UNS

ESCUELA DE POSGRADO

EL PROBLEMA DEL RUTEO DE VEHÍCULOS MULTIDEPÓSITO ABORDAJE POR DIFERENTES HEURÍSTICAS

TESIS PARA OPTAR EL GRADO DE DOCTOR EN MATEMÁTICA

AUTOR:

Bach. PÉREZ CUPE, Rósulo Hilarión

ASESOR:

Dr. FLORES LUYO, Luis Ernesto

NUEVO CHIMBOTE - PERÚ
2022

UNS
ESCUELA DE POSGRADO

CONSTANCIA DE ASESORAMIENTO DE LA TESIS DOCTORAL

Yo, **LUIS ERNESTO FLORES LUYO**

mediante la presente certifico mi asesoramiento de la Tesis Doctoral titulada:

"EL PROBLEMA DEL RUTEO DE VEHÍCULOS MULTIDEPÓSITO ABORDAJE POR DIFERENTES HEURÍSTICAS"

elaborada por el magister **Rósulo Hilarión Pérez Cupe** para obtener el Grado Académico de Doctor en Matemática en la Escuela de Posgrado de la Universidad Nacional del Santa.

Nuevo Chimbote, 18 de abril del 2022.

.....

Dr. LUIS ERNESTO FLORES LUYO

ASESOR

UNS
ESCUELA DE POSGRADO

CONFORMIDAD DEL JURADO EVALUADOR

"EL PROBLEMA DEL RUTEO DE VEHÍCULOS MULTIDEPÓSITO ABORDAJE POR DIFERENTES HEURÍSTICAS"

TESIS PARA OPTAR EL GRADO DE DOCTOR EN MATEMÁTICA

Revisado y Aprobado por el Jurado Evaluador:

.....
DR. MILTON MILCIADES CORTEZ GUTIÉRREZ
PRESIDENTE

.....
DR. ERNESTO ANTONIO CEDRÓN LEÓN
SECRETARIO

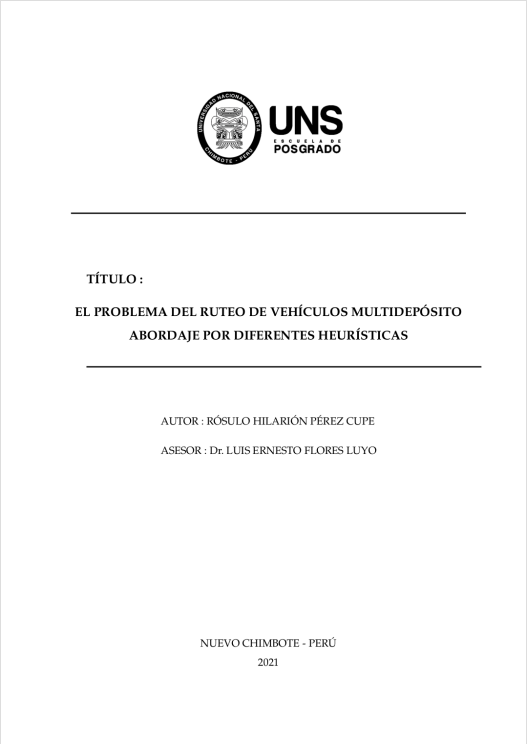
.....
DR. LUIS ERNESTO FLORES LUYO
VOCAL

Recibo digital

Este recibo confirma quesu trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega:	Rosulo Pérez
Título del ejercicio:	TESIS DOCTORAL
Título de la entrega:	Tesis doctoral
Nombre del archivo:	Tesis_de_Doctorado_Rosulo__2.pdf
Tamaño del archivo:	1.99M
Total páginas:	130
Total de palabras:	27,873
Total de caracteres:	135,478
Fecha de entrega:	14-abr.-2022 08:35p. m. (UTC-0500)
Identificador de la entre...	1811069886



UNS
ESCUELA DE POSGRADO

TÍTULO :

EL PROBLEMA DEL RUTEO DE VEHÍCULOS MULTIDEPÓSITO ABORDAJE POR DIFERENTES HEURÍSTICAS

AUTOR : RÓSULO HILARIÓN PÉREZ CUPE

ASESOR : Dr. LUIS ERNESTO FLORES LUYO

NUEVO CHIMBOTE - PERÚ
2021

DEDICATORIA

Este trabajo va dedicado con mucho cariño a las siguientes personas y lugares:

A mis padres Toribia y Doroteo

A mi esposa Liliana y mis hijos Emily y Paulo.

A mi tío Epifanio y mis sobrinos Maria Cielo y Brayan.

A mi linda tierra Andamarca Ayacucho y a la ciudad de Sao Paulo Brasil.

AGRADECIMIENTOS

Mi agradecimiento infinito a mis padres, Toribia Cupe y Doroteo Pérez por haberme inculcado los valores que permitieron cumplir mis metas.
A mi asesor Dr. Luis Ernesto Flores Luyo por su motivación en el tema y su apoyo incondicional en todas las etapas del trabajo.
Al joven estudiante Nelson Jorge Condor por su invalorable apoyo en el lenguaje de programación JULIA y los solvers Gurobi y Cplex.
A la Universidad Nacional de Ingeniería mi alma mater y centro laboral y por intermedio de ella a mis profesores y colegas de la Escuela Profesional de Matemática, quienes generan un ambiente agradable de trabajo e interacción orientado a alcanzar la calidad académica.
A la Escuela de Posgrado de la Universidad Nacional de Santa por la oportunidad y las condiciones favorables para el desarrollo del presente trabajo de Doctorado.

Índice general

Índice de cuadros

Índice de figuras

RESUMEN

El problema del ruteo de vehículos VRP (**V**ehicle **R**outing **P**roblem) es uno de los problemas de optimización mas importantes y desafiantes en el ámbito de la Investigación de Operaciones, fue planteado e introducido por Dantzing y Ramser en 1959, el cual consiste en construir un conjunto óptimo de rutas para una flota de vehículos que deberán satisfacer la demanda de un conjunto de clientes, el problema está catalogado como un problema combinatorio computacionalmente duro, ha sido intensamente estudiado en los últimos 50 años y continúa el interés dado que por su naturaleza $NP-Hard$ aún no se ha logrado una solución eficiente. La importancia de su estudio se debe al importante beneficio económico que se puede lograr al encontrar la ruta óptima. En la práctica han ido apareciendo diferentes necesidades que obligaron a formular ampliaciones o variantes del problema VRP, como por ejemplo el problema MDVRP (**M**ulti **D**epot **V**ehicle **R**outing **P**roblem). Cuando un problema combinatorio computacionalmente duro como el mencionado no se puede resolver de manera exacta se recurre a las soluciones aproximadas obtenidas por métodos heurísticos.

En el presente trabajo de investigación, estudiamos, formulamos y resolvemos (mediante la propuesta de heurísticas) el problema MDVRP, la solución exacta es tratada desde un punto de vista teórico utilizando la relajación de restricciones (relajación lagrangiana) y la optimización subgradiente. En tanto que la solución aproximada es tratada desde el punto de vista práctico mediante la formulación e implementación (en el lenguaje de programación JULIA 1.0.5) de las heurísticas de construcción y de mejora (aquí se encuentra el aporte del trabajo de investigación). En cuanto a las heurísticas de construcción se presentan dos propuestas de clusterización basadas en la ubicación geográfica de los clientes y depósitos y sus cercanías entre si, teniendo en cuenta además la limitación de capacidad de los vehículos; como un subproceso importante se ha formulado y resuelto un subproblema combinatorio de asignación de clústeres a depósitos pero de tamaño menor cuya solución exacta es posible determinar. En cuanto a las heurísticas de mejora se utilizaron las estrategias del vecino más cercano **(nearest neighbor)** y de separación **(split)**. Se presentan finalmente los resultados y las comparaciones respecto a la mejor solución conocida BKS (**B**est **K**now **S**olution) disponible en la literatura.

ABSTRACT

The vehicle routing problem VRP is one of the most important and challenging optimization problems in the field of Operations Research, it was raised and introduced by Dantzing and Ramser in 1959, which consists of building an optimal set of routes for a fleet of vehicles that should satisfy the demand of a set of customers, the problem is classified as a computationally hard combinatorial problem, it has been intensively studied in recent 50 years and the interest continues given that by its nature $NP - Hard$ an efficient solution has not yet been achieved. The importance of its study is due to the important economic benefit that can be achieved by finding the optimal route. In practical application, different needs have emerged that made it necessary to formulate extensions or variants of the VRP problem, such as the MDVRP problem (**M**ulti **D**epot **V**ehicle **R**outing **P**roblem). When a computationally hard combinatorial problem like the one mentioned cannot be solved exactly, the approximate solutions obtained by heuristic methods are used.

In the present research work, we study, formulate and solve (by proposing heuristics) the MDVRP problem, the exact solution is treated from a theoretical point of view using constraint relaxation (Lagrangian relaxation) and subgradient optimization. While the approximate solution is treated from the practical point of view through the formulation and implementation (in the JULIA 1.0.5 programming language) of the construction and improvement heuristics (here is the contribution of the research work). Regarding construction heuristics, two clustering proposals are presented based on the geographic location of customers and warehouses and their proximity to each other, also taking into account the limitation of vehicle capacity; as an important thread, a combinatorial subproblem of assigning clusters to buckets has been formulated and solved, but obviously smaller in size whose exact solution can be determined. Regarding the improvement heuristics, the strategies of the nearest neighbor and of separating were used. Finally, the results and comparisons are presented with respect to the best known BKS solution (**B**est **K**now **S**olution) available in the literature.

INTRODUCCIÓN

En el presente trabajo formulamos, planteamos y resolvemos (mediante heurísticas) el problema de aplicación llamado MDVRP (Ruteo de Vehículos Multi Depósito), al ser el problema mencionado de clase $NP - Hard$ la solución exacta para instancias de tamaño mayor o igual a 50 depósitos es ya impráctico en el sentido de que el tiempo utilizado sería excesivamente grande. Es por esta última razón que se buscan soluciones aproximadas mediante la implementación de dos heurísticas de construcción cuyos resultados serán mejoradas sistemáticamente, mediantes tres heurísticas de mejora que son aplicadas de manera articulada a la solución inicial obtenida, los resultados son mostrados en tablas de comparación y gráficos respectivos.

En el **capítulo 1** se presenta el problema de investigación que se desarrolla en el presente trabajo, asimismo se muestran los antecedentes del problema planteado y el desarrollo de las teorías o estado del arte que existe al momento de iniciar la investigación. Por otro lado se formula el problema, se delimitan el alcance del trabajo y se justifica la importancia del estudio de la investigación, finalmente se muestra el objetivo de la investigación.

En el **capítulo 2** se presentan los conceptos previos necesarios para el desarrollo del presente trabajo en su conjunto, se da una descripción breve acerca de los conceptos de complejidad de algoritmos, la clasificación de los algoritmos segun su orden de complejidad computacional asimismo la clasificación de los problemas computacionales en P y NP y finalmente una descripción del problema TSP, en cuanto a su formulación matemática y su complejidad asi como sus variantes.

En el **capítulo 3** se formula la hipótesis central del problema de investigación la cual deberá ser contrastada con los resultados obtenidos, se definen las variables e indicadores de la investigación asi como los métodos y el esquema de investigación. Se hace un listado cronológico de las actividades del proceso de investigación, mencionando como técnicas e instrumento de investigación a las estrategias teóricas para abordar la solución exacta del problema MDVRP, como son la relajación lagrangiana y la optimización subgradiente, desde el punto de vista teórico. El objetivo principal de la investigación son las heurísticas de construcción, llamadas heurísticas de clusterización se proponen tres heurísticas (por distancias(**HD**), por distancias y deman-

das(**HDD**) y por distancias, demandas y zonas(**HDDZ**)) que nos permitará construir una solución inicial factible, a continuación se proponen las heurísticas de mejoramiento (vecino cercano y splitt) que nos permitirán acercarnos hacia la solución exacta tanto como sea posible, teniendo como referencia previa a la mejor solución obtenida hasta el momento llamada **BKS**.

En el **capítulo 4** se presentan los resultados obtenidos como resultado de la implementación y ejecución en el lenguaje de programación JULIA 1.0.5 de las heurísticas de construcción y las sucesivas mejoras aplicadas a las soluciones iniciales, el reporte es mostrado en tablas completas donde se compara los resultados obtenidos por las heurísticas desarrolladas y la mejor solución conocida hasta el momento BKS, las corridas se realizaron para las 23 instancias existentes en la literatura desde p01 hasta p23. La discusión se genera en torno a la comparación de los resultados obtenidos y la satisfacción o cumplimiento de la hipótesis planteada.

En el **capítulo 5** se hace un listado de las conclusiones a las que se llega como resultado de la investigación asi como las recomendaciones para trabajos futuros basados en la presente investigación.

Capítulo 1

PROBLEMA DE INVESTIGACIÓN

En este capítulo, se presenta el problema de investigación que se desarrollará en el presente trabajo, asimismo se muestran los antecedentes del problema planteado y el desarrollo de las teorías o estado del arte que existe al momento de iniciar la investigación. Por otro lado se formula el problema, se delimitan el alcance del trabajo y se justifica la importancia del estudio de la investigación, finalmente se muestra el objetivo de la investigación.

1.1. Planteamiento y fundamentación del problema de investigación

En general el problema MDVRP consiste en la distribución de algún producto que se encuentra almacenado en cierta cantidad de depósitos y debe ser distribuido a determinado número de clientes cada uno de los cuales tiene una demanda conocida, la distribución se realiza mediante vehículos de capacidad limitada. Específicamente se tiene n depósitos en cada una de las cuales existe una flota de vehículos que trasladarán los productos a los m clientes, los clientes se agruparán de acuerdo a su ubicación geográfica respecto a los depósitos y serán atendidos por la flota de vehículos asignada al correspondiente depósito. Cada vehículo sale de un depósito visita una sola vez a cada cliente, satisface su demanda y regresa al mismo depósito de donde partió. Siendo el costo de transporte proporcional a la distancia, el objetivo será encontrar aquella ruta que minimiza la distancia total recorrida por todos los vehículos, formu-

lándose así un problema de programación lineal entera (más específicamente programación lineal binaria), el problema así formulado es de naturaleza $NP - Hard$, ésta es la razón por la cual encontrar una solución exacta es imposible por ello se requiere la formulación de heurísticas o metaheurísticas que se diseñarán en el desarrollo del presente trabajo, comparándo su desempeño con heurísticas ya existentes y de buen funcionamiento como son los algoritmos genéticos.

Por ejemplo si suponemos que una empresa cervecera tiene tres depósitos principales (tal como se muestra en la figura 1.1) desde los cuales debe distribuir su mercadería a 18 clientes (círculos numerados del 1 al 18 en cuya parte superior aparece la demanda de dicho cliente) y para ello dispone de una flota de 9 vehículos. Se requiere resolver el problema del ruteo de dichos vehículos, puede ser a nivel local o a nivel nacional. En general la aplicación de este modelo puede optimizar la operatividad de cualquier empresa dedicada al rubro mencionado.

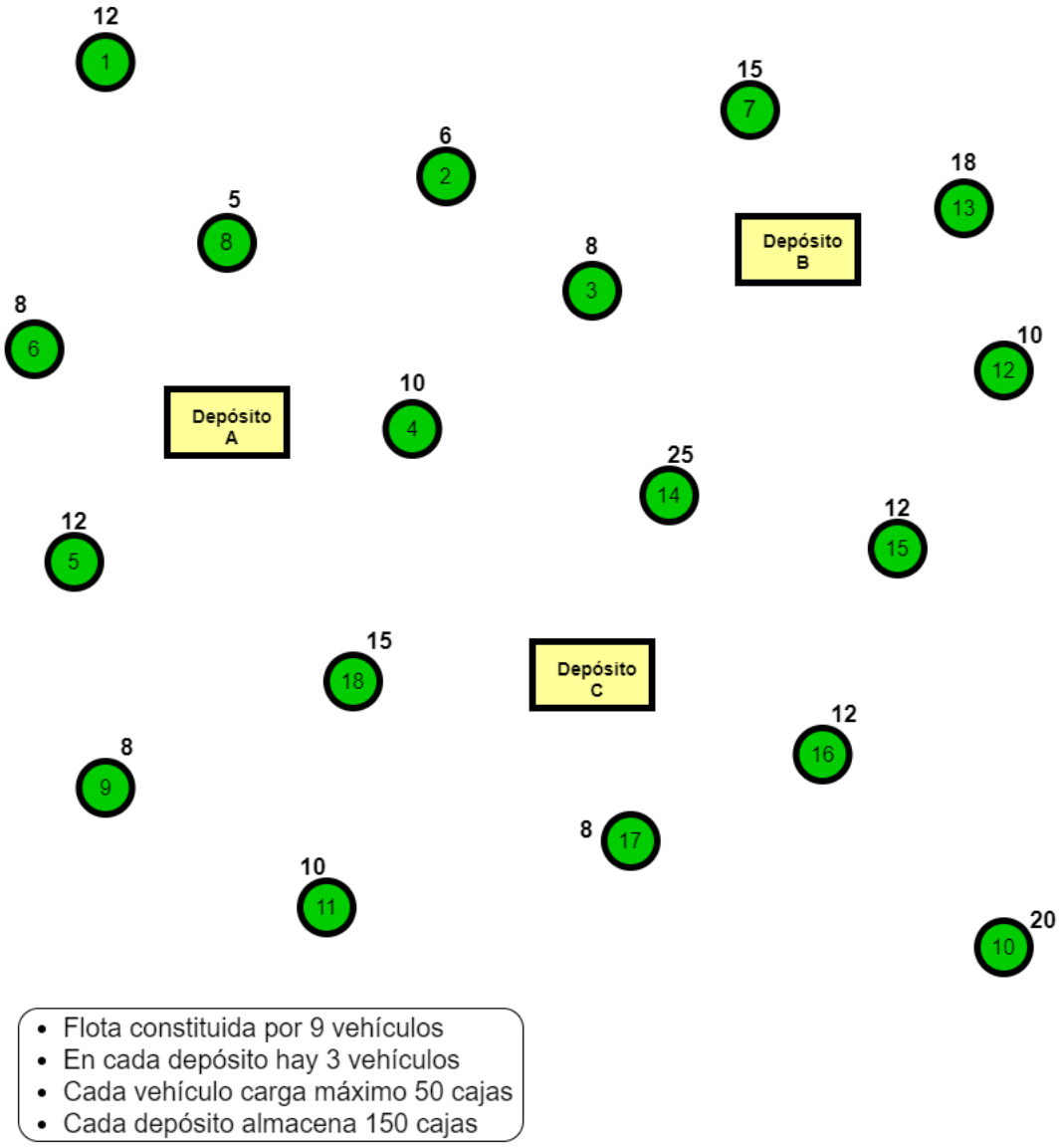


Figura 1.1: Una instancia del problema del ruteo de vehículos multidepósito MDRVP con 3 depósitos y 18 clientes.

Como parámetros a priori se tiene información respecto a las ubicaciones de clientes y depósitos, flota de vehículos, carga máxima de cada vehículo, demanda de cada cliente, capacidad máxima de cada depósito. Para efectos de cuantificación del costo, consideramos la distancia entre clientes y depósitos dado en kilómetros y consideramos un costo por kilómetro igual a 80 soles(por ejemplo puede considerarse consumo de combustible, mantenimiento del vehículo y otros).

La solución exacta se puede determinar utilizando solvers como son CPLEX versión 12.6.1 y GUROBI versión 9.1.0 o también programando el modelo en el lenguaje

de programación JULIA version 6.0.5, las técnicas que utilizan los solvers mencionados son ramificación y acotamiento, desigualdades válidas y relajación lagrangeana, cabe mencionar que es posible obtener soluciones exactas para tamaños de problema pequeños (menos de 50 depósitos) en tiempos razonablemente cortos, para tamaños mayores el tiempo podria ser de orden exponencial, tal como se desarrolla en el capítulo 2, correspondiente a la sección de análisis de complejidad de algoritmos.

