.0 69.54   26.31   45.2    71.94   63.74   100.46  72.66   37.46   74.25
2   69.54   0.0 68.9    114.74  141.48  131.04  32.41   86.32   82.11   50.35
3   26.31   68.9    0.0 56.78   81.98   65.17   101.3   46.41   63.66   55.49
4   45.2    114.74  56.78   0.0 26.77   26.66   145.48  94.87   56.1    112.24
5   71.94   141.48  81.98   26.77   0.0 26.83   172.06  115.47  77.41   137.32
6   63.74   131.04  65.17   26.66   26.83   0.0 162.97  91.72   82.11   118.48
7   100.46  32.41   101.3   145.48  172.06  162.97  0.0 114.8   106.55  74.31
8   72.66   86.32   46.41   94.87   115.47  91.72   114.8   0.0 109.76  42.84
9   37.46   82.11   63.66   56.1    77.41   82.11   106.55  109.76  0.0 104.61
10  74.25   50.35   55.49   112.24  137.32  118.48  74.31   42.84   104.61  0.0;

param c := 15.66900883;

param k :=
1   0
2   19
3   21
4   17
5   6
6   6
7   19
8   17
9   19
10 2;

param m := 10;
```

Figura 2 - Datos del problema reducido

Apéndice IV: Resultados Tablas

Resultados obtenidos a través de GLPK

Tabla 1 - Resultados alcanzados de las instancias con GLPK tras 2 horas de implementación

Instancia	Valor de función objetivo	Tiempo	Gap de dualidad
1	7529	2 horas	83,2
2	1214,15	2 horas	56
3	1774,4	2 horas	68,9
4	1099,23	2 horas	48,8
5	1801,7	2 horas	65,4
6	1278,46	2 horas	58,8
7	1561,23	2 horas	65,2
8	1492,03	2 horas	67,3
9	2371,36	2 horas	72,2

Tabla 2 - Resultados alcanzados de las instancias con GLPK obteniendo la primera solución

Instancia	Valor de función objetivo	Tiempo (s)	Gap de dualidad
1	7800,7	2584	84
2	1545,79	120	66,7
3	2083,08	360	74,1
4	1331,4	60	58,8
5	2480,59	30	75,7
6	1579,41	10	74
7	2359,6	40	79
8	2030,53	40	76,9
9	2809,04	600	76,5

Resultados obtenidos a través del método FP+OSEA

Tabla 3 - Resultados alcanzados de FP+OSEA en la instancia 1

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	11610,1	16401,3	1479,21	297,83	1777,04
2	9597,98	14925,9	1055,04	260,55	1315,59
3	13180,4	15212	662,228	262,083	924,311
4	9540,83	14847,2	881,7	271,93	1153,63
5	9992,27	18214,3	845,578	269,522	1115,1
6	11169,7	16497,7	1972,77	264,4	2237,17
7	13568,9	17339,9	1526,27	267,37	1793,64
8	10583,5	15345,3	4619,59	265,21	4884,8
9	9926,59	16104,7	773,294	267,606	1040,9
10	10748,4	15957,7	105,369	862,88	968,249

Tabla 4 - Resultados alcanzados de FP+OSEA en la instancia 2

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	3197,19	4628,26	360,53	392,071	752,601
2	2821,39	5555,85	577,683	331,856	909,539
3	2730,22	4806,86	476,468	336,939	813,407
4	2898,22	4134,87	373,244	353,622	726,866
5	2411,47	5778,87	753,195	356,555	1109,75

Tabla 5 - Resultados alcanzados de FP+OSEA en la instancia 3

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	5014,04	5456,6	390,808	326,254	717,062
2	5264,22	6709,51	1578,64	305,43	1884,07
3	4379,98	5703,68	1650,1	342,26	1992,36
4	5162,39	5921,37	874,685	336,885	1211,57
5	4496,62	5659,8	856,483	321,197	1177,68

Tabla 6 - Resultados alcanzados de FP+OSEA en la instancia 4

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	3619,57	4967,14	1006,63	309,6	1316,23
2	3778,27	5800,31	902,075	289,285	1191,36
3	4495,96	6014,54	409,315	305,433	714,748
4	3971,24	5418,9	963,103	330,707	1293,81
5	3085,69	5105,73	1067,57	322,97	1390,54

Tabla 7 - Resultados alcanzados de FP+OSEA en la instancia 5

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	3136,93	4237,63	258,159	325,007	583,166
2	3238,94	5174,99	596,717	331,737	928,454
3	2670,22	4846,57	586,885	278,443	865,328
4	2897,81	5308,91	231,255	280,863	512,118
5	3092,51	4516,52	230,673	300,815	531,488

Tabla 8 -Resultados alcanzados de *FP+OSEA* en la instancia 6

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	2368,97	2667,31	19,3717	2,3505	21,7222
2	2025,5	2448,09	14,4152	2,3463	16,7615
3	1847,79	2470,56	3,69431	2,34119	6,0355
4	1892,11	2574,64	14,6842	2,4428	17,127
5	1634,75	2090,82	3,33774	2,2526	5,59034

Tabla 9 - Resultados alcanzados de *FP+OSEA* en la instancia 7

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	2052,54	3093,07	59,2872	43,7508	103,038
2	2572,47	3881,5	155,331	43,668	198,999
3	2490,87	4327,54	115,385	48,296	163,681
4	3012,24	4036,52	56,1766	44,0384	100,215
5	3130,3	3408,66	21,7721	43,7154	65,4875

Tabla 10 - Resultados alcanzados de *FP+OSEA* en la instancia 8

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	4197,58	5044,91	356,21	253,111	609,321
2	3826,76	4288,50	262,731	255,553	518,284
3	4315,38	5641,98	831,343	262,377	1093,72
4	5144,16	5703,64	964,067	254,603	1218,67
5	4119,35	4872,15	703,139	255,876	959,015

Tabla 11 - Resultados alcanzados de *FP+OSEA* en la instancia 9

N° de prueba	Valor de función objetivo	Resultados de FP	Tiempo de FP	Tiempo de OSEA	Tiempo total
1	7991,73	8121,85	1982,33	1087,99	3070,32
2	8244,64	8652,31	4875,33	1094,98	5970,31
3	6423,18	7183,89	7012,17	1082,90	8095,07
4	5587,29	7006,60	5354,18	1046,90	6401,08
5	6102,46	7601,84	4530,54	1058,08	5588,62

Resultados obtenidos a través del método FP+LB

Tabla 12 - Resultados alcanzados de *FP+LB* en la instancia 1

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	9459,93	16801,6	854,011	2252,489	3106,5
2	8995,52	16758,3	1255,05	808,05	2063,1
3	7726,70	16238,5	955,915	7432,945	8388,86
4	9431,82	15572,8	794,983	490,037	1285,02
5	9419,20	13806,5	223,520	170,007	393,527
6	8857,00	16356,4	2493,75	252,990	2746,74
7	9387,85	13888,9	1166,02	162,330	1328,35
8	11439,4	17749,6	1535,45	139,780	1675,23
9	10184,7	14934,9	691,361	168,323	859,684
10	8943,95	17697,1	2764,75	152,700	2917,45