La solución aproximada (vía heurísticas) pasa por tomar la decisión de qué almacenes atenderán a los clientes, esto se realiza en cuatro etapas: **agrupamiento(o clusterización), ruteo, programación y optimización**. En la fase de **ruteo** los clientes que serán atendidos por un mismo mismo depósito son asignados a varias rutas mediante el **método de ahorro** de Clarke and Wright como se puede ver en (Pichpibul and Kawtummachai, 2012, ichpibul and Kawtummachai, 2012), siendo cada ruta secuenciada en la fase de **programación**, el objetivo del ruteo es minimizar la cantidad de rutas sin violar las restricciones de capacidad y demanda de clientes. Dado que existen n depósitos la cantidad mínima de rutas es n, en el caso del ejemplo $n = 3$ rutas como mínimo. Por otro lado debido a que cada ruta será cubierta por un vehículo, una gran cantidad de rutas incrementa la cantidad de vehículos a utilizar, reduciendo así la calidad de la solución. Un mejor ruteo y programación pueden dar como resultado una menor distancia recorrida en la entrega, un menor tiempo empleado, un mayor nivel de eficiencia y un menor costo de entrega. En resumen el objetivo del problema MDVRP es minimizar la distancia total de entrega o el tiempo empleado en atender a todos los clientes (esto último implica una mayor satisfacción del cliente). Por otro lado menos vehículos significan un costo total menor, de tal manera que el objetivo podría ser también minimizar la cantidad de vehículos. Para una descripción mas clara de la formulación del problema consideremos el problema MDVRP con 3 depósitos y 18 clientes mostrado en la figura 1.1, para la cual se muestra una solución factible (mostrada en la figura 1.2)

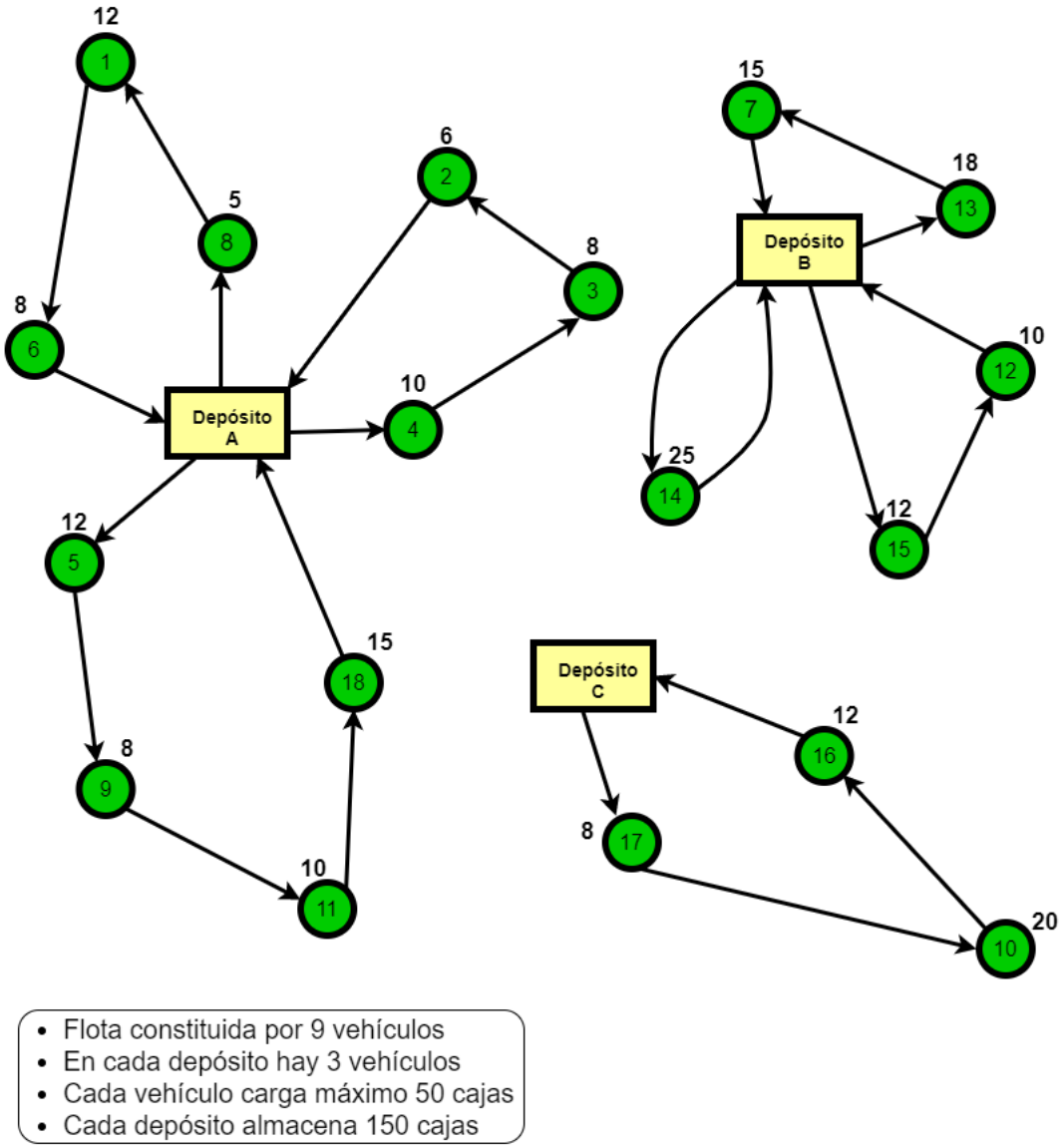


Figura 1.2: Una solución factible para el problema propuesto en la figura 1.1. Esta solución usa 7 vehículos y tiene una longitud total de recorrido de 59.5 kilómetros y en consecuencia un costo total igual a 4760 soles

El problema propuesto si bien es de naturaleza simple en su formulación, en cuanto a la cantidad de soluciones debe observarse que existen **muchas** otras soluciones factibles, la cantidad de soluciones mencionada en general es bastante grande y si cuantificamos en términos de costo computacional o tiempo empleado, éstas pueden ser de orden exponencial como será mostrado en el siguiente capítulo y sección correspondiente al análisis de algoritmos. Mostramos a continuación en la siguiente figura otra solución factible.

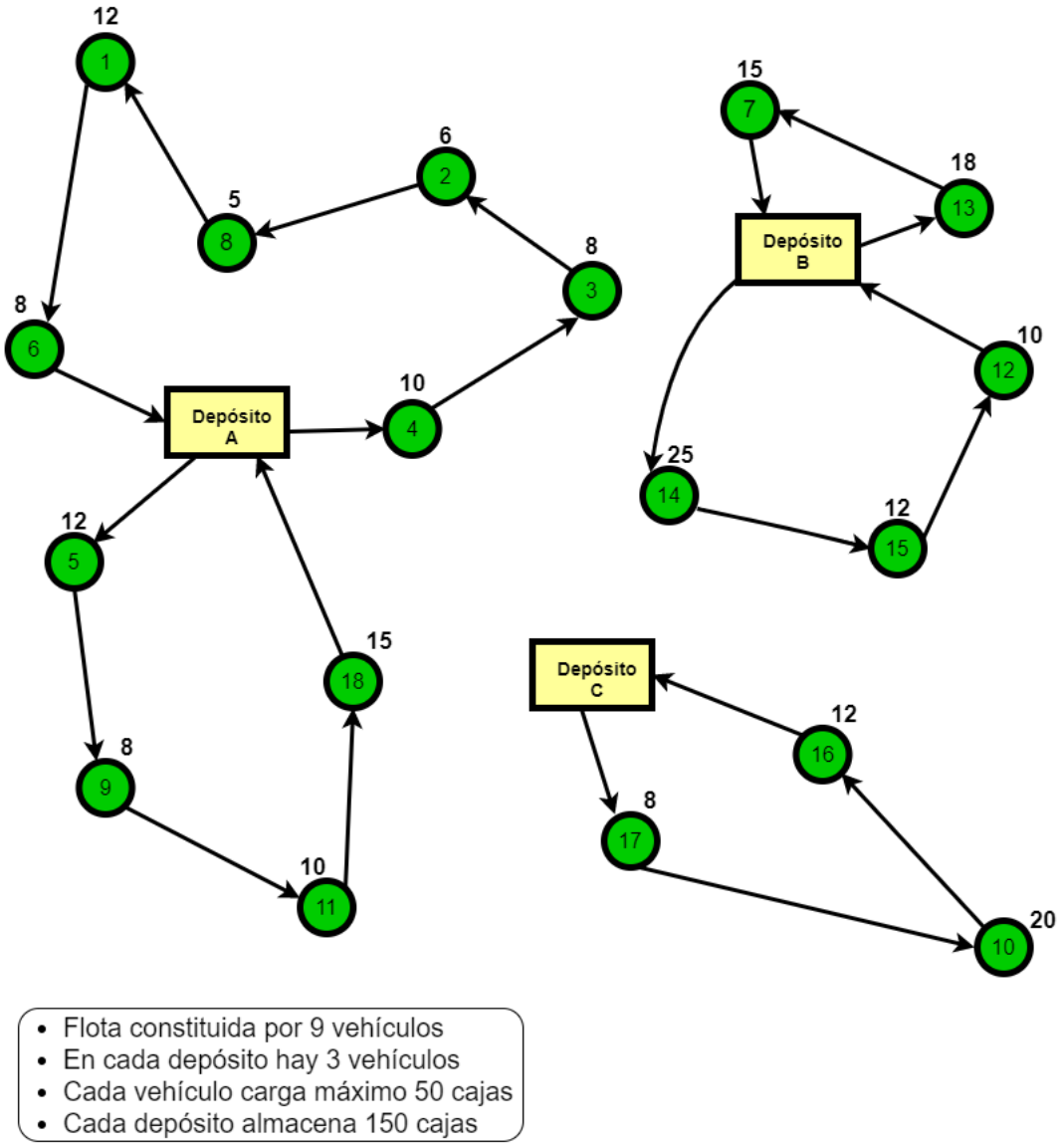


Figura 1.3: Otra solución factible para el mismo problema propuesto en la figura 1.1 que usa 5 vehículos y tiene una longitud total de recorrido de 52 kilómetros y en consecuencia un costo total igual a 4160 soles

1.2. Antecedentes de la investigación

1.2.1. El problema MDVRP

El problema de ruteo de vehículos VRP es una generalización del problema del agente viajero, aparece por primera vez en 1950 y desde entonces ha sido ampliamente estudiado, presentándose una serie de propuestas de mejoras de solución, el avance

tecnológico ha permitido avanzar en cuanto al tamaño del problema, cada solución propuesta mejora los tiempos de ejecución. Debido a su gran aplicabilidad a situaciones reales y retroalimentándose de ellas hay una serie de variantes del problema original, los cuales se mostrarán mas adelante.

El VRP surge de una manera muy natural en las operaciones de transporte, distribución y logística, dado que el transporte en cuaquier actividad económica es la parte que acarrea un buen porcentaje del costo total, su cuantificación es una tarea ineludible para tener una planificación correcta de la actividad. Siendo este problema uno de los problemas del campo de la programación lineal entera de la categoría $NP - Hard$ es decir es imposible resolverlo en tiempo polinomial en el tamaño de la entrada. El tiempo y esfuerzo computacional al resolver este problema aumenta exponencialmente respecto al tamaño del problema (esto es la cantidad de nodos visitados por los vehículos) para estos problemas hallar la solución exacta para tamaños reales es prácticamente imposible por ello se recurre a otras estrategias llamadas heurísticas que nos permiten encontrar buenas soluciones (cercanas al exacto) pero en un tiempo razonablemente corto. Las principales variantes del problema VRP, se mencionan a continuación:

- No se vuelve al depósito (Open VRP - OVRP)
- Existencia de varios depósitos de partida para abastecer a los clientes (MDVRP)
- Entrega de la mercadería en tiempos definidos llamados ventanas de tiempo (VRPTW)
- VRP con ventanas rígidas de tiempo (VRPTD)
- No se vuelve al depósito (open VRP- OVRP)
- Algunos parámetros (número de clientes, demandas, tiempos de servicio, etc) son aleatorios (stochastic VRP- SVRP)
- Los pedidos pueden ser llevados sólo en determinados días (periodic VRP- PVRP)

En el presente trabajo analizaremos **El problema del ruteo de vehículos multidepósito**, como lo mencionamos antes dada la característica $NP - Hard$ del problema

existen diferentes propuestas de solución cada una de las cuales mejoran el tiempo de ejecución, el objetivo nuestro es mejorar el tiempo de ejecución para instancias de tamaño razonablemente grandes, para ello haremos uso de heurísticas para la construcción de una solución aproximada. Una completa revisión del estado del arte de la investigación la encontramos en (Lüer et al., 2009, üer et al., 2009)

El presente trabajo se centra en el problema MDVRP (Multi Depot Vehicle Routing Problem), la solución se basa en los métodos heurísticos. La investigación sobre los problemas MDVRP aún es limitada, por ejemplo aún no se tiene resultados para una flota mixta, también para efectos prácticos se considera que la capacidad de cada depósito es ilimitada.

El problema VRP definido en 1958 por Dantzig y Ramser, consiste en dado un conjunto de clientes dispersos geográficamente, un depósito y una flota de vehículos, determinar un conjunto de rutas de costo mínimo que comiencen y terminen en el mismo depósito, para que los vehículos visiten a los clientes, satisfagan sus demandas a un costo óptimo.

Debido a la característica $NP - Hard$ del problema, se han desarrollado a lo largo de las últimas décadas una serie de heurísticas y metaheurísticas con el fin de obtener buenas soluciones en menor tiempo que los métodos exactos.

La variante con más relevancia para el presente estado del arte es la del Problema de Ruteo de Vehículos con Múltiples Depósitos, MDVRP por su nombre en inglés, Multi-Depot Vehicle Routing Problem. En este caso, el problema cuenta con la posibilidad de utilizar múltiples depósitos para atender la demanda de sus clientes.

La literatura clasifica esta variante como un problema $NP - Hard$ al igual que lo es el problema original VRP, la tasa de artículos publicados por año sobre MDVRP en el período 1984-2000 fue de 1.12, mientras que para el período de 2011-2014 había aumentado a 12.75. Esto se debe a que diversas aplicaciones en la vida cotidiana, como por ejemplo la distribución de gas, de productos químicos, de alimentos y de bebidas entre otras han motivado la investigación acerca de métodos para resolver el problema de MDVRP debido a la considerable importancia económica que tiene en esos casos.

1.2.2. Variantes del problema MDVRP

Una variante importante del MDVRP es el modelo Multi Depot Fleet Size and Mix Vehicle Routing Problem (MDFSMVRP), es decir el problema de múltiples depósitos con diferentes tipo de vehículos o flota de vehículos para satisfacer la demanda de los clientes con demanda desconocida. En esta variante MDFSMVRP se combina tres decisiones de manera simultánea: seleccionando la cantidad de vehículos de cada tipo, planificando rutas de vehículos y asignando rutas a los depósitos. Cada vehículo se caracteriza por tener asociado a sí un costo fijo y variable proporcional a la distancia recorrida, la cantidad de vehículos de cada tipo se asume que es ilimitado. La optimización simultánea de la mejor composición de la flota, la mejor ruta para los vehículos y la elección del depósito cambia sustancialente el problema. Se debe designar un conjunto de rutas de vehículos empezando y terminando en el mismo depósito visitando cada cliente una sola vez y respetando la capacidad de los vehículos, minimizando el costo total.

La literatura para tratar explícitamente el MDFSMVRP no está aún bien desarrollada como los problemas clásicos o mas simples como son VRP y FSMVRP, existen cuatro trabajos centrados en el problema MDFSMVRP los cuales pasamos a mencionar:

- (Salhi and Sari, 1997, alhi and Sari, 1997) proponen una heurística multipropósito basada en la integralidad, su método se basa en cambiarse a un entorno más adecuado para la optimización.
- (Salhi et al., 2014, alhi et al., 2014) propone una progamación lineal entera mixta para formular el problema y un conjunto de desigualdades válidas.
- (Llanes-Font et al., 2014, lanes-Font et al., 2014) propone un algoritmo unificado para resolver diferentes tipos de problemas MDVRP con y sin flota mixta.
- (Mancini, 2016, ancini, 2016) aborda una nueva variante de MDFSMVRP con múltiples periodos y diferentes niveles de incompatibilidad entre clientes y vehículos.

En la siguiente subsección describimos la formulación matemática de una variante del problema MDVRP la cual consiste en considerar una flota de vehículos mixta (es

decir vehículos con diferentes capacidades máximas) y la función objetivo considera esta vez dos términos, una que cuantifica el costo fijo asociado a los vehículos y la otra el costo variable asociada a las rutas. En términos del estado del arte es la variante que mas generalizaciones considera.

1.2.3. La variante MDFSMVRP

La variante del problema MDVRP denominado MDFSMVRP (Multi Depot Fleet Size and Mix Vehicle Routing Problem) considera tamaño de flota mixta y es formalmente definido sobre un grafo dirigido $G = (\mathcal{V}, \mathcal{A})$ donde $\mathcal{V}$ es el conjunto de vértices y $\mathcal{A}$ conjunto de arcos, siendo $\mathcal{V} = \mathcal{V}_d \cup \mathcal{V}_c$ con $\mathcal{V}_d = \{1; 2; \cdots ; m\}$ representa el conjunto de m depósitos y $\mathcal{V}_c = \{m+1; m+2; \cdots ; m+n\}$ representa a los n clientes. Cada cliente $i \in \mathcal{V}_c$ está asociado a una demanda no negativa q_i, mientras que $q_i = 0 \quad \forall i \in \mathcal{V}_d$. La distancia entre los nodos $i \in \mathcal{V}$ y $j \in \mathcal{V}$ es β_{ij}, en lo que sigue usaremos los términos de costo y distancia indistintamente dado que son proporcionales. En general los lazos $(i; i)$ no están permitidos para efectos prácticos $\beta_{ii} = +\infty$ asimismo $\beta_{ij} = +\infty \qquad \forall i, j \in \mathcal{V}_d (i \neq j)$. El grafo G es completo, finalmente $\mathcal{A} = \{(i, j) : i, j \in V \quad i \neq j\}$

La flota heterogénea de vehículos será representado por $\mathcal{K} = \{1; 2; \cdots ; K\}$ con diferentes capacidades y ubicados en múltiples depósitos $d \in \mathcal{V}_d$, cada tipo de vehículo $k \in \mathcal{K}$ está asociado con una capacidad Q^k, un costo fijo F^k y un costo variable α^k por unidad de distancia, cada vehículo debe empezar su recorrido en un depósito y luego de visitar por única vez a determinados clientes debe retornar al depósito de donde partió. También denotamos por $\mathcal{H}$ a la flota completa de vehículos incluyendo las n copias de cada tipo de vehículo k, cumpliéndose $|\mathcal{H}| = nK$.

Cada vehículo tiene un índice $t \in \mathcal{H}$ que representa su número, cada vehículo $t \in \mathcal{H}$ tiene una capacidad Q^t, un costo fijo F^t y un costo variable α^t

La solución del problema será determinado minimizando el costo total tal que cada ruta empezará y culminará en el mismo depósito, cada cliente será visitado exactamente una sola vez y la demanda total de cada ruta no exceda la capacidad del vehículo seleccionado.

1.2.4. Formulación explícita o modelo matemático

Para el modelo matemático consideramos las variables x_{ij}^{td} que indica el número de veces 0; 1 o 2 que el arco (i,j) donde $i > j$ es usado en la solución, es decir

$$x_{ij}^{td} = \begin{cases} 0 & ; \text{ para cualquier otro caso} \\ 1 & ; \text{ si el arco } (i,j) \text{ es utilizado por el vehículo } t \text{ que salió del depósito } d \\ 2 & ; \text{ si se hace un viaje de ida y vuelta al cliente } j \end{cases}$$

asimismo consideramos las variables binarias y_i^{td} definido mediante:

$$y_i^{td} = \begin{cases} 0 & ; \text{ para cualquier otro caso} \\ 1 & ; \text{ si el nodo } i \text{ es visitado por el vehículo } t \text{ que salió del depósito } d \end{cases}$$

notar que t se refiere al índice del vehículo y no al tipo del vehículo dado que todos los vehículos disponibles han sido considerados en el modelo. Entonces el modelo queda formulado en los siguientes términos:

$$\text{mín} \left\{ \sum_{i \in \mathcal{V}_d} \sum_{t \in \mathcal{H}} \sum_{d \in \mathcal{V}_d} F^t y_i^{td} + \sum_{(i,j) \in \mathcal{E}} \sum_{t \in \mathcal{H}} \sum_{d \in \mathcal{V}_d} \alpha^t \beta_{ij} x_{ij}^{td} \right\} \tag{1.1}$$

sujeta a las restricciones:

$$\sum_{t \in \mathcal{H}} \sum_{d \in \mathcal{V}_d} y_i^{td} = 1 \qquad \forall i \in \mathcal{V}_c \tag{1.2}$$

$$\sum_{j \in \mathcal{V}(i>j)} x_{ij}^{td} + \sum_{j \in \mathcal{V}(i<j)} x_{ij}^{td} = 2y_i^{td} \qquad \forall i \in \mathcal{V}, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.3}$$

$$y_i^{td} \leq y_d^{td} \qquad \forall i \in \mathcal{V}_c, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.4}$$

$$y_d^{td} \leq \sum_{(i,j) \in \mathcal{E}} x_{ij}^{td} \qquad \forall t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.5}$$

$$2y_d^{td} \leq \sum_{i \in \mathcal{V}_c} x_{id}^{td} \qquad \forall t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.6}$$

$$\sum_{i \in \mathcal{Z}} \sum_{j \in \mathcal{Z}(i>j)} x_{ij}^{td} \leq \sum_{i \in \mathcal{Z}} y_i^{td} - y_z^{td} \qquad \forall \mathcal{Z} \subseteq \mathcal{V}_c, |\mathcal{Z}| \geq 2, z \in \mathcal{Z}, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.7}$$

$$\sum_{i \in \mathcal{V}_c} q_i y_i^{td} \leq Q^t \qquad t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.8}$$

$$x_{ij}^{td} \in \{0; 1\} \qquad (i, j) \in \mathcal{E}, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.9}$$

$$x_{ij}^{td} \in \{0; 1; 2\} \qquad i \in \mathcal{V}_c, j \in \mathcal{V}_d, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.10}$$

$$y_i^{td} \in \{0; 1\} \qquad i \in \mathcal{V}, t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.11}$$

Respecto al modelo planteado tenemos las siguientes precisiones:

- La función objetivo que se muestra en (1.1) minimiza el costo total, compuesto del costo fijo asociado a los vehículos y el costo variable asociado a las rutas.
- Las restricciones (1.2) indican que todos los clientes serán visitados exactamente una sola vez.
- Las restricciones (1.3) son respecto a los grados de los vértices del grafo.
- Las restricciones (1.4) imponen de que si un cliente es atendido por el vehículo t que partió del depósito d, entonces debe volver al depósito de donde partió.
- Las restricciones (1.5) y (1.6) relacionan los dos tipos de variables del modelo, ellos se aseguran de que si el vehículo t del depósito d es utilizado entonces al menos un cliente i debe ser visitado por este vehículo.
- Las restricciones (1.7) prohiben la existencia de subrutas.
- Las restricciones (1.8) imponen a los vehículos restricciones de capacidad.
- Las restricciones (1.9), (1.10) y (1.11) imponen el dominio de existencia de las variables del modelo x e y.

Esta formulación contiene $\mathcal{O}(2^n)$ restricciones de eliminación de subrutas, la cual crece exponencialmente con n. Esta es una formulación larga que depende fuertemente del número de vehículos disponibles.

El modelo representa suficientemente al MDFSMVRP; sin embargo podemos agregar algunas desigualdades válidas y retirar algunas restricciones para fortalecerlo. La restricción dada en (1.12)

$$\sum_{j \in \mathcal{V}_c} x_{jd}^{td} \leq 2 \qquad t \in \mathcal{H}, d \in \mathcal{V}_d \tag{1.12}$$

hacen que se cumplan las restricciones respecto al uso del vehículo, exactamente indica que cada vehículo t del depósito d realizará exactamente un solo viaje.

Para evitar simetrias debido a la presencia de vehículos idénticos en cada depósito, se introduce las restricciones de rompimiento de simetría del vehículo. Observe que las restricciones (1.13) y (1.14) son válidas solo si la flota es homogénea. Se define el conjunto $\mathcal{H}^k \subset \mathcal{H}$ que contiene solo los vehículos homogeneos del tipo k. De esta manera las restricciones (1.13) indican que el vehículo t puede despachar solo si el vehículo $t-1$ ya despachó. Las restricciones (1.14) clasifican vehículos idénticos de acuerdo con el índice de clientes, estas restricciones están definidas para cada depósito.

$$y_d^{td} \leq y_d^{(t-1)d} \qquad t \in \mathcal{H}^k \setminus \{\mathcal{H}_1^k\}, \quad \mathcal{H}^k \subset \mathcal{H}, k \in \mathcal{K}; d \in \mathcal{V}_d \tag{1.13}$$

$$y_i^{td} \leq \sum_{j \in \mathcal{V}_c, j<i} \sum_{h \in \mathcal{V}_d} y_j^{(t-1)h} \qquad i \in \mathcal{V}_c, t \in \mathcal{H}^k \setminus \{\mathcal{H}_1^k\}, \quad \mathcal{H}^k \subset \mathcal{H}, k \in \mathcal{K}; d \in \mathcal{V}_d \tag{1.14}$$

donde $\mathcal{H}_1^k$ es el primer elemento de $\mathcal{H}^k$.

En términos del estado del arte del problema de investigación lo descrito anteriormente es lo que existe en la literatura, nuestro objetivo será en base a los que se tiene, proponer heurísticas de construcción que logren encontrar soluciones buenas que incluso podrían mejorarse con las heurísticas de mejora.

1.3. Formulación del problema de investigación

Presentamos a continuación el modelo matemático para la solución del problema planteado, en caso de obtener dicha solución en tiempo razonablemente corto llamaremos a dicha solución exacta, la cual nos servirá para evaluar el desempeño

de nuestras heurísticas a desarrollar y a partir de ésta información decidiremos o postularemos que la heurística es buena o no. Para describir el modelo matemático consideremos los siguientes conjuntos:

I : Depósitos ($i \in I$) $\qquad$ J : Clientes ($j \in J$) $\qquad$ K : Vehículos ($k \in K$)

Debe observarse que los conjuntos I, J y K son subconjuntos finitos de $\mathbb{N}$.

Asimismo consideremos los siguientes parámetros: (los cuales son conocidos a priori)

N : cantidad de vehículos

C_{ij} : distancia entre los puntos i y j $\qquad$ $i, j \in I \cup J$

V_i : capacidad máxima en el depósito i

d_j : demanda del cliente j

Q_k : capacidad máxima del vehículo (o ruta) k

Parte importante del modelo es la definición de las variables de decisión:

$$x_{ijk} = \begin{cases} 1 \quad ; & \text{si el cliente } i \text{ precede inmediatamente al cliente } j \text{ en la ruta } k \\ 0 \quad ; & \text{en otro caso} \end{cases}$$

$$z_{ij} = \begin{cases} 1 \quad ; & \text{si el cliente } j \text{ es asignado al depósito } i \\ 0 \quad ; & \text{en otro caso} \end{cases}$$

Asimismo consideramos adicionalmente la variable auxiliar U_{lk} : ($l \in J$ y $k \in K$) usado para denotar las restricciones de eliminación de subrutas en la ruta k. La función objetivo de nuestro modelo matemático consistirá en minimizar la distancia total de recorrido de todos los vehículos, es decir

$$\text{mín} \left\{ \sum_{i \in I \cup J} \sum_{j \in I \cup J} \sum_{k \in K} C_{ij} x_{ijk} \right\}$$

sujeta a las Restricciones:

$$\sum_{k \in K} \sum_{i \in I \cup J} x_{ijk} = 1 \quad \forall\, j \in J \tag{1.15}$$

Esta restricción indica o modela la restricción de que a cada cliente se le asignará una sola ruta.

$$\sum_{j \in J} \sum_{i \in I \cup J} d_j x_{ijk} \leq Q_k \quad \forall\, k \in K \tag{1.16}$$

Esto se refiere a las restricciones de capacidad para los vehículos. Es decir, la suma de demandas de los clientes de una determinada ruta es menor o igual a la capacidad del vehículo asignado.

$$U_{lk} - U_{jk} + N x_{ijk} \leq N - 1 \quad \forall\, l, j \in J \quad ; \quad k \in K \tag{1.17}$$

Referido a las restricciones de eliminación de subrutas.

$$\sum_{j \in I \cup J} x_{ijk} - \sum_{j \in I \cup J} x_{jik} = 0 \quad \forall\, k \in K \quad i \in I \cup J \tag{1.18}$$

Ésta restricción indica la conservación de flujo, para cada nodo o cliente el total que ingresa es igual al total que sale.

$$\sum_{i \in I} \sum_{j \in J} x_{ijk} \leq 1 \quad \forall\, k \in K \tag{1.19}$$

Restricción referida a que una ruta o vehículo puede ser usado a lo más una vez.

$$\sum_{j \in J} d_i z_{ij} \leq V_i \quad \forall\, i \in I \tag{1.20}$$

Se indica las restricciones de capacidad para los depósitos, es decir para cada depósito la suma de demandas de los clientes abastecidos debe ser menor que la capacidad máxima del depósito.

$$-\, z_{ij} + \sum_{u \in I \cup J} (x_{iuk} + x_{ujk}) \leq 1 \quad \forall\, i \in I\ j \in J\ k \in K \tag{1.21}$$

Esta restricción indica que un cliente es asignado a un depósito solo si hay ruta desde el depósito.

$$x_{ijk} \quad ; \quad z_{ij} \in \{0;1\} \quad ; \quad U_{lk} \geq 0 \tag{1.22}$$

Las variables x y z son binarias, mientras que la variable U es real positiva.

1.4. Delimitación del estudio

Es conocido que la solución exacta para tamaños de problema grandes (mayores a 50 depósitos) será técnicamente imposible de obtener, en consecuencia estaremos enfocados en proponer una solución aproximada(la mejor posible) mediante el diseño e implementación de algunas heurísticas de construcción a las soluciones obtenidas, a su vez se les aplicará mejoras sucesivas mediante las heurísticas de mejora.

La investigación estará enfocada a proponer soluciones aproximadas buenas para cada una de las instancias de tamaño grande (mayores a 50 clientes), las instancias (23 en total desde $p01$ hasta $p23$) constituirán la población de la investigación, estas instancias han sido elaboradas por la comunidad académica dedicada al estudio del ruteo de vehículos y al dia de hoy se mantienen vigentes, de hecho cada nueva propuesta de investigación usa a estas instancias como data de prueba, se prevé proponer más de una estrategia de solución y luego realizar la comparación de los resultados obtenidos con las mejores soluciones obtenidas hasta el momento **BKS**.

1.5. Justificación e importancia del estudio

El problema planteado está clasificado como un problema $NP-$Hard, en ese sentido el problema está abierto, dado que es imposible en la práctica resolver el problema de manera exacta, por ello la justificación de la investigación está dada por la necesidad de proponer alternativas de algoritmos heurísticos que nos permitan encontrar buenas soluciones iniciales las cuales podrán ser mejoradas de manera sistemática

mediante la aplicación de otras heurísticas de mejora.

Por otro lado la importancia de la investigación está en el hecho de proponer nuevas heurísticas cuyos resultados nos permitan aproximarnos a la mejor solución existente al dia de hoy, estableciendo así la bondad de nuestras heurísticas respecto a otros trabajos y sentar las bases para futuras investigaciones que tomando como base la presente logren aún mejores resultados.

1.6. Objetivos de la investigación: General y específicos

Objetivo General:

Resolver el problema del ruteo de vehículos multidepósito (MDVRP) para diferentes instancias, mediante la propuesta de heurísticas de construcción y mejoramiento **originales** cuya funcionamiento y eficiencia será comparado con las actualmente existentes, la propuesta de heurísticas **originales** debe representar una contribución al área de investigación.

Objetivos específicos:

- Diseñar heurísticas de clusterización **originales** por lo menos igual de efectivas que las existentes e implementarlas en el lenguaje de programación JULIA 1.0.5, para encontrar una solución inicial al problema MDVPR.
- Diseñar e implementar heurísticas de mejora **originales**, que partiendo de la solución inicial obtenida se aproxime a la mejor solución tanto como se pueda.
- Realizar las comparaciones de los resultados obtenidas por las heurísticas propuestas y comparar cada una de ellas con las heurísticas existentes en la actualidad asi como con la mejor solución disponible BKS (Best Know Solution).
- Establecer las bases teóricas respecto al tema de investigación para futuras investigaciones relacionadas con el tema y tambien realizar mejoras a futuro.

Capítulo 2

MARCO TEÓRICO

En este capítulo se presentan los conceptos previos necesarios para el desarrollo del presente trabajo en su conjunto, se da una descripción breve acerca de los conceptos de complejidad de algoritmos, la clasificación de los algoritmos según su orden de complejidad computacional asimismo la clasificación de los problemas computacionales en P y NP y finalmente una descripción del problema TSP, en cuanto a su formulación matemática y su complejidad asi como sus variantes. Toda la nomenclatura, definición o notación utilizada en todo el trabajo será definida en este capítulo

2.1. Fundamentos teóricos de la investigación

2.1.1. Análisis y complejidad de algoritmos

En ésta sección presentamos las notaciones y definiciones que se utilizan en ciencias de la computación respecto al análisis de un algoritmo como son problemas computacionales, instancias, correctitud y complejidad de un algoritmo asimismo la notación asintótica como modelo matemático para medir la eficiencia de los algoritmos. Esto con el objetivo de mostrar que los algoritmos propuestos o existentes serán imposibles de ser implementados para tamaños de problemas reales, gran parte deldesarrollo de esta sección y material complementario la podemos encontrar en (Cormen et al., 2009, ormen et al., 2009)

2.1.2. Problemas, algoritmos e instancias

Definición 2.1.1 (Problema computacional)

Se llama así a todo problema que requiere para su solución un programa de computadora mediante la implementación de un algoritmo, un problema computacional siempre define un conjunto de parámetros, en términos de los cuales será cuantificado el tamaño de la entrada.

Definición 2.1.2 (Algoritmo)

Conjunto finito de pasos bien definidos que permiten resolver un problema computacional. En el presente trabajo nos interesamos en una clase mas específica de algoritmos llamados algoritmos computacionales, los cuales reciben un conjunto de datos de entrada (input) y transforman (mediante una secuencia de pasos finita) dichos valores de entrada en datos de salida (output).

Luego entonces, un algoritmo es una herramienta para resolver problemas computacionales. Un problema computacional debe estar descrito mediante:

- Una descripción de sus parámetros y
- un enunciado sobre las propiedades que la solucion debe satisfacer.

Definición 2.1.3 (Instancia)

Es un problema computacional para la cual se ha fijado valores particulares para sus parámetros.

Definición 2.1.4 (Tamaño de instancia)

Es la cantidad de datos necesaria para describir dicha instancia

Ejemplo 1

- **Problema:** *Ordenar de menor a mayor una secuencia de n números reales $\langle a_1, a_2, \cdots a_n \rangle$*
- **Parámetros:** n *y* $\langle a_1, a_2, \cdots a_n \rangle$
- **Algoritmos:** *burbuja, inserción, selección, HeapSort, QuickSort MergeSort entre otros*
- **Instancia:** *Aplicar algún algoritmo (de los mencionados anteriormente) para establecer el orden de mérito de los $n = 500$ alumnos de la Facultad de Ciencias de la UNI según el promedio ponderado.*

- **Tamaño:** *La cantidad de elementos a ordenar dado por* $n = 500$

Se puede observar que un problema computacional es entonces una colección de instancias cada una de las cuales tiene un determinado tamaño.

Definición 2.1.5 (correctitud)
Se dice que un algoritmo es correcto si para cada instancia de la entrada termina con la salida requerida. Si el algoritmo es correcto se dice también que el algoritmo resuelve el problema.

La prueba de correctitud de un algoritmo no es una tarea sencilla, para demostrar formalmente que un algoritmo es correcto se requiere un conocimiento profundo del funcionamiento del mismo. El método a utilizar es el de los **invariantes** el cual consiste en identificar alguna característica del funcionamiento del algoritmo que se repite en cada paso y luego utilizar la inducción matemática para demostrar formalmente que luego de la última iteración dicha invariante se sigue cumpliendo con lo cual el problema ha sido resuelto correctamente.

2.1.3. Consumo de tiempo de los algoritmos

Diremos que un algoritmo resuelve un problema si al recibir la descripción de una instancia del problema devuelve una solución de la instancia (o informa que la instancia no tiene solución). Para cada instancia del problema, el algoritmo consume una cantidad de tiempo diferente. Digamos que el algoritmo consume $T(I)$ unidades de tiempo para procesar la instancia I. La relación entre $T(I)$ y el tamaño de la instancia I proporciona una medida de la eficiencia del algoritmo. En general, un problema tiene muchas instancias diferentes de un mismo tamaño. Esto exige la introducción de los conceptos de **peor caso** e **mejor caso**.

Dado un algoritmo A para un problema y un número natural n sea

$$T^*(n) = \text{máx}\{T(I) \ / \ I \text{ es instancia de tamaño } n \text{ para el problema } \}$$

decimos que $T^*(n)$ mide el consumo de tiempo del algoritmo A en el peor caso; en este caso también diremos que el algoritmo realiza el mayor exfuerzo.

De modo similar podemos definir

$$T_*(n) = \text{mín}\{T(I) \ / \ I \text{ es instancia de tamaño } n \text{ para el problema } \}$$

el cual mide el consumo de tiempo en el mejor caso, es decir cuando el algoritmo realiza el menor exfuerzo. Por ejemplo un algoritmo puede consumir $200n$ unidades de tiempo en el mejor caso y $(10n^2 + 100n)$ unidades de tiempo en el peor caso

Cuando analizamos un algoritmo, supondremos que la ejecución de cada linea del pseudocodigo consume una cantidad de tiempo que no depende del tamaño de la entrada, asimismo supondremos que el consumo de tiempo de las principales operaciones aritméticas (adición, multiplicación, comparación, atribución etc) tampoco depende del tamaño de la entrada.

El análisis de un algoritmo para determinado problema fundamentalmente consiste en :

1. Probar que el algoritmo está correcto y

2. Estimar el tiempo que la ejecución de dicho algoritmo consumirá.

Luego entonces, dados dos o más algoritmos que resuelven un mismo problema el análisis de los mismos permite decidir cual de ellos es el más eficiente. El análisis adecuado de un algoritmo permite preveer el comportamiento del algoritmo antes de que sea efectivamente implementado.

Ilustramos todas los conceptos anteriores con el siguiente ejemplo:

Ejemplo 2 *Análisis del algoritmo de Ordenación por Inserción*

Algoritmo 1 Algoritmo de Inserción

Entrada: Un entero positivo n y un vector $A[1..n]$;

Salida: El vector A tal que: $A[1] \leq A[2] \leq \cdots \leq A[n]$

1: $j \leftarrow 2$
2: **mientras** $j \leq n$ **hacer:**
3: $\quad aux \leftarrow A[j]$
4: $\quad i \leftarrow j - 1$
5: $\quad$ **mientras** $(i > 0)$ **y** $(A[i] > aux)$ **hacer:**
6: $\quad\quad A[i + 1] \leftarrow A[i]$
7: $\quad\quad i \leftarrow i - 1$
8: $\quad$ **fin de mientras**
9: $\quad A[i + 1] \leftarrow aux$
10: $\quad j \leftarrow j + 1$
11: **fin de mientras**
12: retorna A

El análisis del algoritmo de inserción tiene dos etapas: la correctitud y la complejidad o análisis de desempeño, veamos cada etapa:

Correctitud

Ejecutando varias veces el algoritmo para diferentes instancias podemos deducir que el algoritmo en cada paso (es decir para cada valor de j) desplaza una posición hacia la derecha los elementos $A[j-1]$, $A[j-2]$, $A[j-3]$ hasta encontrar la posición correcta del elemento actualmente analizado $A[j]$, tal como se muestra en la siguiente figura:

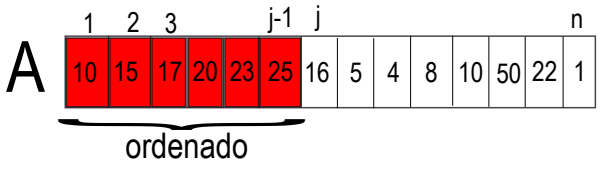


Figura 2.1: Invariante del algoritmo de inserción

Identificamos la siguiente invariante:

$$A[1..(j-1)] \quad \text{está ordenado para cada valor de } j$$

Esta invariante la demostramos por inducción matemática sobre j, veamos:

base: En el primer paso cuando $j = 2$, la secuencia está constituida por un solo elemento, $A[1..(2-1)] = A[1]$ el cual evidentemente está ordenado. Luego la invariante se verifica en el primer paso.

paso inductivo: Suponiendo que la secuencia $A[1..(j-1)]$ está ordenada, veamos qué pasa para $(j+1)$: la instrucción de la linea 5 (que es parte del bloque 5, 6, 7 y 8) termina cuando la condición es falsa, lo cual puede ocurrir en dos casos:

- $i = 0$ en este caso la secuencia está totalmente desplazada, es decir se cumple que $A[i] > aux \ \forall i = 1, 2, \cdots, (j-1)$ y tenemos entonces que al final de la iteración el elemento aux fue insertado en la primera posición, luego la secuencia tiene el siguiente aspecto:

  $$\langle \textbf{aux}, A[1], A[2], \cdots, A[j-1] \rangle$$

  es decir en este caso la invariante es correcta para $(j+1)$

- $A[i] \leq aux$ y $A[k] > aux \ \forall k = (i+1), (i+2), \cdots, (j-1)$, en este caso la secuencia tiene el siguiente aspecto:

  $$\langle A[1], A[2], \cdots, A[i], \textbf{aux}, A[i+1], A[i+2], \cdots, A[j-1] \rangle$$

  es decir que la invariante también se verifica

Luego entonces la invariante se cumple en todos los pasos, en particular en el último paso cuando $j = n + 1$ tenemos que la secuencia está ordenada, esto prueba la correctitud del algoritmo.

complejidad

El análisis de desempeño nos permitirá estimar el tiempo que le tomará al algoritmo resolver un problema de tamaño n, para n arbitrario. Para esto asumimos alguno

hechos básicos, asumimos un modelo de computadora RAM con un procesador donde las instrucciones son ejecutadas secuencialmente una después de la otra, el acceso a la memoria es libre y lleva tiempo constante y finalmente instrucciones diferentes requieren tiempos diferentes de procesamiento.