Tabla 13 - Resultados alcanzados de *FP+LB* en la instancia 2

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	2801,34	3863,66	116,248	167,215	283,463
2	2558,29	5882,08	573,886	225,351	799,237
3	2387,30	4779,45	376,306	141,891	518,197
4	2789,69	4857,69	1244,08	66,020	1310,10
5	2853,14	5053,38	441,966	132,595	574,561

Tabla 14 - Resultados alcanzados de *FP+LB* en la instancia 3

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	3399,40	6013,51	1430,55	2490,04	3920,59
2	3240,93	6513,15	1450,18	441,13	1891,31
3	2998,86	5172,54	661,295	1622,235	2283,53
4	3630,02	5250,18	2602,88	266,84	2869,72
5	4021,25	5670,59	755,341	1068,249	1823,59

Tabla 15 - Resultados alcanzados de *FP+LB* en la instancia 4

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	2578,32	5793,23	368,939	221,926	590,865
2	2752,13	3943,01	334,797	175,295	510,092
3	2609,73	5312,50	396,394	239,621	636,015
4	2434,21	5060,38	373,985	214,585	588,570
5	2591,23	4609,67	554,916	144,726	699,642

Tabla 16 - Resultados alcanzados de *FP+LB* en la instancia 5

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	3506,86	5395,11	847,853	292,387	1140,24
2	3589,47	6239,79	1302,24	763,03	2065,27
3	3457,72	7645,06	650,948	5534,062	6185,01
4	3499,35	6298,70	815,019	931,421	1746,44
5	3313,94	5450,74	1444,54	1912,83	3357,37

Tabla 17 - Resultados alcanzados de *FP+LB* en la instancia 6

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	1178,60	1960,40	3,04697	2,59152	5,63849
2	1852,25	2427,05	13,5998	2,2662	15,8660
3	2004,22	2478,54	9,23768	2,47492	11,7126
4	1755,86	2186,64	11,3973	2,4017	13,7990
5	1930,34	2302,34	4,48688	2,2852	6,77208

Tabla 18 - Resultados alcanzados de *FP+LB* en la instancia 7

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	2264,29	4398,08	107,961	137,680	245,641
2	2313,43	3998,37	51,106	38,1772	89,2832
3	2510,36	3868,46	83,9986	40,5154	124,514
4	2428,63	3988,97	54,8655	162,8675	217,733
5	2362,02	4331,78	87,6124	398,1516	485,764

Tabla 19 - Resultados alcanzados de *FP+LB* en la instancia 8

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	2981,42	6223,13	726,062	329,948	1056,01
2	2760,97	5035,14	447,614	428,997	876,611
3	2968,03	6115,27	1051,74	1091,36	2143,10
4	2794,83	5338,27	419,443	690,197	1109,64
5	3087,74	4948,83	363,425	123,575	487,00

Tabla 20 - Resultados alcanzados de *FP+LB* en la instancia 9

N° de prueba	Valor de función objetivo	Valor después de FP	Tiempo de FP	Tiempo de LB	Tiempo total
1	5357,63	8211,34	7649,27	1608,24	9257,51
2	*	*	*	*	*
3	*	*	*	*	*
4	*	*	*	*	*
5	*	*	*	*	*

Resultados obtenidos a través del método FP+OSEA+LB

Tabla 21 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 1

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	8584,41	15327,3	10783,7	1052,57	289,45	111,76	1453,78
2	7985,30	13947,7	11346,5	176,21	262,07	109,63	547,91
3	7877,02	15758,1	11408,9	1422,90	263,95	76,62	1763,47
4	6870,90	15043,1	13002,9	866,52	273,95	145,57	1286,04
5	8408,25	14061,9	11464,9	828,72	265,76	83,12	1177,60
6	7587,68	14818,6	12344,8	2302,07	262,70	431,01	2995,78
7	8247,35	13544,1	10552,5	860,00	277,38	165,06	1302,44
8	8483,95	15703,9	12653,5	992,77	276,04	125,99	1394,80
9	7471,19	13386,6	10628,5	426,47	275,98	300,17	1002,62
10	9468,00	14803,4	11915,3	75,19	266,39	67,09	408,67

Tabla 22 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 2

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	3030,71	5561,93	3030,71	1172,81	346,94	275,28	1795,03
2	2548,43	5430,13	2548,43	709,77	336,01	178,34	1224,12
3	2567,38	4767,08	2750,20	362,13	336,53	216,09	914,76
4	2408,23	6400,57	3224,48	694,57	334,45	35,62	1064,64
5	2592,09	6626,92	3261,80	286,39	352,21	30,49	669,09

Tabla 23 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 3

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	3016,54	4943,16	4682,71	732,80	318,93	374,86	1426,59
2	3277,49	6000,74	4477,82	423,91	322,81	230,12	976,84
3	3621,86	5977,85	4466,65	1147,09	319,08	194,81	1660,98
4	2915,51	5811,44	5466,09	1617,27	293,04	171,68	2081,99
5	3833,78	6949,17	6781,69	1453,86	282,66	253,95	1990,47

Tabla 24 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 4

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	2416,74	3660,41	3304,37	74,02	291,66	109,11	474,78
2	2488,67	4968,41	2690,88	1538,71	321,48	174,36	2034,55
3	2598,70	4870,03	2840,27	592,71	281,60	157,45	1031,75
4	2427,61	4944,00	3660,46	404,48	288,93	40,70	734,11
5	2786,10	4117,52	2921,33	227,48	269,19	175,69	672,36

Tabla 25 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 5

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	3232,73	5065,57	4050,29	646,14	280,33	80,10	1006,57
2	3461,44	5344,49	5189,59	543,60	274,26	162,17	980,02
3	3563,23	5282,42	4932,91	540,64	274,73	301,94	1117,31
4	3153,99	5462,54	3657,90	449,37	272,19	123,59	845,16
5	4007,13	6663,65	4007,13	1024,44	310,55	179,25	1514,24

Tabla 26 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 6

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	1878,91	2580,61	1878,91	22,28	2,47	2,58	27,33
2	1947,66	2670,43	1947,66	21,80	2,17	2,47	26,44
3	1869,04	2232,47	1869,04	6,17	2,20	2,36	10,73
4	2006,22	2450,92	2163,34	8,16	2,12	2,49	12,77
5	2067,55	2458,19	2067,55	10,73	2,21	2,33	15,27

Tabla 27 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 7

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	2266,39	3204,66	2442,64	19,22	39,01	31,72	89,95
2	2138,19	3571,79	2188,23	113,57	40,52	30,29	184,37
3	2416,22	4077,29	3120,24	50,75	42,83	66,52	160,10
4	2254,08	4449,97	2254,08	102,71	41,54	23,70	167,95
5	2471,80	4247,06	2471,80	53,13	43,31	26,10	122,54

Tabla 28 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 8

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	3255,33	5984,93	3923,51	404,98	241,47	83,32	729,76
2	2997,41	4811,41	3622,12	636,51	250,69	114,25	1001,45
3	3974,78	6684,58	5021,63	806,88	251,31	273,19	1331,38
4	3186,81	6091,08	4133,74	286,64	256,73	175,19	718,57
5	3013,63	4429,94	3089,44	668,04	239,48	128,00	1035,52

Tabla 29 - Resultados alcanzados de *FP+OSEA+LB* en la instancia 9

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de OSEA	Tiempo de FP	Tiempo de OSEA	Tiempo de LB	Tiempo total
1	4338,65	8185,35	7118,58	2799,82	1079,28	7628,20	11507,30
2	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*
4	3866,36	7407,56	7221,48	2025,85	942,27	1906,77	4874,89
5	4280,29	8528,71	8300,08	4376,42	949,12	2254,43	7579,97

Resultados obtenidos a través del método FP+LB+OSEA

Tabla 30 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 1

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	7944,52	14110,2	7944,52	300,23	455,13	280,61	1035,96
2	7714,02	15206,3	7714,02	1615,39	142,57	308,06	2066,02
3	7491,19	14999,2	7491,19	2232,37	297,08	307,19	2836,64
4	9056,85	14525,8	9056,85	654,08	159,68	298,71	1112,46
5	8363,64	12978,6	8363,64	2824,60	219,76	303,84	3348,20
6	8125,06	14497,9	8125,06	563,63	1833,39	271,07	2668,09
7	9407,09	18107,7	9407,09	2658,89	1823,10	263,00	4744,99
8	7305,82	14520,4	7305,82	3412,78	78,26	290,38	3781,42
9	7693,10	14625,9	7693,10	3549,57	598,96	280,86	4429,39
10	8687,17	14496,8	8687,17	186,14	265,57	288,02	739,73

Tabla 31 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 2

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	2880,2	4406,94	2880,2	391,36	265,10	291,51	947,97
2	2489,32	4941,81	2489,32	376,61	163,88	317,40	857,89
3	2803,23	4546,55	2803,23	240,94	136,50	298,42	675,86
4	2763,73	4723,76	2763,73	519,40	118,57	293,71	931,67
5	2731,76	5163,95	2731,76	498,34	115,13	290,29	903,76

Tabla 32 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 3

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	3153,06	6359,56	3153,06	1566,72	1113,83	314,70	2995,25
2	2941,75	5308,71	2941,75	901,60	184,98	280,26	1366,84
3	3114,93	4871,97	3114,93	955,41	273,66	302,63	1531,69
4	3548,34	5280,83	3548,34	1288,66	530,55	286,13	2105,34
5	3375,69	4675,85	3375,69	1133,68	369,04	308,56	1811,28

Tabla 33 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 4

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	2254,41	3906,17	2254,41	857,03	64,35	289,22	1210,60
2	2987,87	5517,32	2987,87	454,25	284,92	297,44	1036,61
3	2262,48	5484,44	2262,48	506,76	108,34	264,45	879,56
4	2225,85	4854,97	2225,85	836,65	244,88	264,45	1345,98
5	2253,19	3922,69	2253,19	105,67	118,26	265,35	489,29

Tabla 34 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 5

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	3335,25	4452,75	3335,25	510,88	242,12	280,04	1033,04
2	3466,95	5330,24	3466,95	443,77	1362,35	311,02	2117,14
3	3214,53	4237,16	3214,53	446,62	208,65	269,45	924,72
4	3552,45	5732,52	3552,45	419,14	80,39	277,20	776,73
5	3331,02	5548,26	3331,02	631,90	2799,12	311,63	3742,65

Tabla 35 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 6

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	1840,43	2396,07	1840,43	7,76	2,77	2,50	13,02
2	1769,37	2194,20	1769,37	6,55	3,06	2,47	12,09
3	1976,65	2377,74	1976,65	8,68	2,68	2,32	13,68
4	1617,31	2400,72	1617,31	11,85	2,88	2,52	17,24
5	1760,11	2114,83	1760,11	10,23	2,65	2,31	15,19

Tabla 36 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 7

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	2795,84	4470,19	2795,84	33,12	429,30	45,10	507,52
2	2400,61	3453,57	2400,61	66,78	87,42	45,41	199,62
3	2662,41	4349,94	2662,41	32,53	186,27	43,12	261,91
4	2253,13	4952,32	2253,13	83,03	761,38	36,84	881,26
5	2225,68	4563,30	2225,68	76,23	256,73	45,01	377,97

Tabla 37 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 8

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	4119,96	7583,05	4119,96	4882,72	1070,83	988,83	6942,38
2	2946,36	5318,70	2946,36	683,89	288,59	251,93	1224,40
3	4137,36	8515,55	4137,36	4281,84	4206,32	1102,92	9591,08
4	3064,46	5488,05	3064,46	528,42	447,43	250,18	1226,03
5	3463,68	5040,52	3463,68	2167,91	264,34	251,38	2683,63

Tabla 38 - Resultados alcanzados de *FP+LB+OSEA* en la instancia 9

N° de prueba	Valor de función objetivo	Valor luego de FP	Valor luego de LB	Tiempo de FP	Tiempo de LB	Tiempo de OSEA	Tiempo total
1	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*

Apéndice V: Representación visual de los recorridos

En este apéndice se presenta la representación gráfica de los resultados obtenidos mediante el mejor resultado de la resolución del problema del PTIC. Se observa el plano del PTIC en donde se ubican las diferentes empresas clientes del servicio de recolección de residuos, el centro de clasificación (“depósito”) y las aristas que representan los recorridos que se hacen. Además, se identifica el orden del recorrido según el número que esté colocado al lado del nodo. Se distingue las empresas por las letras C (del inglés de *customer*) y a su lado el número en el que se atenderá. De los 21 días hábiles disponibles se hacen recorridos en 9 días de los cuales se presentan los diagramas correspondientes. Además, de corresponder el caso, se presentan los recorridos mejorados por conceptos de formulación de recorridos gráficamente.