Sea t_i el tiempo que demora el algoritmo en ejecutar la linea i para $i = 1, 2, 3, \cdots 12$ luego es necesario calcular además cuántas veces se ejecuta cada linea para finalmente determinar el tiempo total que le tomará para resolver el problema. Para todas las lineas es sencillo el cálculo de la cantidad de veces que se ejecuta dicha linea, con exepción de la linea (5) el cual depende de cada valor de j, entonces sea n_j el número de veces que se ejecuta la linea (5) para cada valor de j siendo $1 \leq n_j \leq j$

luego la siguiente tabla proporciona la información acerca de el tiempo y las veces para cada linea.

linea	tiempo empleado por la linea	número de veces que se ejecuta la linea
1	t_1	1
2	t_2	n
3	t_3	$n-1$
4	t_4	$n-1$
5	t_5	$\sum_{j=2}^{n} n_j$
6	t_6	$\sum_{j=2}^{n} (n_j - 1)$
7	t_7	$\sum_{j=2}^{n} (n_j - 1)$
8	t_8	$\sum_{j=2}^{n} n_j$
9	t_9	$n-1$
10	t_{10}	$n-1$
11	t_{11}	n
12	t_{12}	1

Tabla 2.1: Análisis del algoritmo de inserción

Luego el tiempo total de ejecución del algoritmo de inserción está dado por

$$\begin{aligned} T(n) &= t_1 \cdot 1 + t_2 \cdot n + t_3 \cdot (n-1) + t_4 \cdot (n-1) + t_5 \cdot \sum_{j=2}^{n} n_j + \\ &\quad + t_6 \cdot \sum_{j=2}^{n} (n_j - 1) + t_7 \cdot \sum_{j=2}^{n} (n_j - 1) + t_8 \cdot \sum_{j=2}^{n} n_j + \\ &\quad + t_9 \cdot (n-1) + t_{10}(n-1) + t_{11} \cdot n + t_{12} \cdot 1 \end{aligned} \tag{2.1}$$

Si $n_j = 1 \ \forall j$, se tiene el **caso mas favorable o mejor caso**, en este caso el algoritmo realiza su menor exfuerzo, es la situación en la que el vector ingresado ya está ordenado y en cada paso no es necesario desplazar los elementos. Al reemplazar en la ecuación (2.1) obtenemos la expresión:

$$T(n) = (t_2 + t_3 + t_4 + t_5 + t_8 + t_9 + t_{10} + t_{11}) \cdot n + (t_1 - t_3 - t_4 - t_5 - t_8 - t_9 - t_{10} + t_{11})$$

es decir,

$$T(n) = a \cdot n + b \qquad \text{siendo } a \text{ y } b \text{ constantes} \tag{2.2}$$

Por otro lado, si $n_j = j \ \forall j$, se tiene el caso **mas desfavorable** o **peor caso**; en éste caso el algoritmo realiza su mayor exfuerzo, es la situación en la que el vector ingresado está ordenado de mayor a menor y en cada paso se desplazan todos los elementos anteriores. Si reemplazamos en la ecuación (2.1) obtenemos:

$$\begin{aligned} T(n) &= t_1 \cdot 1 + t_2 \cdot n + t_3 \cdot (n-1) + t_4 \cdot (n-1) + \\ &\quad + (t_6 + t_7) \cdot \sum_{j=2}^{n} (j-1) + (t_5 + t_8) \cdot \sum_{j=2}^{n} j + \\ &\quad + t_9 \cdot (n-1) + t_{10}(n-1) + t_{11} \cdot n + t_{12} \cdot 1 \end{aligned} \tag{2.3}$$

es decir,

$$T(n) = a \cdot n^2 + b \cdot n + c \qquad \text{siendo } a, b \text{ y } c \text{ constantes} \tag{2.4}$$

2.1.4. Comparación asintótica de funciones

Es absolutamente deseable expresar el consumo de tiempo de un algoritmo de tal manera que no dependa del lenguaje de programación ni de los detalles de implamentación y mucho menos del computador empleado. Esto es posible mediante la comparación asintótica de funciones en la cual estamos interesados en el comportamiento para valores grandes del argumento de dichas funciones. Existen tres formas

de comparar asintóticamente las funciones: una de tipo $\leq$ (notación $\mathcal{O}$) otra de tipo $\geq$ (notación Ω) y finalmente una tercera del tipo $=$ (notación Θ), pasamos a detallar la primera de ellas por ser la más utilizada por lo menos en el presente trabajo, los detalles de las otras notaciones las encontramos en (Cormen et al., 2009, ormen et al., 2009).

Notación $\mathcal{O}$

Dadas las funciones $F, G : \mathbb{N} \longrightarrow \mathbb{R}^+$, decimos que F pertenece a $\mathcal{O}(G)$

si existen c y $n_0 \in \mathbb{N}$ tales que: $\quad F(n) \leq c \cdot G(n) \qquad\qquad \forall\, n \geq n_0$

tomamos dos formas en cierta forma equivalentes cuyo uso se han popularizado en la literatura para decir lo mismo,

$$F \text{ es del orden de } \mathcal{O}(G) \qquad \text{o} \qquad F(n) = \mathcal{O}(G(n))$$

En realidad $c \cdot G(n)$ constituye una cota superior asintótica para la función $F(n)$ tal como se muestra en la siguiente figura:

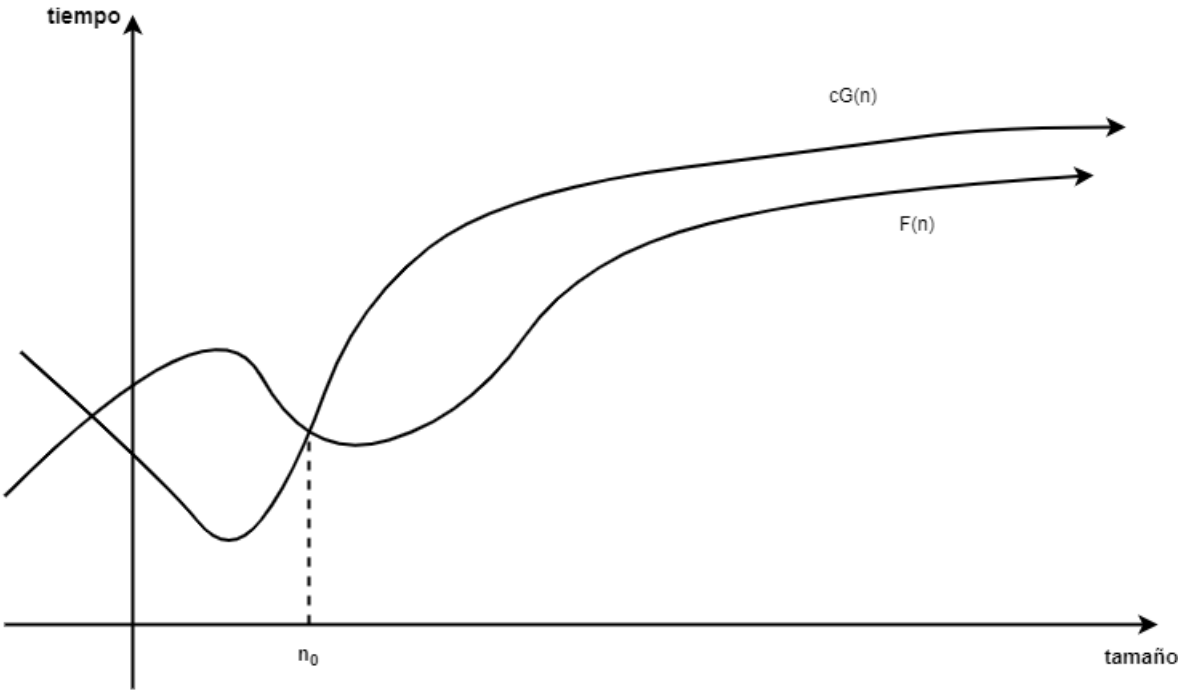


Figura 2.2: Notación asintótica para expresar que $F = \mathcal{O}(G)$, note que antes de n_0 el comportamiento es desconocido, pero cuando $n \geq n_0$, $F(n)$ siempre está por debajo de $c \cdot G(n)$

Por ejemplo:

- $100n$ está en $\mathcal{O}(n^2)$, pues $100n \leq n \cdot n = n^2$ para todo $n \geq 100$.
- $2n^3 + 100n$ es del orden de $\mathcal{O}(n^3)$ pues $2n^3 + 100n \leq 2n^3 + 100n^3 = 102n^3$ para todo $n \geq 1$
- Como $n \leq 2^n$ para todo $n \geq 1$ entonces $\log(n) \leq n$, luego $\log(n) = \mathcal{O}(n)$ (en realidad es posible demostrar que $\log(n) = \mathcal{O}(n^{0,5})$)
- como consecuencia de lo anterior $n \log(n) = \mathcal{O}(n^2)$

Esta herramienta matemática nos permitirá acotar superiormete el tiempo de ejecución de los algoritmos, lo cual a su vez mide la eficiencia del mismo y permite seleccionar el mejor entre varios que resuelven el mismo problema.

2.1.5. Clasificación de los algoritmos

La notación asintótica introducida en la sección anterior permite hacer una clasificación de los algoritmos dependiendo de su complejidad de tiempo. Para esto si $F(n) = \mathcal{O}(G(n))$ entonces diremos que la complejidad del algoritmo es $G(n)$, según diversas funciones G podemos tener la siguiente clasificación respecto al tamaño de entrada n:

Función: $G(n)$	complejidad: $F(n)$	el algoritmo es:
c (constante)	$\mathcal{O}(1)$	constante
n	$\mathcal{O}(n)$	lineal
n^2	$\mathcal{O}(n^2)$	cuadrático
n^3	$\mathcal{O}(n^3)$	cúbico
$n^k \quad (k \in \mathbb{N})$	$\mathcal{O}(n^k)$	polinomial
$\log(n)$	$\mathcal{O}(\log(n))$	logarítmico
$n\log(n)$	$\mathcal{O}(n\log(n))$	polinomial
$c^n \quad (c > 1$ constante)	$\mathcal{O}(c^n)$	exponencial
n^n	$\mathcal{O}(n^n)$	super exponencial

Tabla 2.2: Diferentes tipos de algoritmos dependiendo de su complejidad

A continuación realizamos una comparación entre los diferentes tiempos de ejecución, para ésto supondremos que disponemos de una máquina capáz de efectuar un millón de operaciones por segundo, es decir la velocidad de la máquina esta dada por $v = 10^6 \ \frac{operaciones}{segundo}$, siendo $T(n)$ la cantidad de operaciones que realiza la máquina para ejecutar determinado algoritmo, entonces el tiempo (en segundos) que le toma a la máquina para ejecutar el algoritmo está dado por

$$t = \frac{T(n)}{10^6} \quad \text{segundos}$$

considerando diferentes tamaños de entrada y diferentes tiempos de ejecución tenemos los siguientes resultados:

	10	20	40	50	100	500
n	0,00001 s	0,00002 s	0,00004 s	0,00005 s	0,0001 s	0,0005 s
$n\log(n)$	0,00003 s	0,00008 s	0,00021 s	0,00028 s	0,00066 s	0,00448 s
n^2	0,0001 s	0,0004 s	0,0016 s	0,0025 s	0,01 s	0,25 s
n^3	0.001 s	0,008 s	0,64 s	0,125 s	1 s	125 s
n^{10}	2,7 horas	118 dias	3,3 siglos	30 siglos	$3 \cdot 10^4$ siglos	$3 \cdot 10^{11}$ siglos
2^n	0.001 s	1 s	12,7 dias	35,7 años	$4 \cdot 10^{14}$ siglos	10^{135} siglos
n^n	10000 s	$3 \cdot 10^{16}$ horas	$3,8 \cdot 10^{50}$ años	$2,8 \cdot 10^{69}$ siglos	$3,2 \cdot 10^{184}$ siglos	∞

Tabla 2.3: El tiempo que tarda un computador que realiza un millón de operaciones por segundo para diferentes tamaños de entrada y diferentes algoritmos

Observando los resultados anteriores corresponde realizar la pregunta: ¿Que tanto debe mejorar la velocidad de la máquina para que los algoritmos polinomiales de grado alto o los exponenciales resuelvan los problemas en tiempos mas realistas?.

Supongamos que un computador actual con su velocidad resuelve en una hora problemas de tamaños a b c y d, calculamos en el mismo tiempo el tamaño de problema que sería capáz de resolver computadoras con mayor velocidad. Por ejemplo tomemos un algoritmo cuya complejidad es cúbica y comparamos un computador actual y uno cuya velocidad es 1000 veces mayor, entonces

$$1 \text{ hora} = \frac{T(n)}{v} = \frac{c^3}{v} = \frac{x^3}{1000v} \quad \Longrightarrow \quad x = 10 \cdot c$$

Es decir, Si la máquina actual puede ejecutar en una hora una instancia de tamaño c, entonces un computador futuro cuya velocidad sea 1000 veces superior a la actual podrá ejecutar en el mismo tiempo una instancia de tamaño $10 \cdot c$.

En la siguiente tabla comparamos diferentes complejidades así como el desempeño de un computador actual y computadoras 100 veces y 1000 veces más rápido.

Función de complejidad	Computador actual puede resolver un problema de tamaño	computador 100 veces mas rápido puede resolver	computador 1000 veces mas rápido puede resolver
n	a	$100 \cdot a$	$1000 \cdot a$
n^2	b	$10 \cdot b$	$31,6 \cdot b$
n^3	c	$4,6 \cdot c$	$10 \cdot c$
2^n	d	$d + 6,6$	$d + 10$

Tabla 2.4: ¿Qué esperanza tienen los algoritmos de mejorar su desempeño con el avance de la tecnología?

Se observa de la tabla anterior que el desempeño de un algoritmo polinomial puede mejorar con la mejora de la tecnología ya que hay un factor multiplicativo; sin embargo el desempeño de un algoritmo de complejidad exponencial no tiene muchas esperanzas de mejorar ya que la mejora es sólo en una constante aditiva.

2.2. Marco conceptual

2.2.1. Clasificación de los problemas en clases P y NP

El problema motivo de la presente investigación del ruteo de vehículos multidepósito, está clasificado como $NP - Hard$, el motivo de la presente sección es dar una visión general de la clasificación de los diferentes problemas computacionales, entre ellas la de la optimización combinatoria.

La mayoría de los algoritmos que son implementados y funcionan en la práctica se ejecutan en un tiempo que está limitado por un polinomio en el tamaño de la entrada, como por ejemplo:

- Algoritmos de ordenación $\mathcal{O}(n^2)$, $\mathcal{O}(n \log(n))$
- Multiplicación de matrices: $\mathcal{O}(n^3)$, $\mathcal{O}(n^{\log_2 7})$
- Algoritmos en grafos: $\mathcal{O}(V + E)$, $\mathcal{O}(E \log(V))$

De hecho, el tiempo de ejecución de estos algoritmos está limitado por polinomios de grado pequeño.

Definición 2.2.1

Se dice que un problema computacional es **tratable** *si y solo si, puede ser resuelto mediante un algoritmo que se ejecuta en tiempo polinomial.*

Por ejemplo los problemas de ordenación, problemas de multiplicación de matrices y los algoritmos en grafos. Los problemas computacionales para los cuales no existe un algoritmo que permita su solución en tiempo polinomial se dice **intratable**. También se llama asi a aquellos problemas que pueden ser resueltos teóricamente pero en la práctica son imposibles de resolver, por ejemplo un problema computacional cuya solución requiere de un algoritmo que realiza n^{100} operaciones.

Definición 2.2.2

Se dice que un problema computacional es de **decisión** *si y solo si, tiene como respuesta si o no o alternativamente cierto o falso.*

Por ejemplo, el problema de la ruta mas corta en un grafo, cuya formulación consiste en dado un grafo $G = (V, E)$ no dirigido y sin pesos en los arcos, dos vértices u y v en V y un entero no negativo k, existe un camino en G que une u y v cuya longitud es a lo sumo k.

Definición 2.2.3 (La clase P)

Se define la clase P como la clase de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial.

Debe notarse que los problemas de optimización y los problemas de búsqueda son equivalentes a un problema de decisión, por ejemplo el problema de la ruta más corta planteado en la definición anterior puede ser formulado también de la siguiente manera: Dado un grafo sin pesos y no dirigido $G = (V, E)$ y dos vértices u y v, encontrar el camino más corto que une los vértices u y v.

Es conveniente definir los problemas de decisión para que sean manejables si pertenecen a la clase P, puesto que:

- La clase P es cerrada bajo la operación de composición.

- La clase P es casi independiente del modelo computacional que se considera.

Esto por el hecho práctico de que, nadie consideraría un problema que requiera un algoritmo de complejidad $\Omega(n^{100})$ como eficientemente resoluble. Sin embargo, parece que la mayoría de los problemas en P que son interesantes en la práctica se puede resolver de manera bastante eficiente. Sabemos que P contiene muchos problemas de manera natural, por ejemplo la versión problema de decisión del clásico problema de programación lineal, el cálculo del máximo común divisor y encontrar en un grafo el emparejamiento máximo. En el año 2002, se demostró que el problema de determinar si un número es primo o no está en la clase P

Definición 2.2.4 (La clase NP)
Es la clase de problemas de decisión que pueden ser resueltos sobre una máquina de Turing no determinista en un tiempo polinomial

Una máquina de Turing no es una máquina física sino un mecanismo lógico por medio del cual el cálculo puede descomponerse en iteraciones de operaciones concretas extremadamente simples (controladas por un programa). El principio de Turing es **cualquier procedimiento que puede ser descrito con precisión puede ser programado para que lo realice una computadora**
Observaciones:

- NP es la abreviatura de tiempo polinomial no determinista (Non-deterministic Polinomial-time)
- NP no significa **no polinomial** dado que existen problemas que ni siquiera pueden ser verificados en tiempo polinomial.
- La clase NP es el conjunto de problemas que pueden ser **verificados** por una máquina de Turing determinista en tiempo polinomial.
- Todos los problemas de esta clase tienen la propiedad de que sus soluciones pueden ser comprobado de forma eficaz.

Esta clase NP contiene muchos problemas que a todos nos gustaría poder resolverlos de manera eficaz, como son: el problema de la satisfactibilidad booleana (SAT), el problema del camino Hamiltoniano (caso especial de TSP) o el problema de la cobertura

de vértices en un grafo. Es posible demostrar que $P \subseteq NP$ puede verse en (Cook, 2006, ook, 2006)

Definición 2.2.5 (La clase $NP-$completo)

Esta clase la conforman todos los problemas que pertenecen a NP clasificados como los mas dificiles de resolver, en el sentido de que tienen mayor probabilidad de no estar dentro de la clase P

Debemos notar que si fuera posible resolver de manera rápida y eficiente (en tiempo polinomial) cualquier problema que se encuentra en la clase NP, entonces se podría usar dicho algoritmo para resolver todos los problemas NP en forma eficiente (tiempo polinomial). Asimismo observar que en la actualidad, todos los algoritmos conocidos para resolver problemas $NP-$completos requieren tiempo superpolinomial en el tamaño de entrada. Finalmente para resolver un problema $NP-$completo para cualquier tamaño de problema no trivial, generalmente se utilizan técnicas especiales como Aproximación, probabilísticos o heurísticas.

A continuación una lista de problemas conocidos entre otros que están clasificados o catalogados como problemas $NP-$completos.

El problema de la satisfactibilidad booleana (SAT): Consiste en saber si, dada una expresión booleana con variables y sin cuantificadores, existe por lo menos una asignación de valores para sus variables que hacen que la expresión booleana sea verdadera. Por ejemplo, una instancia de SAT sería el saber si existen valores verdadero (**V**) o falso (**F**) para x_1, x_2, x_3, x_4 tales que la expresión: $(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$ sea verdadera **V**

El problema de la mochila: Supongamos que tenemos n distintos tipos de objetos, enumerados desde 1 hasta n y de cada tipo de objeto se tienen q_i copias disponibles, donde $q_i \in \mathbb{N}$ y cumple $1 \leq q_i < \infty$. Cada objeto i tiene un beneficio asociado dado por b_i y un peso (o volumen) w_i (ambas cantidades usualmente no negativos). Por otro lado se tiene una mochila, donde se pueden introducir los ítems, que soporta un peso máximo (o volumen máximo) W. El problema consiste en introducir objetos en la mochila de tal forma que se maximice el beneficio total y no se supere el peso (o volumen) máximo que puede soportar la mochila.

El problema del ciclo Hamiltoniano: Dado un grafo $G = (V, E)$, un ciclo de Hamilton es un camino cerrado que pasa por todos los vértices del grafo, el problema consiste en determinar si un grafo dado contiene un ciclo de Hamilton.

El problema del agente viajero: Consiste en encontrar un recorrido completo que conecte todos los nodos de una red, visitándolos una sola vez y volviendo al punto de partida, y que además minimice la distancia total de la ruta, o el tiempo total del recorrido.

El problema del isomorfismo de grafos: Dados dos grafos $F = (V_F, E_F)$ y $G = (V_G, E_G)$, se dice que ellos son isomorfos si existe una biyección entre los vértices $f : V_F \longrightarrow V_G$ tal que los arcos se preserven. El problema consiste en si dados dos grafos con el mismo número de vértices n y arcos m son isomorfos o no. Este problema admite un ataque por fuerza bruta que exigiría comprobar si las $n!$ biyecciones posibles preservan la adyacencia, pero no se conoce un algoritmo eficiente, para el caso general. El problema del isomorfismo de grafos presenta una curiosidad en teoría de complejidad computacional al ser uno de los pocos problemas citados por Garey y Johnson en 1979 pertenecientes a NP de los que se desconoce si es resoluble en tiempo polinómico o si es NP-completo (actualmente está en revisión la demostración de que el problema está en P)

El problema de la cubrimiento de vértices: Sea el grafo $G = (V, E)$ La cobertura de vértices del grafo G es un subconjunto S de V (el conjunto de vértices) tal que para cada arista (a, b) del conjunto de arcos E, ya sea el vértice a o el vértice b pertenece al conjunto S.

El problema de coloración en grafos: Dado un grafo $G = (V, E)$ el problema de coloración de arcos o vértices consiste en asignar la menor catidad de colores a los arcos o vértices de tal manera que dos vértices adyacentes no tengan asignado el mismo color o dos arcos adyacentes no tengan asignado el mismo color.

Existen muchos problemas que parecen ser $NP-$completos pero finalmente no lo son, por ejemplo el recorrido de Euler, ¿existe un camino que conecte dos vértices y use todos los arcos exactamente una vez?

- Si lo que buscamos es un **tour de Euler**, es decir, debemos volver al mismo punto de partida, existe solución sólo si es un grafo conexo y cada vértice tiene un número par de arcos adyacentes.

- Si lo buscado es un **camino Euleriano**, es decir, debemos cubrir todos los arcos una vez, pero no volvemos al punto de partida, hay solución si y sólo si, el grafo es conexo y hay exactamente dos nodos de grado impar.

En ambos casos hay una forma analítica de determinar si hay o no solución al problema de decisión.

Definición 2.2.6 (La clase $NP - Hard$)
Cuando se prueba que un problema de optimización combinatoria en su versión problema de decisión pertenece a la clase $NP-$completa, entonces la versión optimización es $NP - hard$

Algunos problemas $NP-Hard$ están también en NP (son los llamados $NP-$completos), pero otros no están en NP (Karp, 1972, arp, 1972) en el artículo Reducibility Among Combinatorial Problems, Proceedings of a Symposium on the Complexity of Computer Computations, usa el teorema de Cook (de que el SAT es un problema NP-completo), para probar que otros 21 problemas también lo son incluyendo, la programación entera, los ciclos Hamiltonianos (problema del agente viajero), el número cromático entre otros. Muchos de esos problemas aparecen en problemas de optimización del mundo real y Karp mostró que si uno de ellos tuviera un algoritmo eficiente todos lo tendrían. En términos simples los problemas mas complicados de la clase $NP-$completo justamente son los problemas $NP - Hard$. Cabe mencionar que el problema del agente viajero TSP tiene ésta categoría y al ser el problema TSP un caso particular del ruteo de vehículos multidepósito MDVRP, se concluye que el problema que abordaremos está categorizado como $NP - Hard$. La siguiente gráfica nos muestra la posición relativa entre las diferentes clases estudiadas en la presente sección, teniendo en cuenta que la pregunta $P = NP$ aún es un problema abierto vea (Cabezas, 2014, abezas, 2014), presentamos la posición relativa en ambos casos.

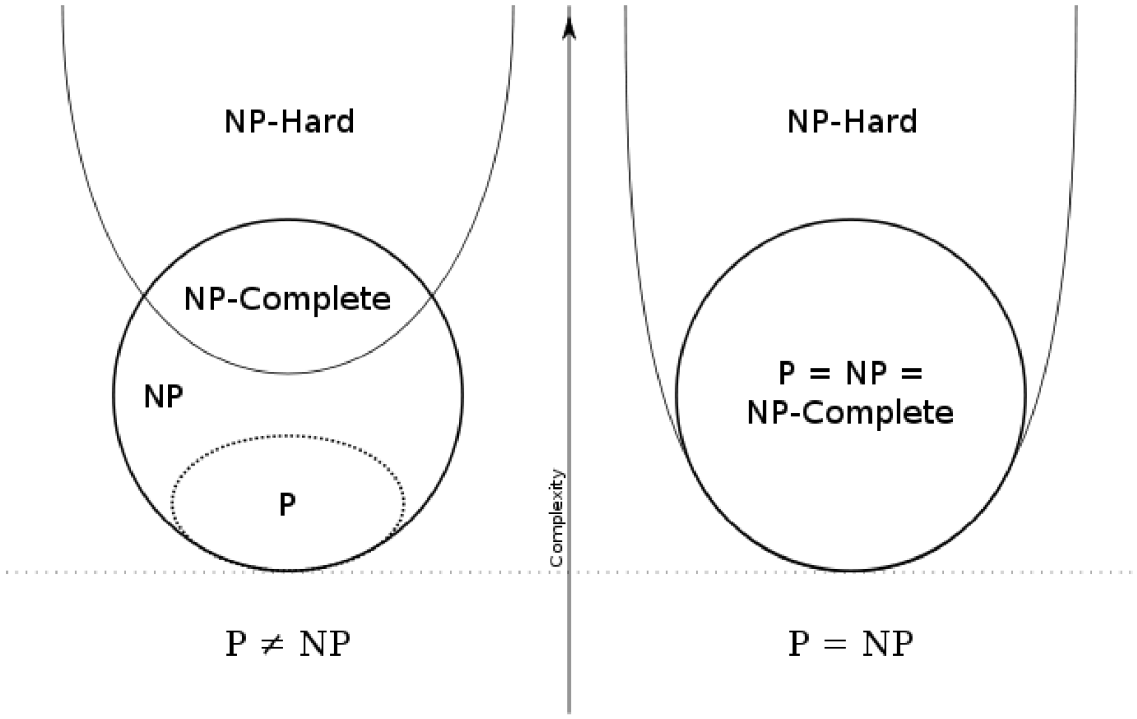


Figura 2.3: Las diferentes clases de problemas computacionales, entre todas las clases los problemas que pertenecen a la clase $NP - Hard$ son los más difíciles de resolver en tiempo polinomial.

Todo lo mencionado en esta subsección puede ser resumido en las tablas 2.5 y 2.6 que describimos a continuación, antes una definición

Definición 2.2.7 (Problema de Optimización Combinatoria)
Se llama asi a un problema de optimización

$$\min_{x \in S} f(x)$$

donde $S \subset \mathbb{R}^n$ es un conjunto **discreto** *que está definido por un conjunto finito de restricciones y $f : S \longrightarrow \mathbb{R}$ una función (lineal o no) real de variable vectorial.*

Problema de	**Forma de resolver**	**tipo de solución**
Decisión	Responder SI o NO a una pregunta en un problema.	¿Tiene solución el problema?
Búsqueda de una solución	Verificar la existencia e identificar una solución factible para un problema (generalmente cuando se quiere encontrar una solución inicial).	Una solución factible
Optimización	Verificar la existencia e identificar la mejor solución posible (dentro de las soluciones factibles para un problema)	La mejor solución factible

Tabla 2.5: Clasificación de los problemas computacionales

clase de complejidad	Tipo de problemas que pertenecen a la clase	tipo de solución
P	Corresponde a los problemas cuyos algoritmos de solución son de complejidad en tiempo polinomial.	tratable
NP	Contiene a los problemas cuya solución hasta la fecha no han podido ser resueltos de manera exacta por medio de algoritmos deterministas, pero que pueden ser resueltos por algoritmos no-deterministas y cuya solución son de complejidad en tiempo polinomial. (su solución puede ser verificada en tiempo polinomial) (la N de no-deterministicos y la P de polinómicos).	intratable
$NP-$ completo	Estos problemas se caracterizan por ser todos **iguales** en el sentido de que si se descubriera una solución S para alguno de ellos, esta solución sería fácilmente aplicable a todos ellos, cumplen dos condiciones: (a) es un problema NP y (b) todo problema de NP se puede transformar polinomialmente en él.	intratable
$NP-$Hard	Un problema que satisface la segunda condición pero no la primera pertenece a la clase $NP - hard$.	intratable

Tabla 2.6: Tipos de complejidad de problemas computacionales

2.2.2. El problema del agente viajero (TSP)

El problema del agente viajero o el problema del vendedor viajero (TSP por sus siglas en inglés **T**raveling **S**alesman **P**roblem) es uno de los problemas más estudiados en matemática computacional por sus diversas aplicaciones en diferentes actividades, como el reparto de mercancías, producción de circuitos integrados, genética, logística, planeación de presupuesto y otros. Resolver este problema demanda gran esfuerzo computacional es por ello que el (TSP) está clasificado como un problema con com-

plejidad $NP - hard$ vea (Lenstra and Kan, 1981, enstra and Kan, 1981), hasta hoy en día no es posible encontrar la solución exacta a problemas con un gran número de ciudades, el problema consiste en encontrar la ruta mínima posible sobre un conjunto de ciudades, conociendo las distancias entre cada par de ciudades, con la condición de visitar cada ciudad una sola vez y volviendo al punto de partida (encontrar un ciclo Hamiltoniano que tenga el menor costo posible). Una característica atractiva del problema del vededor viajero es que las soluciones y los procedimientos de solución a menudo se pueden visualizar. Existen diferentes formulaciones o modelos matemáticos para la descripción del problema TSP, consideramos la de Miller-Tucker-Zemlin la cual elimina la aparición de subrutas, se encuentra en el artículo (Desrochers and Laporte, 1991, esrochers and Laporte, 1991).

Si no tenemos cuidado en la formulación del modelo matemático, podríamos obtener subrutas en la solución como el mostrado en la figura.

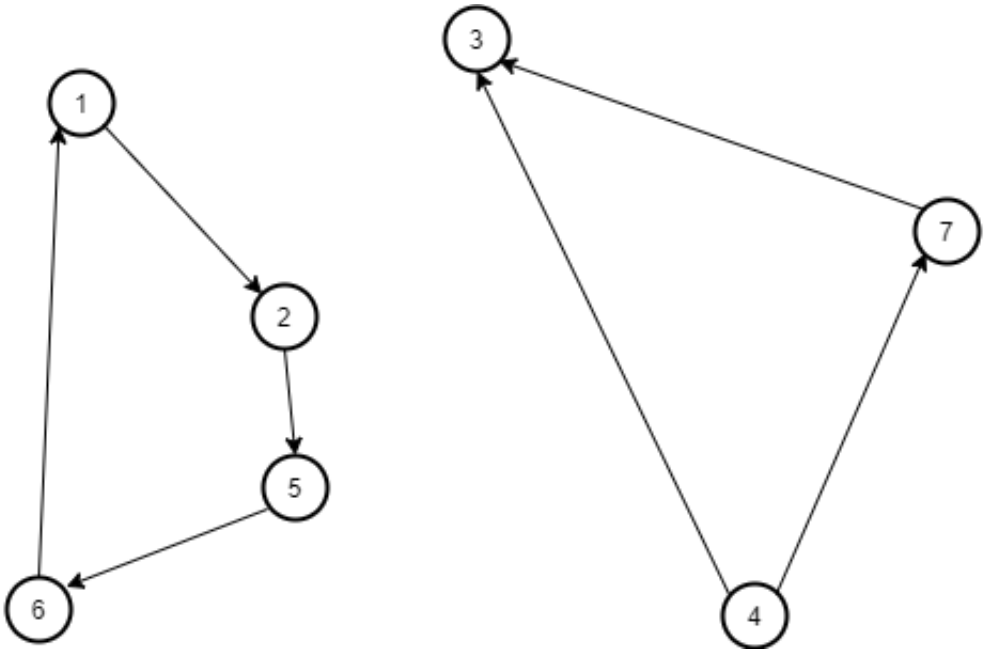


Figura 2.4: Posible ruta en la cual se formaron dos subrutas, en este caso $n = 7$.

Formulación de Miller-Tucker-Zemlin (1960)

Las restricciones usuales no son suficientes para resolver el problema, pues no evitan que puedan obtenerse sub-rutas como solución en el artículo (Miller et al.,

1960, iller et al., 1960) Miller-Tucker-Zemlin propusieron una formulación en (1960), por otro lado en (Padberg and Sung, 1991, adberg and Sung, 1991) se puede encontrar que para un problema con n ciudades se necesitan a lo más n^2 restricciones logrando evitar que se generen subrutas en la solución, se agrega a la formulación usual del problema el siguiente conjunto de restricciones:

$$\begin{aligned} \alpha_i - \alpha_j + nx_{ij} &\leq n-1 \\ i &= 1,2,3,\cdots,n \\ j &= 1,2,3,\cdots,n \end{aligned}$$

Donde α_i es una variable real positiva asociado a la ciudad i.

Definición de los parámetros

$n:$ cantidad de ciudades

$c_{ij}:$ El costo (o tiempo) que se tarda en viajar desde la ciudad i hasta la ciudad j.

Definición de la variable binaria

$$x_{ij} = \begin{cases} 1 & \text{Si el vendedor llega directamente de la ciudad } i \text{ a la ciudad } j. \\ 0 & \text{En otro caso.} \end{cases}$$

Definición de variable continua.

$$1 \leq \alpha_i \leq n-1$$

Formulación del modelo matemático M-T-Z

$$\text{mín} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$s.a. \sum_{i=2}^{n} x_{1i} = t$$

$$\sum_{i=2}^{n} x_{i1} = t$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in \{1, 2, ..., n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1, 2, ..., n\}$$

$$\alpha_i - \alpha_j + nx_{ij} \leq n - 1 \qquad 2 \leq i, j \leq n \text{ con } i \neq j$$

$$1 \leq \alpha_i \leq n + 1 \qquad \forall i \in \{1, ..., n\}$$

$$\left\lceil \frac{n-1}{p} \right\rceil \leq t \leq n - 1$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, 2, ..., n\}.$$

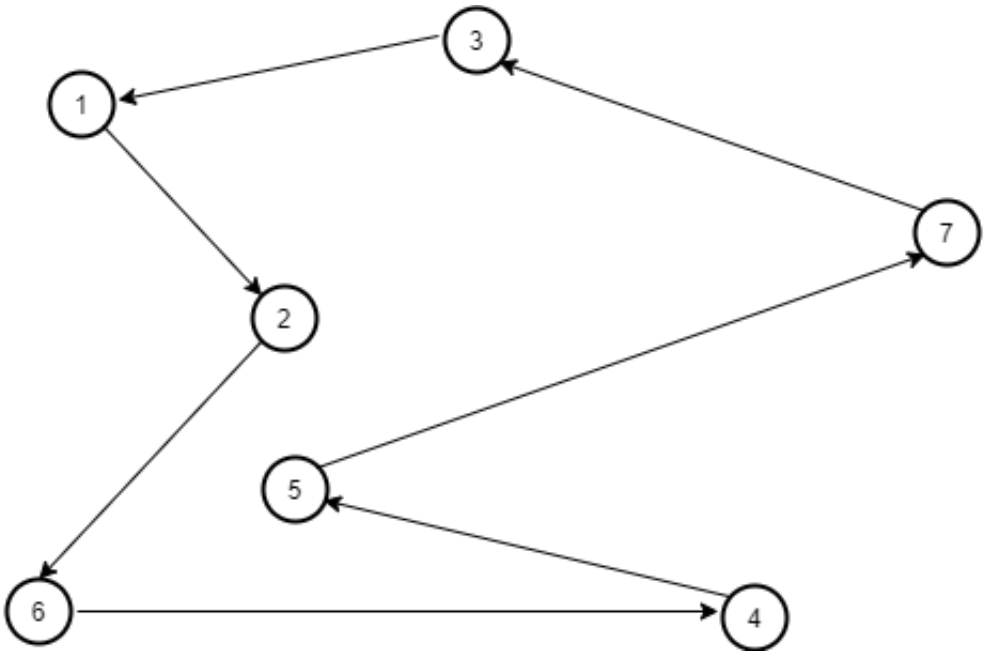


Figura 2.5: Posible ruta sin bucles ni subrutas, con $n = 7$.

Observación

Cuando $t = 1$ y $p \geq n - 1$ se obtiene la formulación estándar con n^2 restricciones y $n^2 - 1$ variables.

Formulación estándar de Miller-Tuker-Zemlin

$$\text{mín} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$s.a. \sum_{i=1,i\neq j}^{n} x_{ij} = 1 \qquad \forall\, j \in \{1,2,...,n\}$$

$$\sum_{j=1,j\neq i}^{n} x_{ij} = 1 \qquad \forall\, i \in \{1,2,...,n\}$$

$$\alpha_i - \alpha_j + nx_{ij} \leq n-1 \qquad \forall i,j \in \{2,3,...,n\}.$$

$$1 \leq \alpha_i \leq n-1 \qquad \forall\, i \in \{2,...,n\}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i,j \in \{1,2,...,n\}$$

.

Para ver como funciona la restricción de Miller-Tucker-Zemlin
Supongamos que exista un ciclo con k ciudades $\{i_1, i_2, ..., i_k, i_1\}$ con $k < n$.

$$x_{i_1,i_2} = x_{i_2,i_3} = x_{i_3,i_4} \ldots = x_{i_{k-1},i_k} = x_{i_k,i_1} = 1$$

Como parte de las restricciones se tiene:

$$\begin{aligned}
\alpha_{i_1} - \alpha_{i_2} + (n) &\leq n-1 \\
\alpha_{i_2} - \alpha_{i_3} + (n) &\leq n-1 \\
\alpha_{i_3} - \alpha_{i_4} + (n) &\leq n-1 \\
\vdots \qquad\qquad &\quad \vdots \\
\alpha_{i_k} - \alpha_{i_1} + (n) &\leq n-1
\end{aligned}$$

Sumando las k ecuaciones obtenemos:

$$kn \leq k(n-1)$$

Lo cual es una contradicción.
Finalmente un teorema que indica la clasificación para el problema TSP

Teorema 2.1

El problema del agente viajero TSP es $NP-$completo y $NP-Hard$

La prueba la encontramos en (Papadimitriou and Steiglitz, 1998, apadimitriou and Steiglitz, 1998), ésta aparente contradicción en realidad no lo es, tan solo prueba la compleja clasificación de los problemas computacionales. El hecho de que el problema TSP sea $NP-Hard$ indica que es muy difícil llegar a calcular una solución general eficiente para este problema, en tanto el hecho de que sea $NP-$completo, se demuestra a partir de considerar el problema del circuito Hamiltoniano en un grafo como un caso particular del TSP y siendo conocido que el problema del circuito Hamiltoniano es $NP-$completo, el problema TSP también es $NP-$completo.

El problema del agente viajero TSP es un problema de optimización combinatoria y por tanto se puede plantear el problema en tres versiones: versión optimización, versión evaluación y versión de econocimiento, la complejidad computacional de cada una de estas versiones es diferente, justificándose de esta manera la posible contradicción mencionada en el teorema 2.1.

2.2.3. El problema VRP y su complejidad computacional

El Problema de Ruteo de Vehículos (Vehicle Routing Problem) o simplemente VRP, por sus siglas en inglés, consiste en determinar un conjunto de rutas para una flota de vehículos que parten de uno o más depósitos o almacenes para satisfacer la demanda de varios clientes dispersos geográficamente. El objetivo es entregar los bienes y satisfacer la demanda de dichos clientes minimizando el costo total generado principalmente por las rutas que son recorridos por los vehículos. En los últimos años el problema del ruteo de vehículos ha tomado gran importancia para las empresas ya

que el costo del transporte, tanto en la industria como en el sector de servicios, representa una porción importante en el valor final del producto o del servicio ofrecido, según (Toth and Vigo, 2002, oth and Vigo, 2002) dicho costo representa entre el 10 % y el 20 % del costo final de los bienes. por tal razón lograr una adecuada distribución de los productos a los usuarios finales juega un papel importante en la gestión de sistemas y su adecuada planificación puede significar considerables ahorros para las empresas.

Pero no sólo en el ámbito empresarial ha tomado fuerza este tema ya que los problemas de ruteo de vehículos dentro del área de programación matemática se clasifican como problemas de Optimización Combinatoria y pertenecen, en su mayoría, a la clase $NP - Hard$. Por tal razón, en el ámbito académico existe una gran motivación por encontrar y estudiar nuevas técnicas que den solución a este tipo de problemas ya que no es posible construir un algoritmo exacto que en un tiempo polinomial resuelva cualquier instancia del problema ya que el tiempo que se demora el algoritmo en encontrar una solución óptima, crece de manera exponencial con el tamaño del problema.

Componentes de un problema de Ruteo de Vehículos

Un problema de ruteo de vehículos puede presentar diferentes características en cuanto a los clientes, depósitos y vehículos, éstas diferencias precisamente dan lugar a diferentes variantes del problema.

Los clientes: Cada cliente tiene cierta demanda que debe ser cubierta por algún vehículo, esta demanda puede ser determinística o estocástica. En algunos casos, es posible que un mismo vehículo no pueda satisfacer la demanda de todos los clientes en una misma ruta. En otros casos la demanda no es un bien sino un servicio donde se da por cumplido el objetivo si el cliente es visitado por un vehículo, lo usual es que cada cliente deba ser visitado exactamente una vez, sin embargo, en ciertos casos se acepta que la demanda de un cliente sea satisfecha en momentos diferentes y por vehículos diferentes. Otra característica relacionada con los clientes es que se puede tener restricciones asociadas al instante de tiempo en el cual se puede visitar al cliente (horario pactado). Usualmente estas

condiciones se expresan como intervalos de tiempo conocidos como ventanas de tiempo.

Los Depósitos: Los vehículos y los productos a entregar usualmente parten de un solo depósito y se tiene como condición que cada ruta definida comience y finalice en el depósito, sin embargo existen variaciones donde se tienen problemas que consideran problemas multidepósito en los que cada depósito tiene características propias como lo son su ubicación y la capacidad máxima de almacenamiento. Cuando el vehículo no está obligado a regresar al depósito se tipifica un OpenVRP, situación que se presenta en la vida real, por ejemplo, cuando el conductor es el mismo propietario del vehículo.