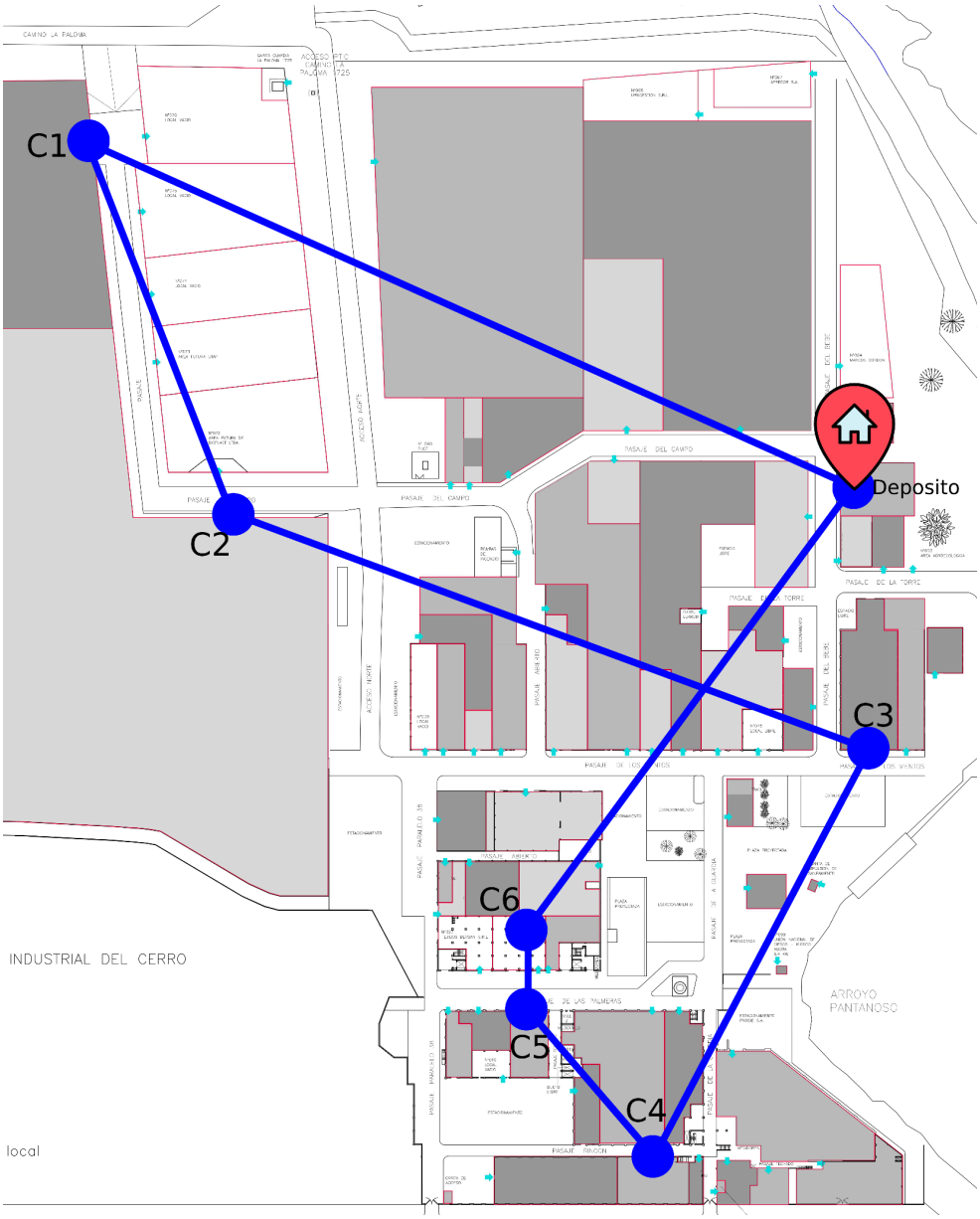

Figura 1 - Recorrido propuesto para el día 1

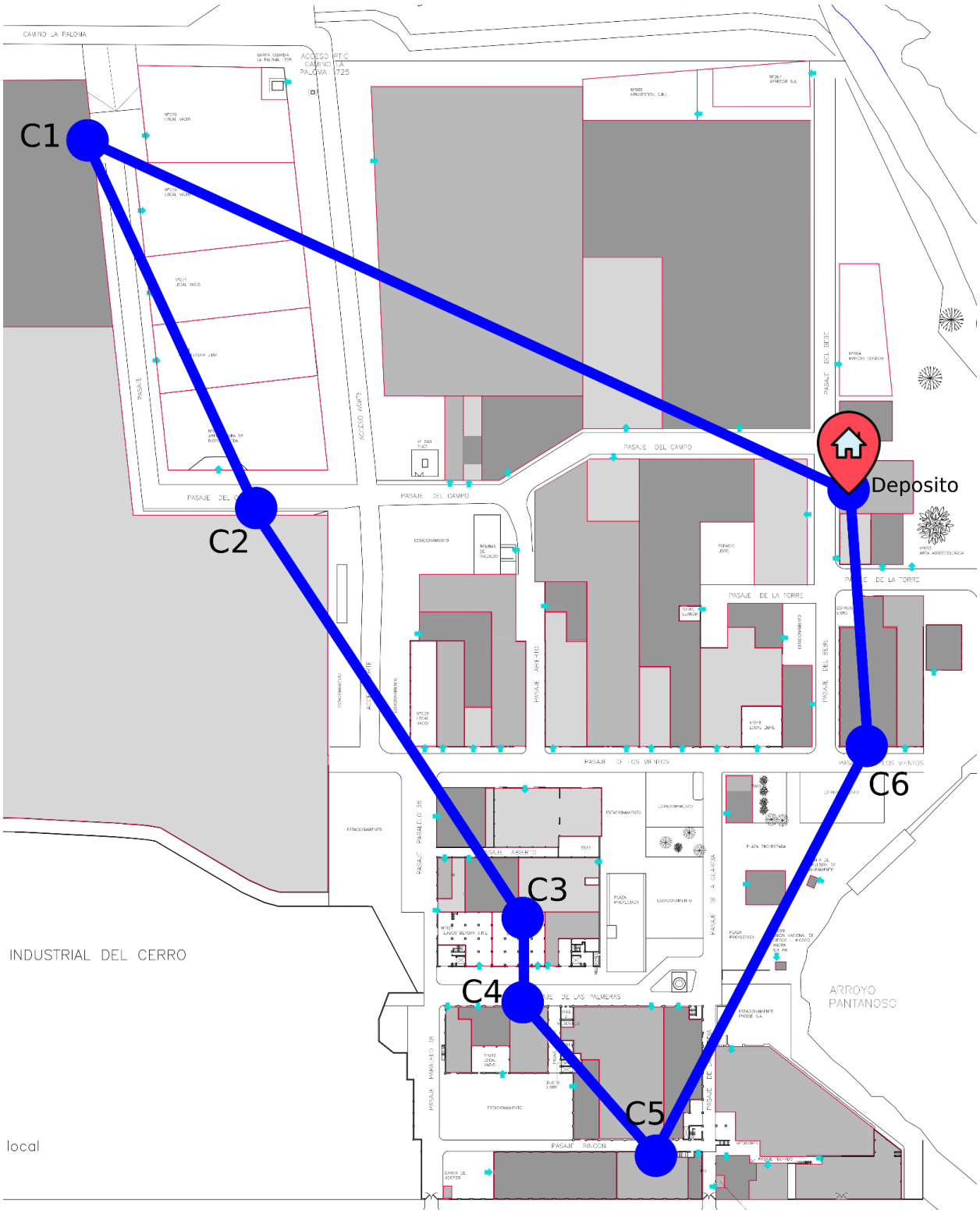

Figura 2 - Recorrido mejorado por métodos de refinamiento para el día 1

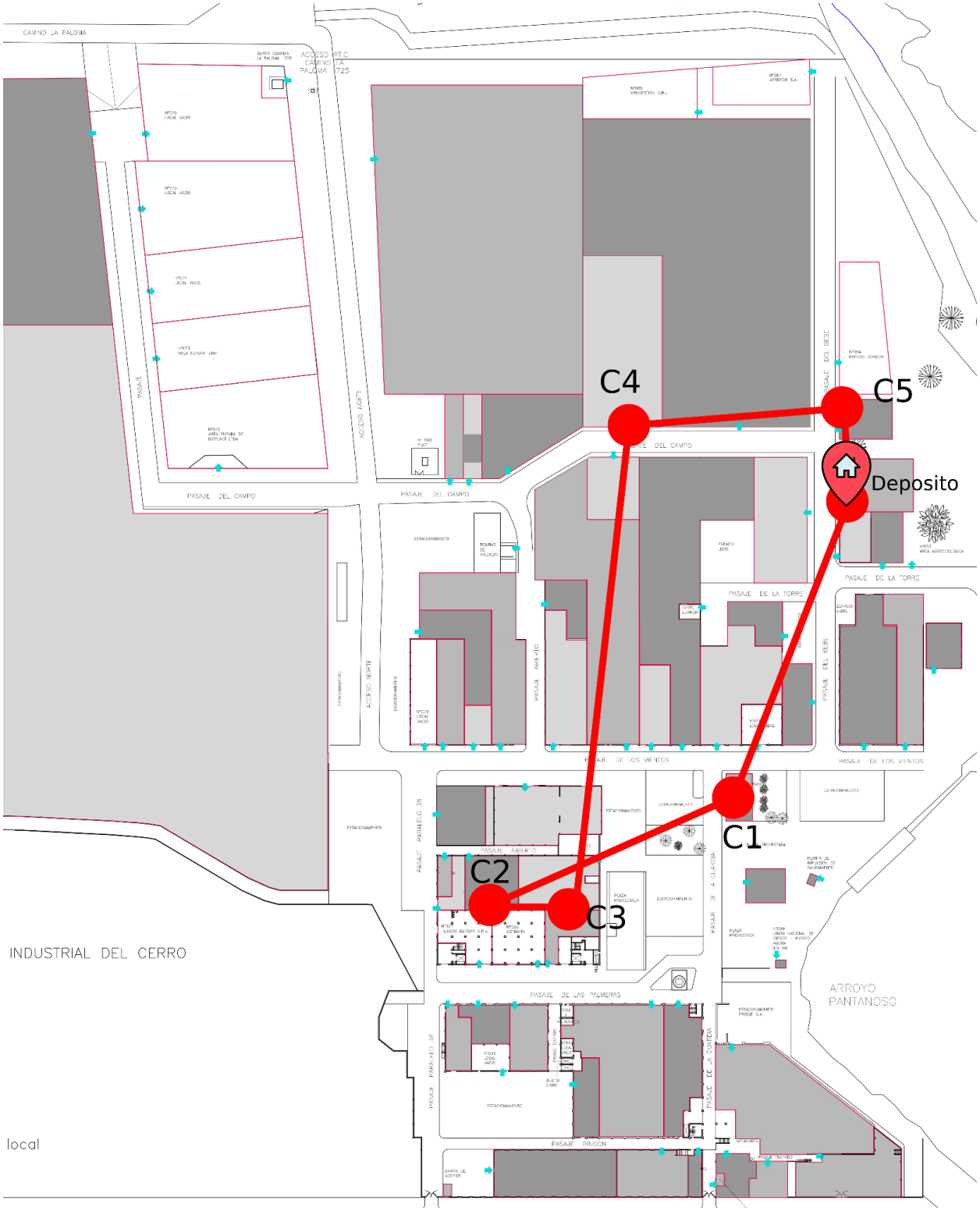