Los Vehículos: La flota de vehículos puede ser homogénea o heterogénea en cuando a su capacidad, tipo de producto que puede transportar y el costo fijo en el que se incurre al usar cada vehículo. Se encuentran problemas con vehículos con capacidad limitada o ilimitada. Es posible encontrar restricciones sobre el tiempo máximo que un vehículo puede estar en circulación y en algunos casos se desea que la cantidad de trabajo realizado por los vehículos (usualmente el tiempo de viaje) sea equilibrada entre ellos.

Em la siguiente figura se puede observar una instancia para un problema VRP.

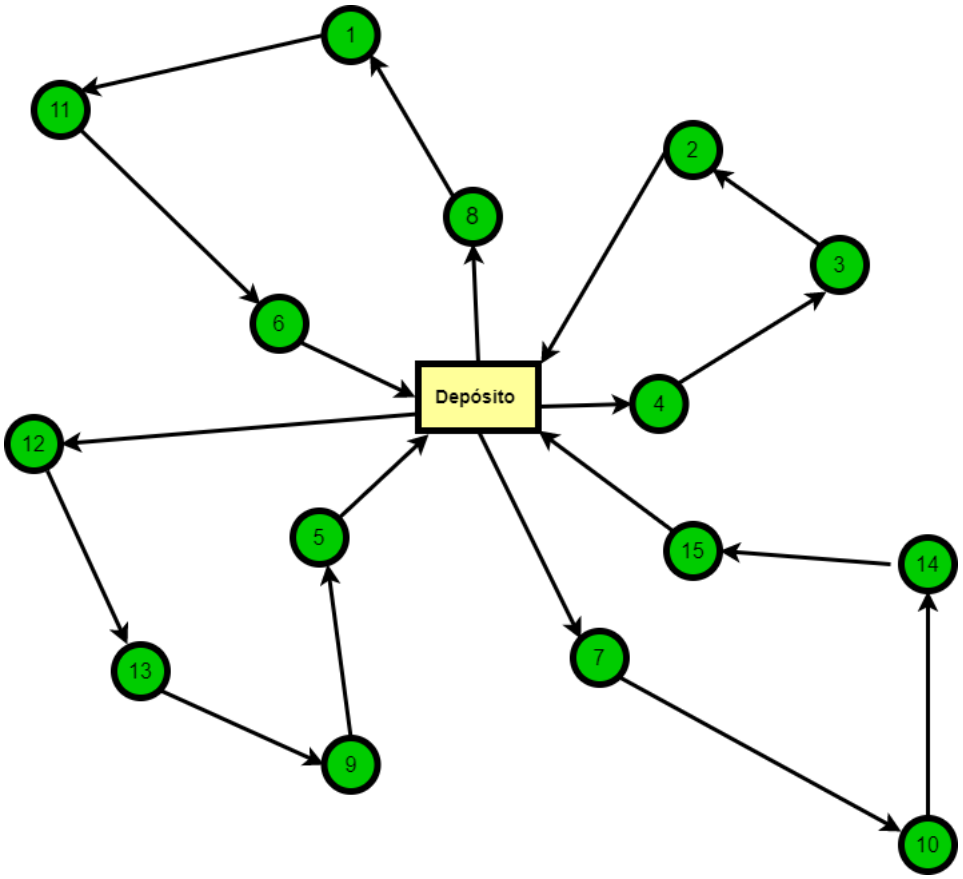


Figura 2.6: Una instancia para el problema VRP con 15 clientes.

El problema VRP tiene una formulación o modelo matemático, sin embargo al ser motivo de estudio de la presente tesis el ruteo de vehículos multidepósito, consideramos que el modelo matemático vista en la seccion 1.3 del capítulo 1 ya está cubierto el caso general. Finalmente enunciamos un teorema respecto a la complejidad computacional del VRP

Teorema 2.2

El problema VRP está catalogado como $NP - Hard$

Esto ocurre dado que un problema VRP, sin considerar demandas en los nodos o clientes se reduce a un problema del agente viajero TSP del cual ya conocemos que es $NP-$ completo y $NP - Hard$, en consecuencia VRP es $NP - Hard$.

Corolario 2.3

El problema MDVRP es $NP - Hard$

Esto es inmediato considerando el caso particular de un solo depósito el cual es VRP en su forma simple, de este modo se tiene el corolario.

2.2.4. Algoritmos, Heurísticas y Metaheurísticas

Para la solución de un problema de optimización combinatoria en general y en particular para el problema MDVRP a tratar en la presente investigación, tenemos tres estrategias:

- **Métodos exactos** Aquellos algoritmos que encuentran la solución exacta, basados en la implementación del problema de programación lineal entera o binaria (generalmente se aplican técnicas de ramificación y acotamiento), los tiempos de ejecución de dichos algoritmos para tamaños de problema $n \geq 50$ depósitos son muy grandes (de orden exponencial).

- **Heurística** En el contexto de la Optimización Combinatoria, se llama así a la implementación via algoritmo de una idea original y creativa que permitirá resolver un problema computacional que proviene de la optimización. Como resultado de la implementación de una heurística se obtene una solución aproximada que si bien no es la exacta es de aceptable calidad (en el sentido de ser los mas cercano posible a la exacta). Las heurísticas dentro del contexto mencionado pueden ser Constructivas o de Mejora.

- **Metaheurística** Son heurísticas (desarroladas a partir de finales de 1990) mucho más específicas, siendo su objetivo encontrar soluciones de mayor calidad (respecto a una heurística). Modifican soluciones intermedias guiándose por la idoneidad de su función objetivo. Computacionalmente son las que han proporciondo las mejores soluciones. A su vez se clasifican en: Redes neuronales, Búsqueda Tabú, Algoritmos genéticos, Colonia de hormigas, entre otras.

Capítulo 3

MARCO METODOLÓGICO

En el presente capítulo se formula la hipótesis central del problema de investigación la cual deberá ser contrastada con los resultados obtenidos, se definen las variables e indicadores de la investigación asi como los métodos y el esquema de investigación, se hace un listado cronológico de las actividades del proceso de investigación, mencionando como técnicas e instrumentos de investigación a las estrategias teóricas para abordar la solución exacta del problema MDVRP, como son la relajación lagrangiana y la optimización subgradiente, desde el punto de vista teórico. El objetivo principal de la investigación son las heurísticas de construcción, llamadas heurísticas de clusterización se proponen tres heurísticas (por distancias(**HD**), por distancias y demandas(**HDD**) y por distancias, demandas y zonas(**HDDZ**)) que nos permitarán construir una solución inicial factible. De la misma manera se proponen las heurísticas de mejoramiento (vecino cercano y split), los cuales a su vez nos permitirán mejorar la solución inicial obtenida por una de las heurísticas de construcción.

3.1. Hipótesis central de la investigación

El problema planteado en el trabajo de tesis doctoral llamado problema del ruteo de vehículos multidepósito (MDVRP por sus siglas en inglés) tiene como parámetros la cantidad de clientes, la cantidad de depósitos, la cantidad de vehículos, la demanda de cada cliente, la capacidad de cada vehículo y la capacidad máxima de cada depósito, asimismo se conoce la ubicación en un sistema de coordenadas de cada uno de los

clientes y depósitos, éstos parámetros principalmente la cantidad de clientes constituye el tamaño del problema a resolver. El problema a resolver consiste en definir un conjunto de rutas que permita satisfacer la demanda de todos los clientes, cumpliendo las restricciones que impone el problema como son: cada vehículo sale y regresa de un solo depósito, la capacidad máxima de cada depósito no puede ser excedida y todo ello utilizando la menor cantidad de vehículos y al menor costo posible. La naturaleza combinatoria del problema clasifica al mismo como un problema $NP-HARD$. La principal dificultad para la solución del problema radica en el tamaño del problema (dado por la cantidad de clientes). La solución exacta solo puede ser determinada en un tiempo razonable para tamaños del problema por debajo de 50 clientes.
En consecuencia planteamos la siguiente hipótesis:
Plantear alternativas de solución mediante el diseño e implementación de heurísticas que nos permitan en tiempos razonables (pequeños) resolver el problema para tamaños mayores o iguales al máximo tamaño resuelto de manera exacta (50 clientes), sacrificando sin embargo la exactitud de la solución en su lugar tendremos una solución factible cercana (cota superior), la cual además podrá ser refinada o mejorada con la aplicación de otras heurísticas llamadas de mejora.

Debe mencionarse que las heurísticas a desarrollar (tanto de construcción como de mejora) son **originales**, que puedan tener un desempeño similar o mejorar a las existentes en la comunidad académica dedicada a estudiar el problema del ruteo de vehículos.

3.2. Variables e indicadores de la investigación

En cuanto a las variables en la presente investigación, estas serán:

Variable Independiente:

Instancia del problema planteado.

Variable Dependiente:

Costo de la instancia considerada.

Indicadores:

- Cercanía del valor del costo de la instancia obtenida en la variable dependiente respecto al mejor valor conocido BKS.
- Magnitud de la mejora obtenida respecto a la solución inicial que proporciona la heurística de construcción.

3.3. Métodos de la investigación

La metodología utilizada en el presente trabajo es la bibliográfica, científica, inductiva y deductiva. A partir de propuestas existentes en artículos científicos, se construye una propuesta inédita la cual es validada mediante su implementación en el lenguaje de programación JULIA 1.0.5. Se utilizan las instancias existentes para realizar las pruebas, el programa o código pasa por constante refinamiento y depuración para evitar fallos en la fase de ejecución, esta validación se realiza comparando los resultados obtenidos por la heurística propuesta con los de otros propuestos en diferentes artículos de divulgación científica (como por ejemplo en (Shi et al., 2020, hi et al., 2020), (Surekha and Sumathi, 2011, urekha and Sumathi, 2011) y (Stodola, 2018, todola, 2018)) y también comparándolo con la mejor solución conocida a la actualidad. Finalmente para justificar la pertinencia y bondad de las nuevas heurísticas propuestas se elaboran tablas comparativas y gráfico de barras con los resultados obtenidos en promedio.

3.4. Diseño o esquema de la investigación

La investigación parte del conocimiento y estudio teórico del problema y consiste fundamentalmente en proponer una nueva estrategia de solución que esté a la par o sea mejor a las existentes y que sirva de fundamento para futuras mejoras. El camino a seguir está esquematizado en tres partes:

Diseño En esta parte se pone en evidencia la originalidad del trabajo, se idea una clusterización basada en la información preliminar recolectada en cada una de

las instancias. En el proceso de construcción y diseño se irán aplicando estrategias de depuración del algoritmo con la intención de evitar fallos en la fase de implementación.

Implementación En esta fase tomando como base la idea diseñada, se utilizará el lenguaje de programación JULIA version 1.0.5 para implementar el proceso de clusterización, se espera tomarse un tiempo razonable para obtener la experticia en el lenguaje mencionado. El código obtenido debe ser modular y suficientemente documentado para su uso posterior y mantenimiento y/o modificación.

Pruebas En esta fase, se toma como insumo algunas instancias creadas cuyos datos o parámetros ponen a prueba y verifican la correcta funcionalidad del programa obtenido, asimismo el programa debe ser sometido a instancias ya existentes en la literatura y a partir de los resultados obtenidos verificar su bondad. En paralelo se debe realizar la implementación en el lenguaje especializado CPLEX versión 12.6.1 para la constrastación de resultados obtenidos.

3.5. Población y muestra

La **población** a la cual será aplicada el resultado de la investigación, es decir las heurísticas diseñadas para resolver el problema del ruteo de vehículos multidepósito, serán todos los problemas de ruteo de vehículos con múltiples depósitos que pueden ser formulados en alguna actividad logística de distribución de bienes.

Sin embargo las heurísticas diseñadas e implementadas en el presente trabajo serán aplicadas a una **muestra** dado por las 23 instancias (desde $p01$ hasta $p23$) estandarizadas por la comunidad académica que se dedica a estudiar el problema motivo de la siguiente investigación, éstas instancias puede ser encontradas en la página webb `https://github.com/fboliveira/MDVRP-Instances`

3.6. Actividades del proceso de investigación

En esta sección listamos las diferentes actividades cronológicamente ordenadas que se llevaron a cabo durante el proceso de investigación.

- Revisión de la bibliografía especializada en el tema del ruteo de vehículos multidepósito.
- Aprendizaje, capacitación e instalación de los lenguajes de programación JULIA 1.0.5 y CPLEX 12.6.1
- Búsqueda de instancias estandarizadas en la WEBB para el problema MDVRP, junto con sus mejores soluciones.
- Diseño y construcción de estrategias de clusterización.
- Implementación en JULIA 1.0.5 de las Heurísticas de construcción, llamadas heurísticas de clusterización.
- Pruebas de la Implementación con instancias creadas y con instancias existentes y diseñadas para fines de comparación.
- Implementación en JULIA del modelo exacto y pruebas con las instancias creadas y también con las instancias existentes.
- Elaboración del marco conceptual para el tratamiento de la solución exacta vía algoritmos de relajamiento y subgradientes, con el objetivo de obtener cotas inferiores para la solución.
- Implementación de las heurísticas de mejora y su aplicación a cada una de las soluciones obtenidas mediante la heurística de construcción,con sus respectivas pruebas.
- Aplicación del programa diseñado a todas las instancias existentes y elaboración de tablas comparativas con los resultados obtenidos.
- Contrastación de la hipótesis planteada a partir de los resultados obtenidos.
- Elaboración de las conclusiones y recomendaciones.

3.7. Técnicas e instrumentos de la investigación

En esta sección desarrollamos las técnicas e instrumentos matemáticos que nos permitirán contrastar la hipótesis planteada. Asimismo desarrollamos herramientas matemáticas que nos permitirán relajar el modelo exacto para obtener cotas inferiores.

3.7.1. Relajación lagrangiana y dualidad

Consideremos el siguiente problema general de optimización $(\mathcal{P})$

$$
\begin{array}{lll}
 & Minimizar & f(x) \\
(\mathcal{P}) & s.a. & g_i(x) \leq 0 \quad i = 1, 2, \cdots, m \\
 & & h_j(x) = 0 \quad j = 1, 2, \cdots, p \\
 & & x \in S \subseteq \mathbb{R}^n
\end{array}
$$

Siendo $S \neq \phi$ y f función real de variable vectorial, denotaremos al problema $(\mathcal{P})$ como **problema primal**.

Cada una de las m restricciones de desigualdad y cada una de las p restricciones de igualdad son denotadas mediante:

$$
g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{pmatrix} \qquad h(x) = \begin{pmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{pmatrix}
$$

Luego entonces el conjunto $F = \{x \in S \quad / \quad g(x) \leq 0 \quad , \quad h(x) = 0\}$ es la región factible del problema primal $(\mathcal{P})$ y cualquier punto de dicho conjunto se denomina solución factible.

- El conjunto de restricciones $g_i(x) \leq 0$ se denominan restricciones explícitas de desigualdad, mientras que el grupo de resticciones $h_j(x) = 0$ se denominan restricciones explícitas de igualdad.
- La restricción $x \in S$ se denomina implícita.

Debemos notar que en la formulación del problema primal se ha separado el conjunto de restricciones en explícitas e implícitas, precisamente para enfatizar el hecho de que

las restricciones **mas complicadas** son las explícitas y que el problema primal sería mas fácil de resolver sin considerar éstas.

Definición 3.7.1 (Lagrangiano)

Dado el problema primal $(\mathcal{P})$, se define como Lagrangiano a la expresión

$$L(x, \mu, \lambda) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{j=1}^{p} \lambda_j h_j(x)$$

donde $\mu_i \geq 0$ y $\lambda_j \in \mathbb{R}$, además $\mu = [\mu_1, \mu_2, \cdots, \mu_m]$ y $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_p]$ son llamados variables duales o multiplicadores de Lagrange.

Finalmente se define la función Lagrangiana de la siguiente manera:

$$\phi(\mu, \lambda) = \inf_{x \in S} L(x, \mu, \lambda)$$

Definición 3.7.2 (Condiciones globales de Optimalidad)

Sean $\bar{x} \in S$, $\bar{\mu} \in \mathbb{R}^m_+$ y $\bar{\lambda} \in \mathbb{R}^p$, se dice que el punto $(\bar{x}, \bar{\mu}, \bar{\lambda})$ satisface las Condiciones globales de Optimalidad para el problema primal $(\mathcal{P})$ si y solo si se cumplen las siguientes condiciones:

1. $f(\bar{x}) + \bar{\mu} g(\bar{x}) + \bar{\lambda} h(\bar{x}) = \inf_{x \in S} \left\{ f(x) + \bar{\mu} g(x) + \bar{\lambda} h(x) \right\}$
2. $\bar{\mu} g(\bar{x}) = 0$
3. $g(\bar{x}) \leq 0 \quad y \quad h(\bar{x}) = 0$

Enunciamos a continuación un teorema respecto a las condiciones suficientes de optimalidad para el problema primal $(\mathcal{P})$, debemos acotar que este teorema no son de condiciones necesarias.

Teorema 3.1

Si el punto $(\bar{x}, \bar{\mu}, \bar{\lambda})$ satisface las Condiciones globales de Optimalidad, entonces $\bar{x}$ es un óptimo para el problema primal.

Teorema 3.2 (Dualidad Débil)

Si $p^* = \inf_{x \in F} f(x)$, entonces $\phi(\mu, \lambda) \leq p^* \quad \forall\, (\mu, \lambda) \in \mathbb{R}^m_+ \times \mathbb{R}^p$

Denotamos a continuación el problema dual $(\mathcal{D})$ asociado al problema primal $(\mathcal{P})$, mediante

$$(\mathcal{D}) \qquad \begin{array}{ll} Maximizar & \phi(\mu, \lambda) \\ \mu, \lambda & \\ s.a. & \mu \geq 0 \end{array}$$

Denotamos por $d^* = \sup_{\mu \geq 0, \lambda} \phi(\mu, \lambda)$. Por otro lado respecto al problema dual llamaremos **solución factible dual** a cualquier punto (μ, λ) que satisfaga, $\mu \geq 0$ y $\phi(\mu, \lambda) > -\infty$. Nótese que fijado x la función $L(x, \mu, \lambda)$ es lineal con respecto a (μ, λ), en consecuencia es convexa y cóncava, se deduce inmediatamente que $\phi(\mu, \lambda)$ es una función cóncava, asi entonces el problema dual $(\mathcal{D})$ es un problema que consiste en maximizar una función objetivo cóncava sobre una región factible convexa.

El teorema de Dualidad Débil indica que $d^* \leq p^*$ y en el caso que la desigualdad sea estricta, tenemos una brecha dual de magnitud $p^* - d^*$.

3.7.2. Dualidad en programación entera

Debemos notar que en el presente trabajo se ha planteado un problema de Optimización Entera, donde el conjunto S es finito y generalmente la brecha dual es mayor que cero. Además debido a que el conjunto S es finito se tiene que la función $\phi(\mu, \lambda)$ es una función que posee las siguintes particularidades: **cóncava, lineal a trozos y no diferenciable**. Planteado un problema primal, es posible obtener varias formulaciones para el problema dual, dependiendo que restricciones se toman como explícitas y cuales como implícitas, dependiendo de a naturaleza del problema se tiene que realizar una adecuada selección del conjunto S. A continuación se enuncian dos teorema que nos aproximan a un tipo de teorema de dualidad fuerte.

Teorema 3.3

Si el punto $(\bar{x}, \bar{\mu}, \bar{\lambda})$ satisface las Condiciones globales de Optimalidad, entonces el punto $(\bar{\mu}, \bar{\lambda})$ es óptimo para el problema Dual, mas aún $p^* = d^*$.

Teorema 3.4 (del Punto silla)

El punto $(\bar{x}, \bar{\mu}, \bar{\lambda})$ es óptimo si y solo si $\bar{x} \in S$, $\bar{\mu} \in \mathbb{R}^m_+$, y

$$L(\bar{x}, \mu, \lambda) \leq L(\bar{x}, \bar{\mu}, \bar{\lambda}) \leq L(x, \bar{\mu}, \bar{\lambda}) \quad \forall x \in S \ , \ \mu \in \mathbb{R}^m_+ \ , \ \lambda \in \mathbb{R}^p$$

Existe una inherente dificultad para usar estos teoremas en el modelo planteado (Programación Entera) dado que como lo mencionamos en la subsección anterior la brecha dual casi siempre es mayor que cero. Si el punto silla del Lagrangiano no existe se demuestra que resolver el problema dual es equivalente a resolver el problema convexificado la prueba de la afirmacion la encontramos en (ovu, , vu,). La dificultad

del problema convexificado se encuentra en la dificultad de conocer una expresión explícita de la envoltura convexa, se puede utilizar el método de **cortes de Fenchel** en Programación Entera para resolver el problema convexificado.

En el presente trabajo nos limitaremos a resolver el problema dual, eligiendo adecuadamente las restricciones implícitas y explícitas.

3.7.3. Subgradientes y el problema dual

En esta subsección mostramos resultados matemáticos que nos permitirán relacionar la solución del problema dual con el concepto de subgradiente.

Sean $x_0 \in \mathbb{R}^n$ un punto y $f : \mathbb{R}^n \to \mathbb{R}$ una función, se define el conjunto

$$\partial f(x_0) = \{\xi \in \mathbb{R}^n \ / \ f(x) \geq f(x_0) + \langle \xi, x - x_0 \rangle, \forall x \in \mathbb{R}^n\} \tag{3.1}$$

como subdiferencial de f en el punto x_0, mientras que diremos que ξ es un subgradiente de f en x_0, mencionamos asimismo algunas propiedades al respecto.

1. $\partial f(x_0)$ es cerrado y convexo

2. Si $f_1, f_2, \cdots, f_m$ son funciones convexas y $\lambda_1, \lambda_2, \cdots, \lambda_m \geq 0$, entonces

$$\partial \left(\sum_{i=1}^{m} \lambda_i f_i(x) \right) = \sum_{i=1}^{m} \lambda_i \partial (f_i(x))$$

3. Si f es una función cóncava, entonces diremos que ξ es un subgradiente de f en x_0 si $-\xi$ es un subgradiente de la función $-f$ en x_0, es decir si se cumple

$$f(x) \leq f(x_0) + \langle \xi, x - x_0 \rangle, \forall x \in \mathbb{R}^n$$

4. La función f es diferenciable en el punto x_0 con gradiente $\nabla f(x_0)$ si y solo sí $\nabla f(x_0)$ es el único subgradiente de f en el punto x_0

A continuación algunas definiciones respecto a Conos y Cono tangente.

Definición 3.7.3

Sean $X \subseteq \mathbb{R}^n$ y $x_0 \in X$. Se dice que un vector $v \in \mathbb{R}^n$ es una **dirección factible** *de X en x_0 si existe $\bar{\alpha} > 0$, tal que $x_0 + \alpha v \in X$ para todo $\alpha \in [0, \bar{\alpha}]$. En el caso de que X sea convexo, las direcciones factibles de X en x_0 son vectores de la forma $\alpha(v - x_0)$ con $\alpha > 0$ y $v \in X$*

Definición 3.7.4 (Cono)

El conjunto de todas las direcciones factibles de X en x_0 es un CONO que se denota por $F_X(x_0)$

Definición 3.7.5

Sean $X \subseteq \mathbb{R}^n$ y $x_0 \in X$. Se dice que un vector $y \in \mathbb{R}^n$ es una **tangente** *de X en x_0 si o bien $y = 0$ o bien existe una sucesión $(x_n) \subset X$ tal que*

- $x_n \neq x_0 \; \forall n$
- $\lim\limits_{n\to\infty} x_n = x_0$
- $\lim\limits_{n\to\infty} \dfrac{x_n - x_0}{\|x_n - x_0\|} = \dfrac{y}{\|y\|}$

Definición 3.7.6 (Cono Tangente)

El conjunto de todas las tangentes de X en x_0 es un cono llamado **Cono Tangente** *de X en x_0 la cual se denota mediante $T_X(x_0)$*

Proposición 3.5

Sean $X \subseteq \mathbb{R}^n$ no vacío y $x_0 \in X$. Entonces

a) $T_X(x_0)$ es un cono cerrado

b) $cl(F_X(x_0)) \subset T_X(x_0)$

c) Si X es convexo, entonces $F_X(x_0)$ y $T_X(x_0)$ son convexos y además $cl(F_X(x_0)) = T_X(x_0)$

Definición 3.7.7 (Cono Normal)

Sean $X \subseteq \mathbb{R}^n$ y $x_0 \in X$.Se define **Cono Normal** *de X en x_0 al conjunto $N_X(x_0) = cl((T_X(x_0))^*)$, donde $(T_X(x_0))^*$ es el* **Cono Polar** *de $T_X(x_0)$*

Algunas observaciones importantes

- Si $x_0 \notin X$ se tomará como convención $N_X(x_0) = \phi$
- Se dice que X es regular en x_0 Si se cumple $N_X(x_0) = (T_X(x_0))^*$
- Si X es regular en x_0, entonces $T_X(x_0)$ es un cono convexo

- Si x_0 es un punto interior a X, entonces $T_X(x_0) = \mathbb{R}^n$ y $N_X(x_0) = \{0\}$

Proposición 3.6

Si $X \subseteq \mathbb{R}^n$ es un conjunto convexo. Entonces para cada $x_0 \in X$ se tiene que $\xi \in (T_X(x_0))^*$ sí y solo sí $\langle \xi, x - x_0 \rangle \leq 0 \; \forall x \in X$

Finalmente enunciamos el siguiente teorema y un lema que nos permitirán obtener resultados para la resolución del problema dual y el fundamento teórico de la optimización subgradiente que se tratará en la siguiente sección.

Teorema 3.7

Sea X el conjunto de soluciones óptimas, entonces el punto $x \in X$ si y solo si $0 \in \partial f(x) + N_X(x)$ o de manera equivalente $-\partial f(x) \cap N_X(x) \neq \phi$

Lema 3.7.1

Sea $\bar{x}$ una solución óptima para la función Lagrangiana en $(\bar{\mu}, \bar{\lambda})$, esto es, $f(\bar{x}) + \bar{\mu} g(\bar{x}) + \bar{\lambda} h(\bar{x}) = \inf_{x \in S} \{ f(x) + \bar{\mu} g(x) + \bar{\lambda} h(x) \}$. Entonces $\begin{pmatrix} g(\bar{x}) \\ h(\bar{x}) \end{pmatrix}$ es un subgradiente de $\phi(\mu, \lambda)$ en el punto $(\bar{\mu}, \bar{\lambda})$

3.7.4. Optimización subgradiente

En definitiva el problema que enfrentamos tiene como conjunto factible un conjunto discreto en consecuencia se trata de un problema de optimización no diferenciable, luego entonces utilizamos la estrategia llamada **Optimización Subgradiente**, para ello consideremos en general el problema de optimización no diferenciable

$$(NDP) \qquad \begin{array}{ll} Maximizar & f(x) \\ s.a. & x \in X \end{array}$$

Siendo $X \subseteq \mathbb{R}^n$ convexo y no vacío y f una función real de variable vectorial cóncava que en general es no diferenciable, los métodos basados en la gradiente no se pueden utilizar por tanto usaremos los métodos de Optimización Subgradiente, los cuales constituyen una familia de algoritmos de ascenso aproximados, en resumen son una simplificación heurística del algoritmo primal-dual generalizado.

El método consiste en partir del punto x^k con $k \geq 1$, elegir $\xi^k \in \partial f(x^k)$ y calcular la siguiente aproximación mediante:

$$x^{k+1} = P_X\left(x^k + T_k \frac{\xi^k}{\|\xi^k\|}\right) \tag{3.2}$$

siendo

- T_k es la longitud del tamaño de paso en la iteración k $(k > 0)$
- P_X denota a la proyección sobre el conjunto X

En la mayoría de los casos prácticos de interés $X \subseteq \mathbb{R}^n$ cuya estructura permite hallar la proyección de manera muy sencilla como por ejemplo el ortante no negativo $\mathbb{R}^n_+$

Respecto al tamaño de paso, en la literatura podemos encontrar diferentes propuestas, la elección de cada propuesta da lugar a un algoritmo de optimización subgradiente. Mencionamos tres propuestas

a) Elegir T_k tal que $\lim_{k\to+\infty} = 0$ y $\sum_{k=1}^{\infty} T_k = \infty$

b) $T_k = C\rho^k$, siendo $0 < \rho < 1$ y $C > 0$. Si se toman ρ y C suficientemente grandes se garantiza $\lim_{k\to\infty} x^k = x^*$ donde x^* pertenece al conjunto de soluciones óptimas

c) $T_k = \theta \dfrac{f^* - f(x^k)}{\|\xi^k\|}$, siendo $0 < \theta < 2$ y f^* el valor óptimo del problema (NDP). El proceso converge sin necesidad de imponer condiciones adicionales, salvo el valor de f^* que no es sencillo conocerlo de antemano sin embargo se conoce un tratamiento heurístico conocido como heurística de Held, Wolfe y Crowder, mayores detalles sobre la heurística mencionada la podemos ver en (Held, , eld,), en la cual se sustituye el valor de f^* por una cota superior.

Existen otros propuestas como el método **bundle** cuya descripción la podemos encontrar en (Urruty and Lemaréchal, 1996, rruty and Lemaréchal, 1996)

Metodología para la contrastación de la hipótesis planteada

Con el objetivo de contrastar la hipótesis planteada se diseñarán tres heurísticas con el objeto de encontrar una solución factible inicial (no necesariamente la óptima)

cada una de las cuales a su vez será mejorada mediante las heurísticas de mejoramiento. Se presentan tres heurísticas de construcción, la heurística de clusterización por distancias **HD** la cual no alcanzó resultados factibles en general, la heurística de clusterización por distancias y demandas **HDD** y la heurística de clusterización por distancias, demandas y zonas **HDDZ**. La colección de datos fue realizada desde instancias existentes en la comunidad científica dedicada a resolver éste tipo de problemas de ruteo de vehículos. Finalmente se presentan dos heurísticas de mejora una basada en el intercambio local de nodos y arcos y otra basada en partir (split) aquellos clústeres que tienen una configuración que cruza a varios otros clústeres.

Heurísticas de clusterización

3.7.5. La Heurística de clusterización por Distancias (HD)

En este caso tomamos como insumo principal las ubicaciones (coordenadas) de cada uno de los clientes asi como también la de los depósitos, dadas las coordenadas hemos calculado las distancias entre ellos para cada par de clientes. Considerando la cantidad de depósitos construimos tantos clústeres como depósitos existen, para finalmente asignar a cada cluster el depósito más cercano. En ésta heurística no se tiene en cuenta las demandas de los clientes, ni la capacidad máxima de los depósitos, asi como tampoco la capacidad de cada vehículo; todos estos detalles se dejan para el final del proceso.

En cada paso del algoritmo se tiene una determinada cantidad de clústeres (inicialmente cada cliente constituye un clúster, por lo tanto la cantidad de clústeres al inicio es igual a la cantidad de clientes), entonces el algoritmo identifica los dos clústeres más cercanos (considerando la distancia entre dos clústeres como la distancia entre sus centroides) y une a ambos clústeres, constituyendo así un nuevo cluster de mayor tamaño (en cuanto a cantidad de nodos y demanda), el proceso se detiene cuando se ha logrado construir tantos clústeres como depósitos. Luego de considerar las siguientes notaciones:

- $m = |I|$ (cantidad de depósitos)
- $n = |J|$ (cantidad de clientes)

- $\mathcal{C}$: (conjunto finito de clústeres, cada elemento de $\mathcal{C}$ es un conjunto de clientes)
- $d(C_i, C_j)$: distancia entre los clústeres C_i y C_j

se muestra el algoritmo:

Algoritmo 2 (**HD**) Algoritmo de clusterización por distancias

Entrada: Clientes:(ubicación, demandas y cantidad). Depósitos:(ubicación, capacidad, cantidad). Vehículos:(capacidad y cantidad)

Salida: Conjunto de clústeres

1: $\mathcal{C} = \{\{c_1\}, \{c_2\}, \cdots, \{c_n\}\}$ (cada cliente es un cluster)
2: **mientras** $|\mathcal{C}| > m$ **hacer:**
3: Hallar dos índices i_{min} y j_{min} tales que
4: $d(C_{i_{min}}, C_{j_{min}}) = min\{d(C_i, C_j) \ / \ C_i \ , \ C_j \in \mathcal{C}\}$
5: $C_{min} = C_{i_{min}} \cup C_{j_{min}}$
6: $\mathcal{C} = (\mathcal{C} \setminus \{C_{i_{min}}, C_{j_{min}}\}) \cup C_{min}$
7: **fin de mientras**
8: **retorna** $\mathcal{C}$

El resultado de este algoritmo es el conjunto **clusteres** implementado como un vector cuyos elementos son a su vez vectores, cada componente de éste vector es un cluster indizado según su posición dentro del vector, cabe resaltar los procedimientos utilizados SPLICE! APPEND! y PUSH! del lenguaje de programación JULIA 1.0.5 para el manejo dinámico de vectores.

Debe notarse que cada vez que se extrae un cluster del conjunto de Clústeres su longitud disminuye en una unidad y cuando se agrega aumenta en una unidad. De tal manera que en cada paso la cantidad de Clusteres va disminuyendo en una unidad, lo cual garantiza la culminación del algoritmo.

Se aplicará esta heurística a los datos mostrados en la figura (3.1), donde tenemos $nc = 15$ clientes enumerados desde 1 hasta 15, con sus respectivas demandas (los cuales están señalados en la parte superior de los respectivos nodos), por otro lado se tienen $nd = 5$ depósitos (A , B , C , D y E) los cuales para efectos de implementación se enumeran a continuación (en este caso 16; 17; 18; 19 y 20 respectivamente) cuyas capacidades máximas también están señalados en la parte superior de cada depósito

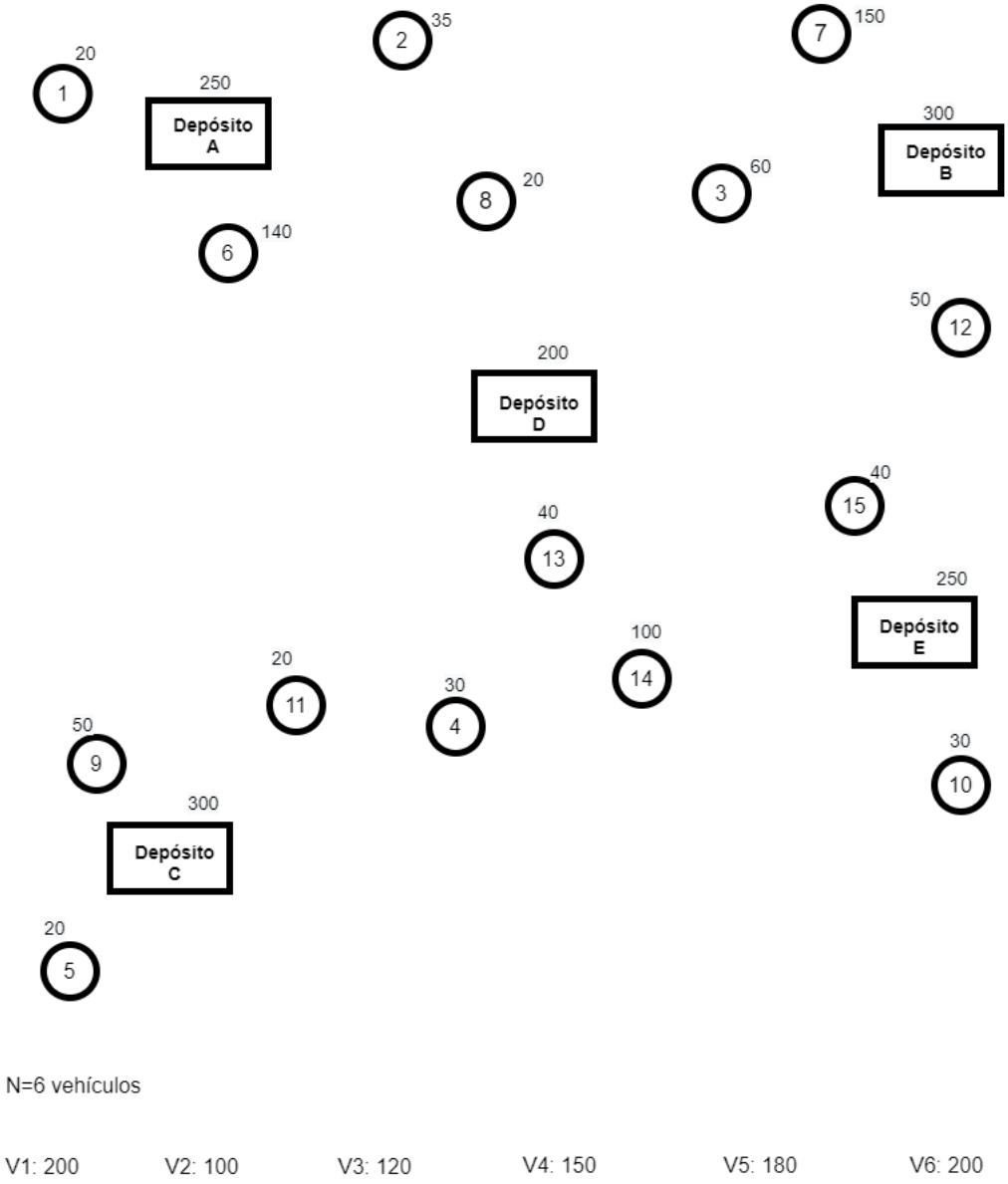


Figura 3.1: Instancia de prueba

Finalmente se dispone de una flota de $N = 6$ vehículos $(V_1, V_2, \cdots, V_6)$ con sus respectivas capacidades máximas de carga, al aplicar el algoritmo antes descrito se obtuvieron 5 clústeres (C_1 , C_2 , C_3 , C_4 y C_5).

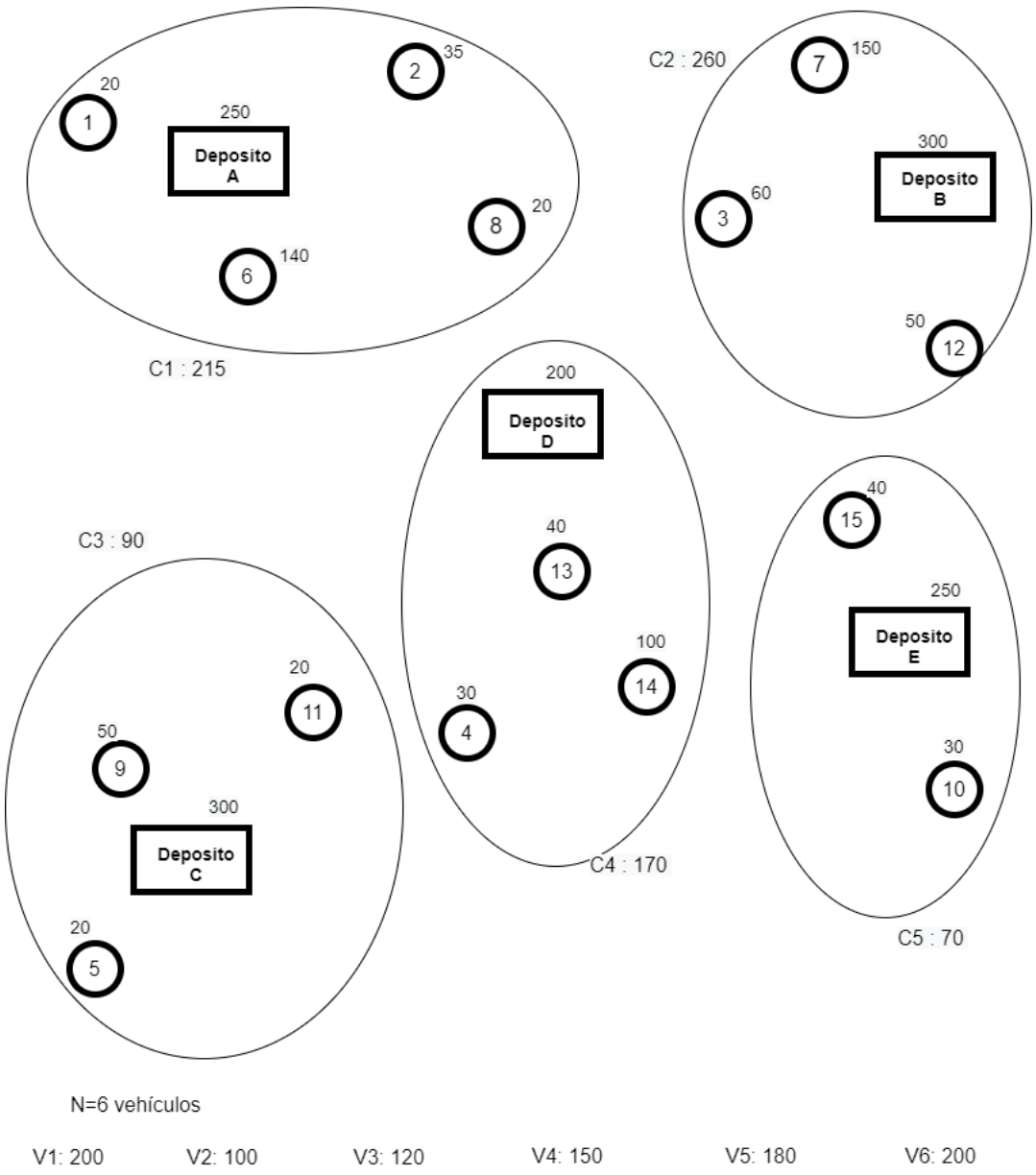


Figura 3.2: Se obtuvieron 5 clusteres (igual al número de depósitos) C_1, C_2, C_3, C_4 y C_5 se muestran las demandas totales de cada cluster formado

Esta heurística presenta un inconveniente, si bien la clusterización de los clientes y depósitos es correcta y muy eficiente (en el sentido de utilizar la mayor capacidad del vehículo), al no tener control sobre la capacidad total de cada cluster obtenido y las capacidades del vehículo o vehículos a asignar podría tornarse complicado o incluso imposible de asignar rutas. Por ejemplo según las demandas totales de los clusters C_1 y C_2 y teniendo en cuenta las capacidades máximas de los vehículos disponibles sería

necesario asignar dos vehiculos a cada uno de dichos clusteres, usando asi 4 vehículos de los 6 en total quedando 3 clústeres por asignar y dos vehículos disponibles los cuales de ninguna manera lograrán satisfacer la demanda de los otros clusteres. Una posible solución seria incrementar la cantidad de vehículos o incrementar la capacidad de cada uno de ellos. Ambas alternativas implican un mayor costo, lo cual va en dirección opuesta al objetivo del problema de optimización que consiste en minimizar costos.