Figura 3 - Recorrido propuesto para el día 2

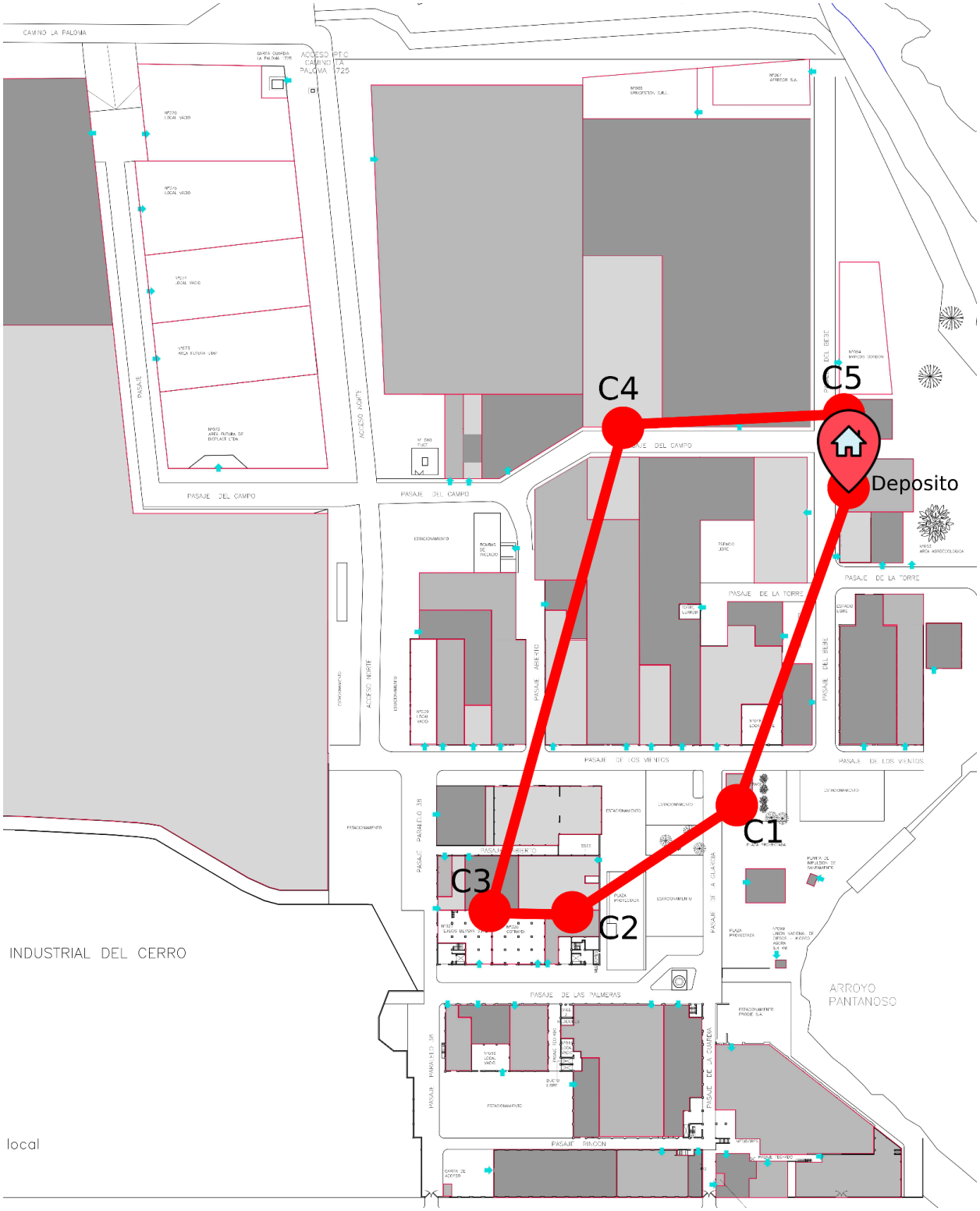

Figura 4 - Recorrido mejorado por métodos de refinamiento para el día 2

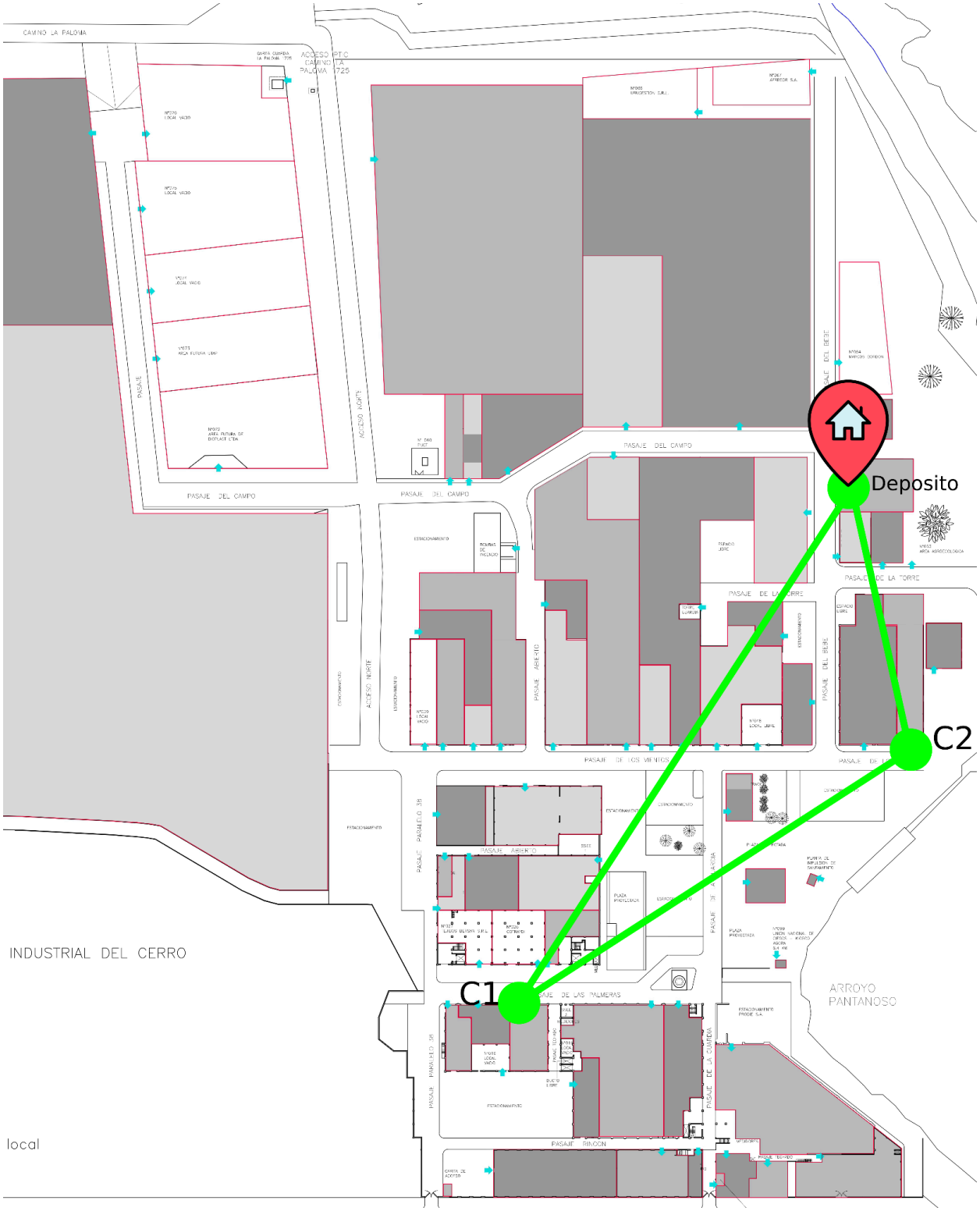

Figura 5 - Recorrido propuesto para el día 6

Figura 6 - Recorrido propuesto para el día 8

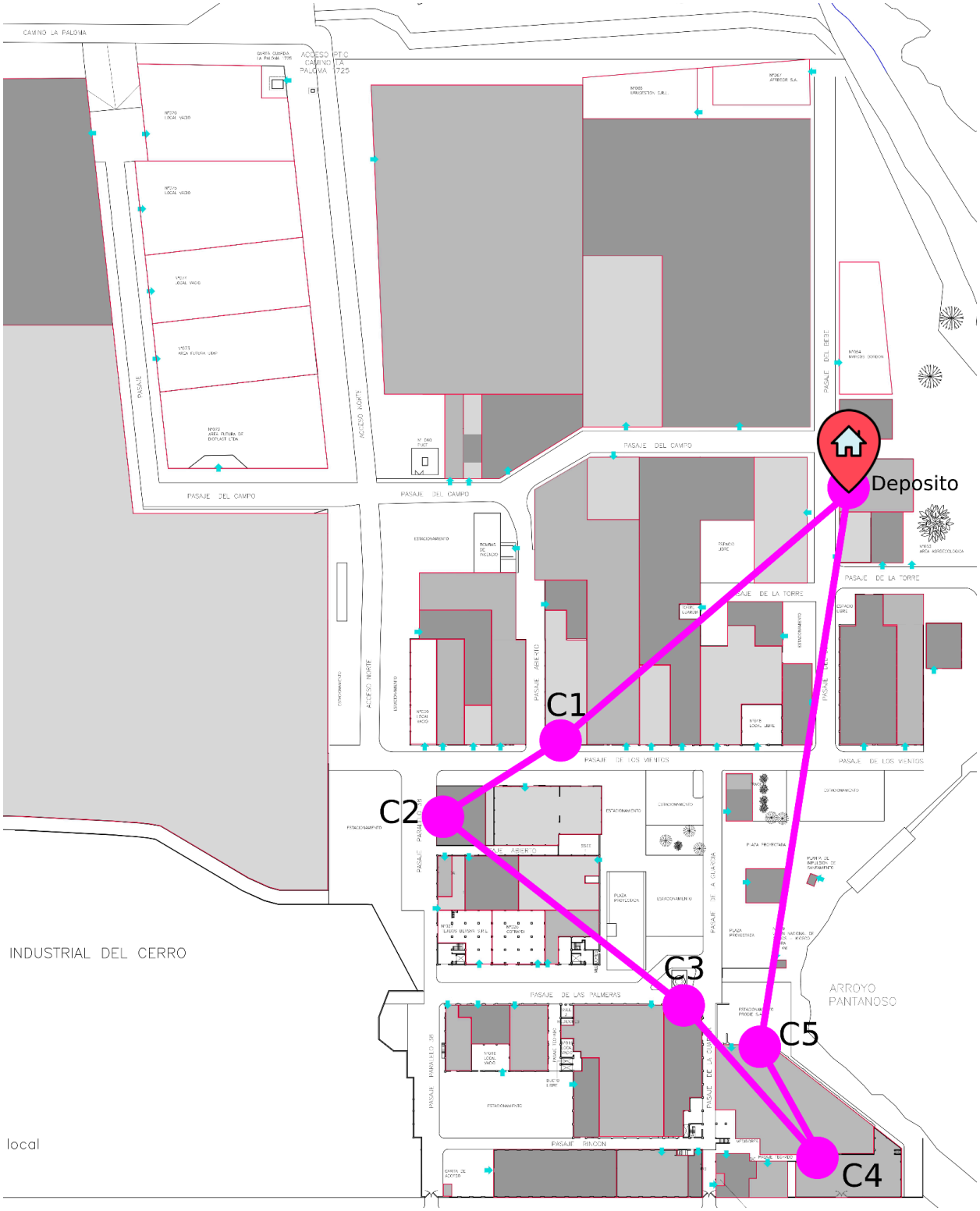

Figura 7 - Recorrido propuesto para el día 10

Figura 8 - Recorrido propuesto para el día 13

Figura 9 - Recorrido mejorado por métodos de refinamiento para el día 13

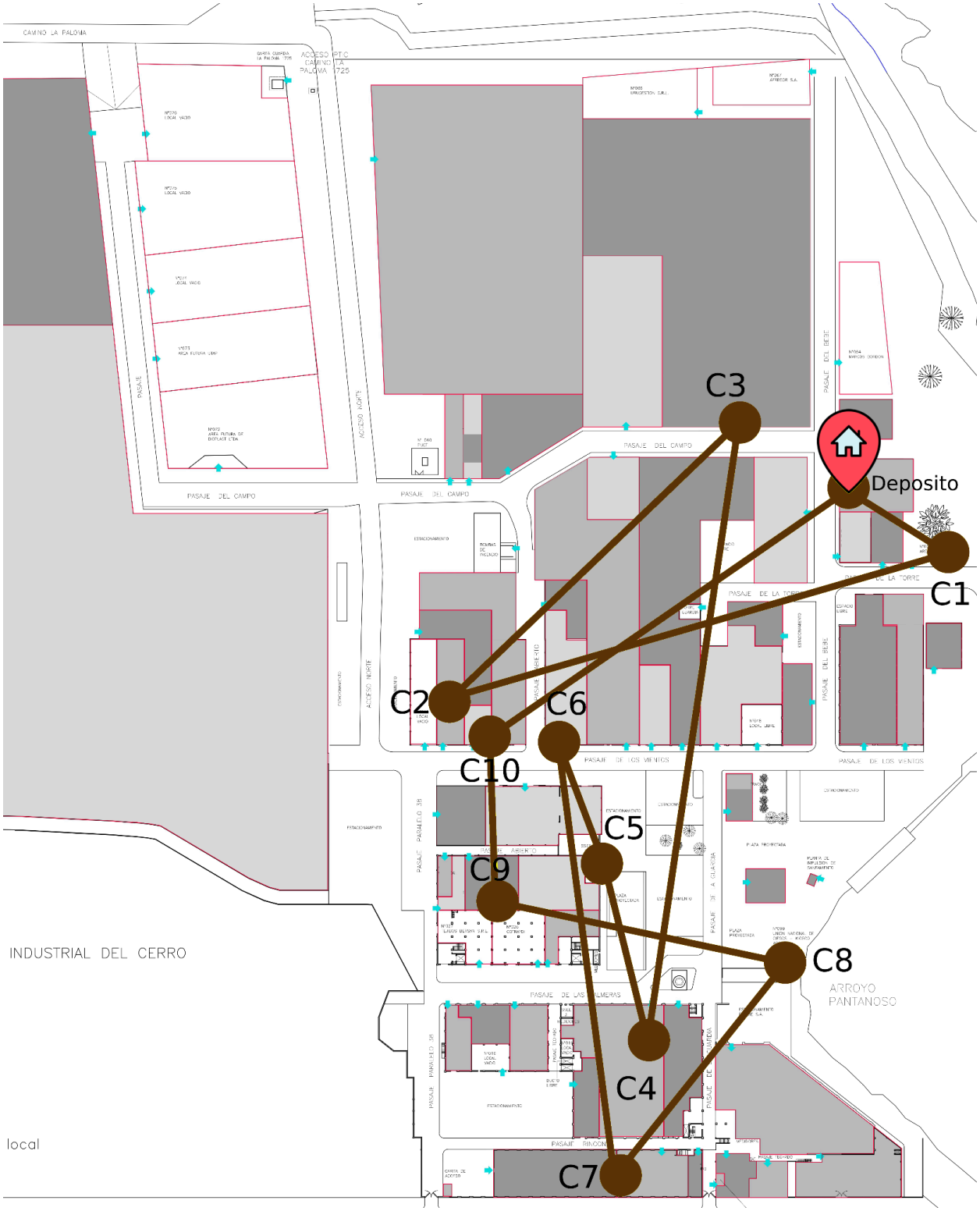

Figura 10 - Recorrido propuesto para el día 15

Figura 11 - Recorrido mejorada por métodos de refinamiento para el día 15

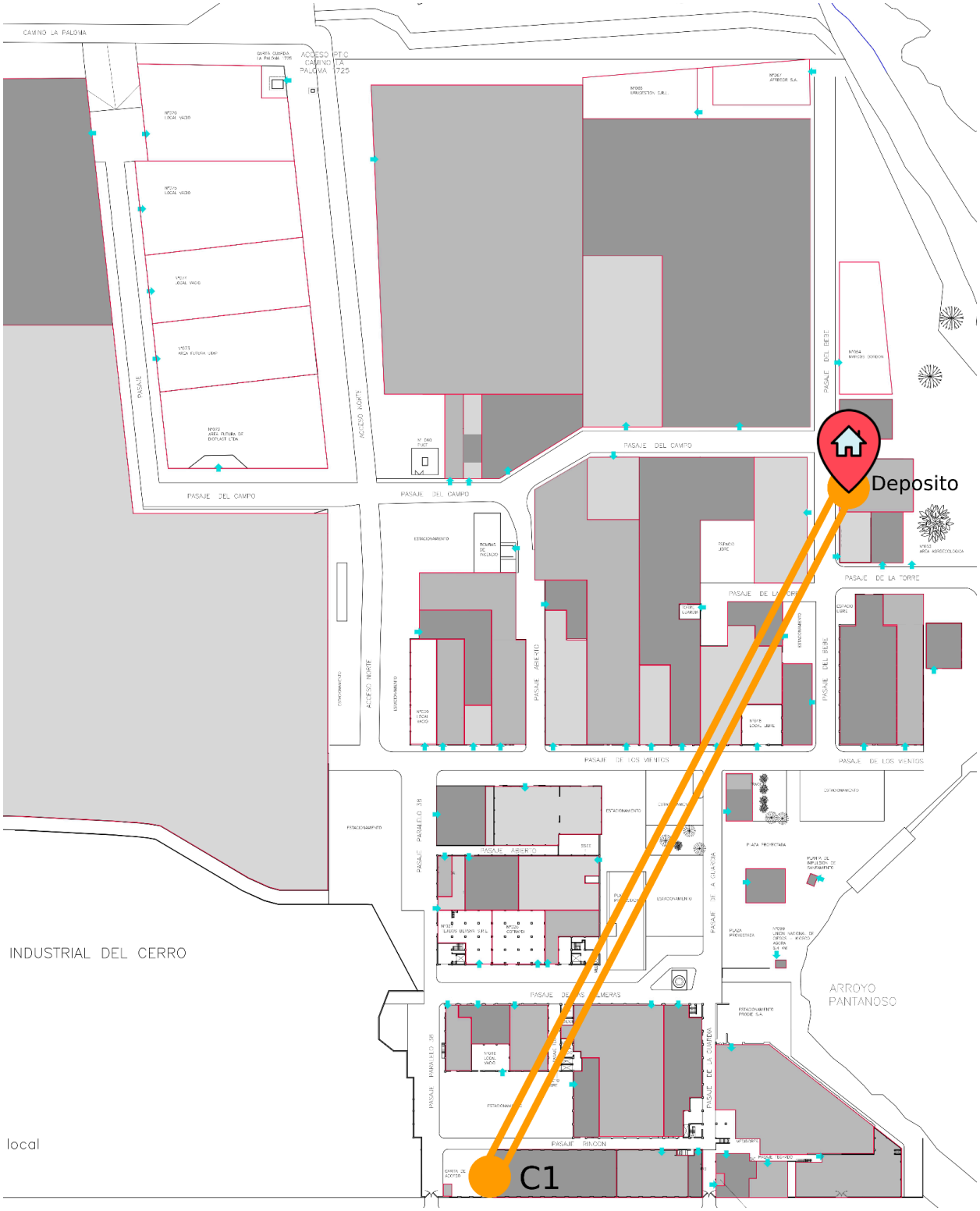

Figura 12 - Recorrido propuesto para el día 16

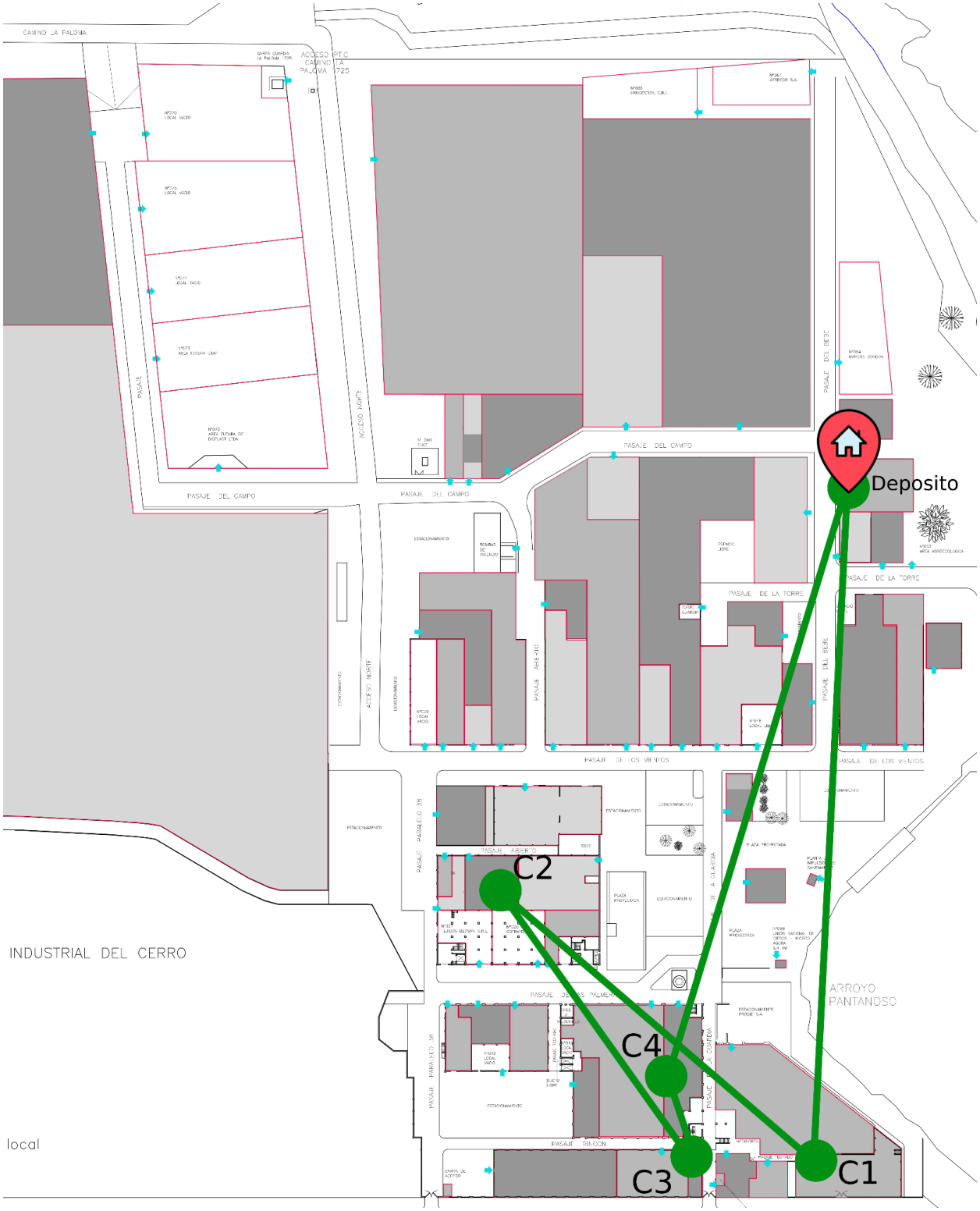

Figura 13 - Recorrido propuesto para el día 21

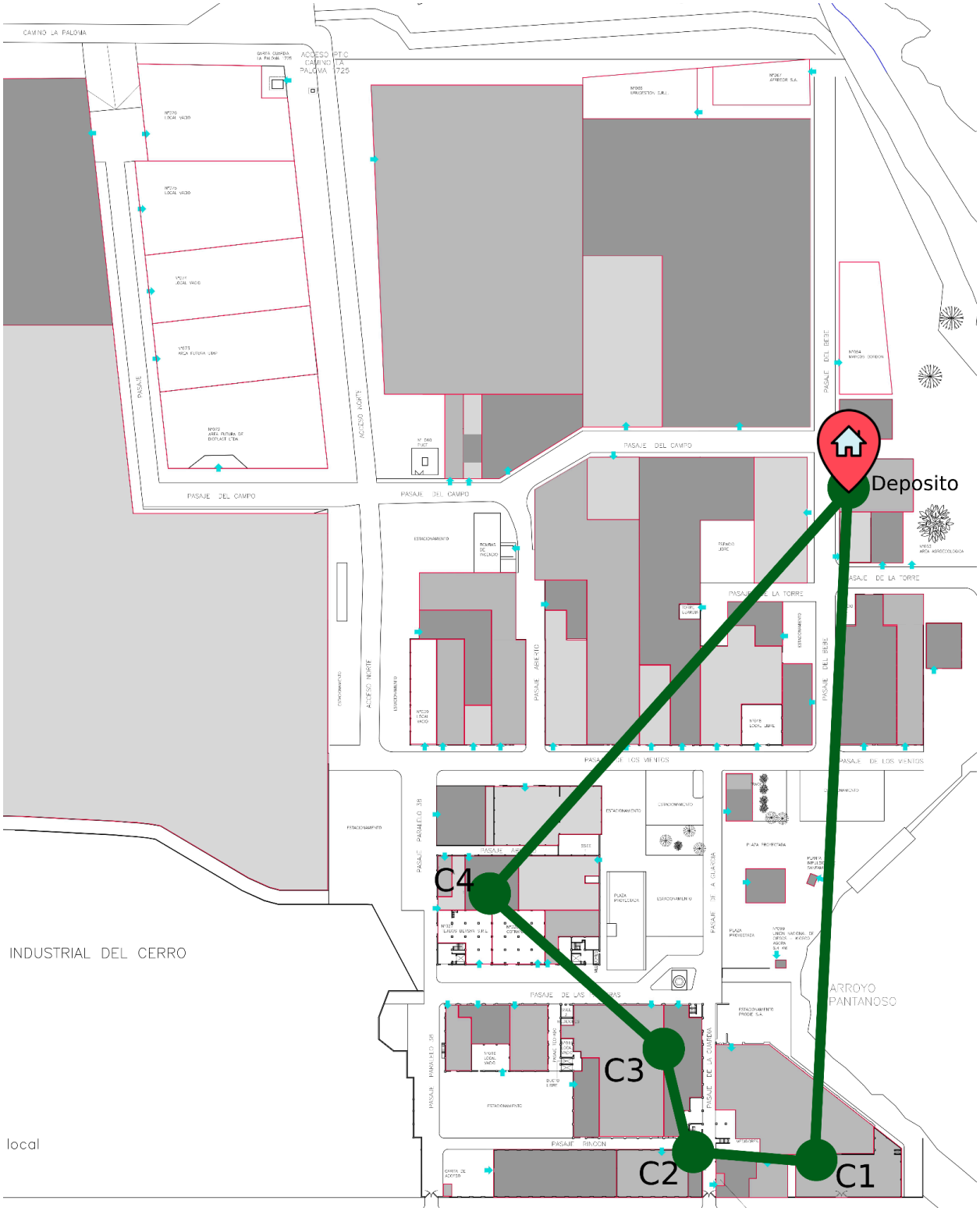

Figura 14 - Recorrido mejorado por métodos de refinamiento para el día 21