En la siguiente subsección, realizaremos la clusterización pensando también en las futuras rutas, es decir ya asignando de manera eficiente los vehiculos disponibles.

3.7.6. La heurística de clusterización por distancias y demandas (HDD)

Para evitar lo ocurrido con la heurística **HD**, construiremos los clústeres considerando no solo las distancias (cercanías) sino también las demandas y la asignación de vehículos a cada clúster, esta heurística a la que llamaremos **HDD** está compuesto de tres partes: Clusterización, Asignación óptima y TSP, cada una de ellas se describen en las siguientes subsecciones.

Clusterización

Para esta parte de la heurística definimos algunos términos y notaciones que serán utilizados en la construcción de la heurística.

1. Se definen los conjuntos:
   - $\mathcal{C}_{temp}$: que contiene a los clústeres temporales aún en proceso de depuración y construcción
   - $\mathcal{C}_{defi}$: que contiene a los clústeres definitivos.
2. Se dice que la construcción de clústeres temporales **colapsa** cuando se elige dos clústeres más cercanos (cada uno con vehículo asignado) y NO existe vehículo que pueda atender la demanda de la unión de dichos clústeres.
3. Si $V_1, V_2, \cdots, V_t$ son vehículos disponibles para ser asignados al clúster C (cuya demanda total es d_C) y $Q_{V_1}, Q_{V_2}, \cdots, Q_{V_t}$ son las cargas máximas de los respec-

tivos vehículos. El vehículo V_k $(k = 1, 2, \cdots, t)$ se llama **mejor vehículo** si y solo si

$$Q_{V_k} - d_C = \min_{1 \leq i \leq t} \{Q_{V_i} - d_C \ / \ Q_{V_i} > d_C\}$$

Por ejemplo si un clúster C tiene demanda total $d_C = 190$ y se tienen $t = 4$ vehículos disponibles de capacidades 200, 160, 250 y 300, el **mejor vehículo** es el de capacidad 200.

4. Dados dos clústeres con vehículos asignados (C_1, V_1) y (C_2, V_2) y demandas totales conocidas, diremos que (C_1, V_1) es **más saturado** que (C_2, V_2) si $Q_{V_1} - d_{C_1} \leq Q_{V_2} - d_{C_2}$.

5. El control de los vehículos asignados se lleva a cabo a través de la función $v : \mathcal{C} \to \{0, 1\}$, donde $\mathcal{C}$ es el conjunto actual de clústeres y para cada $C \in \mathcal{C}$

$$v(C) = \begin{cases} 1 & ; \quad C \text{ tiene vehículo asignado} \\ 0 & ; \quad \text{en caso contrario} \end{cases}$$

Con respecto a la heurística anterior han sido mejorados varios aspectos, los cuales mencionamos a continuación:

- Los clústeres serán construidos teniendo en cuenta las distancias pero también la posibilidad de asignárseles el **mejor vehículo**.

- La idea principal en la presente heurística es seleccionar los dos clústeres más cercanos, intentar unirlos en un solo clúster y asignarle el **mejor vehículo**. Este proceso repetitivo inevitablemente **colapsa** en la imposibilidad de unir dos clústeres más cercanos con vehículos ya asignados. Al ocurrir este caso se envía el clúster **más saturado** al conjunto $\mathcal{C}_{defi}$ dejando al otro clúster en $\mathcal{C}_{temp}$, este proceso continúa hasta que solo queda un clúster en el conjunto $\mathcal{C}_{temp}$.

- Dependiendo del caso, un vehículo que previamente fue asignado a un clúster podría ser liberado posteriormente. Por ejemplo si el algoritmo selecciona dos clústeres C_i y C_j(más cercanos) cuyas demandas totales hasta el momento son 30 y 80 respectivamente y tienen asignados dos vehículos cuyas capacidades son 100 y 150 respectivamente, entonces los clústeres mencionados se unirán

para constituir un solo clúster de capacidad $30 + 80 = 110$ al cual se le asigna el vehículo de capacidad 150 quedando libre el vehículo de capacidad 100 para que sea utilizado en otro cluster.

El algoritmo selecciona en cada paso los dos clústeres más cercanos C_i y C_j, y dependiendo de los vehículos asignados a dichos clústeres se analizan los siguientes tres casos:

1. **Ambos no tienen asignado un vehículo:** (esto ocurre en los primeros pasos), se escoge el **mejor vehículo** para cada uno o si es posible se une a ambos en un solo cluster y se le asigna el **mejor vehículo** disponible siempre que exista.

2. **Ambos tienen asignado un vehículo:** en este caso se intenta unir ambos clústeres en uno solo con el mejor vehículo escogido entre los dos vehículos que estan siendo usados o algún otro disponible. Cuando no es posible realizar la unión ocurre el **colapso**.

3. **Solo uno de los clústeres tiene un vehículo asignado:** en este caso se intenta aglutinar en el clúster con vehículo todo el contenido del otro clúster, en caso no fuera posible se busca el mejor vehículo disponible para el clúster sin vehículo.

El algoritmo termina cuando hay un solo clúster en el conjunto $\mathcal{C}_{temp}$ el cual pasa a formar parte del conjunto $\mathcal{C}_{defi}$ quedando el conjunto de clústeres temporales vacío ($\mathcal{C}_{temp} = \phi$), el cumplimiento de ésta condición está garantizado dado que en cada paso se eligen los dos clústeres más cercanos y se unen para formar otro cluster de mayor demanda y debido a la capacidad limitada de los vehículos asignados inevitablemente se llegará al **colapso** lo cual obliga a eliminar un clúster de $\mathcal{C}_{temp}$ y enviarlo al conjunto $\mathcal{C}_{defi}$ permitiendo así que la cantidad de clústeres en $\mathcal{C}_{temp}$ disminuya sistemáticamente. Respecto a la entrada del algoritmo se requieren: Clientes:(ubicación, demandas y cantidad); Depósitos:(ubicación, capacidad, cantidad) y Vehículos:(capacidad y cantidad)

Con todas las consideraciones anteriores mostramos el algoritmo:

Algoritmo 3 (HDD) Algoritmo de clusterización por distancias y demandas

Entrada: Clientes, Depósitos y Vehículos

Salida: Conjunto de clústeres $\mathcal{C}$

1: $\mathcal{C}_{temp} = \{\{c_1\}, \{c_2\}, \cdots, \{c_n\}\}$; $\mathcal{C}_{defi} = \{\}$; $v(C) = 0 \; \forall C \in \mathcal{C}_{temp}$
2: **mientras** $|\mathcal{C}_{temp}| > 1$ **hacer:**
3: Sean i_{min} y j_{min} tales que $d(C_{i_{min}}, C_{j_{min}}) = min\{d(C_i, C_j) \; / \; C_i \; , \; C_j \in \mathcal{C}_{temp}\}$
4: **si** $(v(C_{i_{min}}) + v(C_{j_{min}}) = 0)$ **entonces**
5: Juntarlos en un solo clúster y asignarle el mejor vehículo en caso se pueda o asignarle a cada clúster un mejor vehículo
6: **sino**
7: **si** $(v(C_{i_{min}}) + v(C_{j_{min}}) = 2)$ **entonces**
8: **si** ocurre **colapso entonces**
9: $C =$ más saturado entre $C_{i_{min}}$ o $C_{j_{min}}$
10: $\mathcal{C}_{temp} = \mathcal{C}_{temp} \setminus \{C\}$; $\mathcal{C}_{defi} = \mathcal{C}_{defi} \cup \{C\}$
11: **sino**
12: $C = C_{i_{min}} \cup C_{j_{min}}$
13: Asignar el mejor vehículo disponible a C y liberar el otro vehículo
14: $\mathcal{C}_{temp} = (\mathcal{C}_{temp} \setminus \{C_{i_{min}}, C_{j_{min}}\}) \cup \{C\}$
15: **fin de si**
16: **sino**
17: **si** $(v(C_{i_{min}}) + v(C_{j_{min}}) = 1)$ **entonces**
18: $C_1 =$ clúster con vehículo asignado ; $C_0 =$ clúster sin vehículo asignado
19: **si** C_0 puede aglutinarse en C_1 **entonces**
20: $C_1 = C_1 \cup C_0$; $\mathcal{C}_{temp} = \mathcal{C}_{temp} \setminus \{C_0\}$
21: **sino**
22: Asignar el mejor vehículo disponible a C_0
23: **fin de si**
24: **fin de si**
25: **fin de si**
26: **fin de si**
27: **fin de mientras**
28: $\mathcal{C} = \mathcal{C}_{temp} \cup \mathcal{C}_{defi}$
29: **retorna** $\mathcal{C}$

Finalmente los clústeres obtenidos luego del proceso anterior deberán ser asignados de manera óptima a los depósitos para construir las rutas. Este procedimiento se detalla en la siguiente subsección.

Asignación óptima de clústeres a depósitos

Luego de obtener los clústeres con vehículos ya asignados y teniendo toda la información respecto a los depósitos, a continuación debemos asignar los clústeres obtenidos a los depósitos. Eventualmente algunos depósitos podrían quedar sin ningún cluster asignado o un depósito podría albergar a mas de un clúster siempre que su capacidad máxima no haya alcanzado. Para resolver éste subproblema, planteamos un problema de optimización binaria (dado que la variable de decisión solo puede tomar el valor de 0 o 1), para ello consideremos los siguientes parámetros:

p : cantidad de clústeres obtenidos

m : cantidad de depósitos

$i = 1; 2; \cdots ; m$ (índice de depósitos)

$j = 1; 2; \cdots ; p$ (índice de clústeres)

M_{ij} : representa la distancia del depósito i al cluster j

d_{C_j} : es la demanda total del clúster j

D_i : es la capacidad máxima del depósito i

NVD : es la cantidad de vehículos en cada depósito

Consideramos la variable de decisión:

$$x_{ij} = \begin{cases} 1 & ; \quad \text{si el cluster } j \text{ es asignado al depósito } i \\ 0 & ; \quad \text{en otro caso} \end{cases}$$

La función objetivo para este subproblema consistirá en minimizar la distancia entre clústeres asignados y los depósitos, es decir,

$$\text{mín} \left\{ \sum_{j=1}^{p} \sum_{i=1}^{m} M_{ij} x_{ij} \right\}$$

sujeta a las restricciones:

$$\sum_{j=1}^{p} d_{C_j} x_{ij} \leq D_i \quad \forall\, i = 1; 2; \cdots ; m \tag{3.3}$$

Esta restricción asegura que la suma de demandas de los clústeres asignados al depósito i no superan su capacidad.

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall\, j = 1; 2; \cdots ; p \tag{3.4}$$

Esta restricción garantiza que cada clúster es asignado a un solo depósito.

$$\sum_{j=1}^{p} x_{ij} \leq NVD \quad \forall\, i = 1; 2; \cdots ; m \tag{3.5}$$

Esta restricción garantiza que la cantidad de clústeres asignados a cada depósito es menor o igual a la cantidad de vehículos en dicho depósito.

$$x_{ij} \in \{0; 1\} \tag{3.6}$$

Restricción que expresa el tipo de variable.

Esta parte de la clusterización se resuelve de manera exacta a travez de su implementación en el lenguaje de programación JULIA 1.0.5 junto al solver Gurobi, sin ninguna aproximación (heurística) dado que en general el tamaño del problema (expresado en términos de cantidad de clústeres y depósitos) es relativamente pequeña.

Ejecutamos la heurística para la misma instancia de prueba mostrada en la figura (3.1), se puede observar en este caso la obtención de 6 clusteres (color rojo) en total cada uno de ellos con el mejor vehículo ya asignado y satisfaciendo absolutamente las capacidades de los vehículos versus la demanda de clusteres, vea la figura (3.3). A continuación asignamos a cada depósito cierto número de clústeres considerando su cercanía y también la capacidad máxima del depósito, obteniendo clústeres de mayor tamaño (a los que llamaremos superclusteres) que incluyen a los clústeres asignados a determinado depósito y también al depósito. Estos superclusteres (que agrupan a varios clusteres en torno a un mismo depósito asignado) están representados en la figura (3.3) de color celeste.

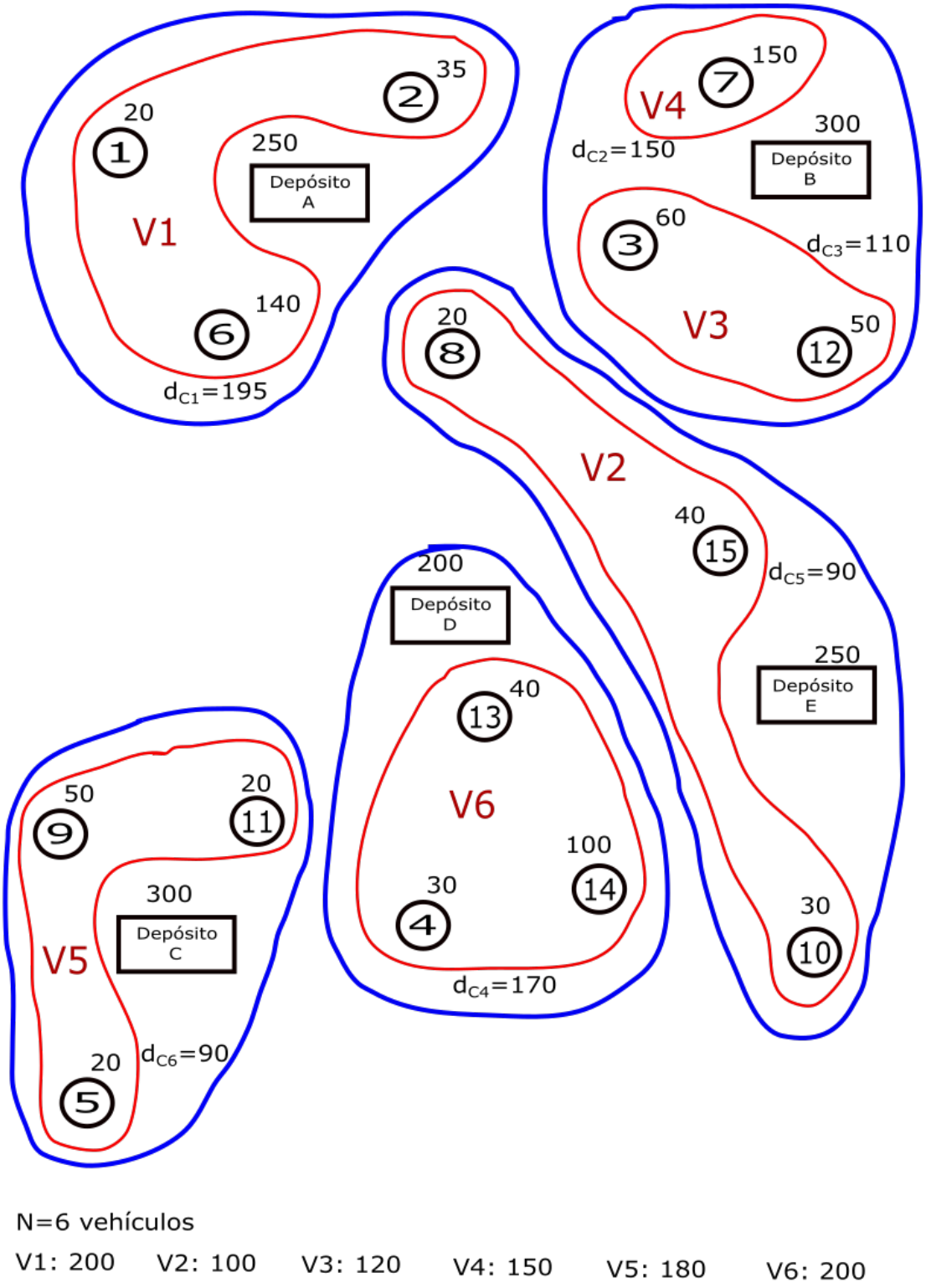


Figura 3.3: Los clusteres de color rojo son los 6 originales de acuerdo a las distancias y demandas además cada cluster ya tiene asignado un vehículo, mientras que los superclusteres de color celeste en general aglutinan a varios clústeres y sus depósitos asignados, eventualmente algún depósito podría quedar sin clústeres asignados.

TSP para definir las rutas

Luego de culminar el proceso de clusterización y la asignación de cada uno de los clústeres a los depósitos, se construyen las rutas resolviendo para cada cluster y su depósito asignado un problema del agente viajero TSP, definiendo la ruta de menor distancia, obteniendo así un conjunto de rutas que constituye una solución inicial factible. La heurística está diseñada para ser aplicada a un problema MDVRP con flota mixta y con restricciones de capacidad máxima para los depósitos. Sin embargo con el objetivo de realizar las pruebas para las instancias definidas en el marco del presente trabajo https://github.com/fboliveira/MDVRP-Instances/ se considera la misma cantidad de vehículos en cada depósito y todos los vehículos con la misma capacidad es decir flota homogénea. Resumimos la heurística **HDD** simbólicamente en la expresión:

$$\text{HDD} = \text{Clusterización} + \text{Asignación Óptima} + \text{TSP}$$

Esta heurística ha sido implementada en el lenguaje de programación JULIA 1.0.5 y se usará una instancia creada manualmente de tamaño pequeño (15 clientes y 5 depósitos) con el claro fin de obtener la solución exacta para efectos de comparación, se muestran además las ubicaciones iniciales de los clientes y depósitos, el resultado de la clusterización por distancias y demandas y la solución exacta para la instancia creada llamada **DataPrueba**

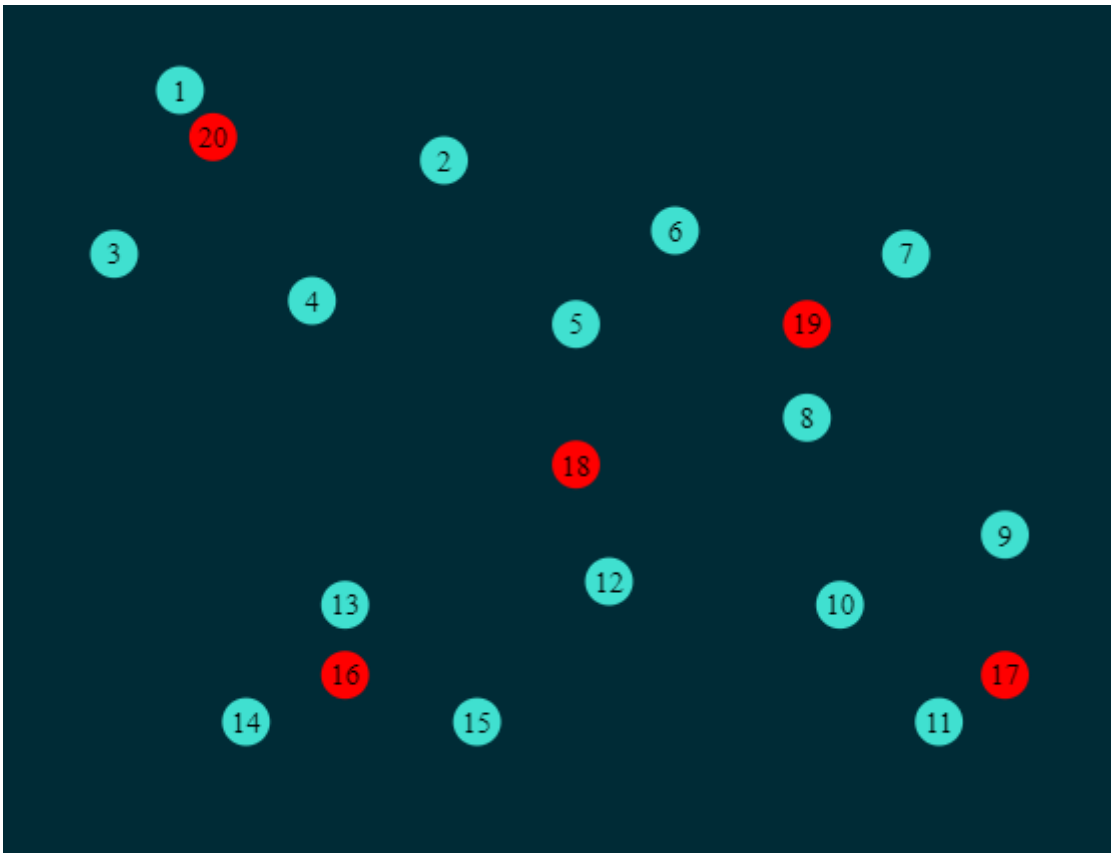


Figura 3.4: Nube de puntos correspondiente a la instancia manual **DataPrueba** con 5 depósitos (pintados de color rojo y enumerados de 16 a 20), 15 clientes (pintados de color turquesa y enumerados de 1 a 15) y dos vehículos cuya carga máxima es 60 en cada depósito.

Al aplicar la heurística por distancias y demandas (HDD) desarrollada en esta sección a la instancia **DataPrueba** se obtiene como resultado la solución que se muestra en la siguiente figura.

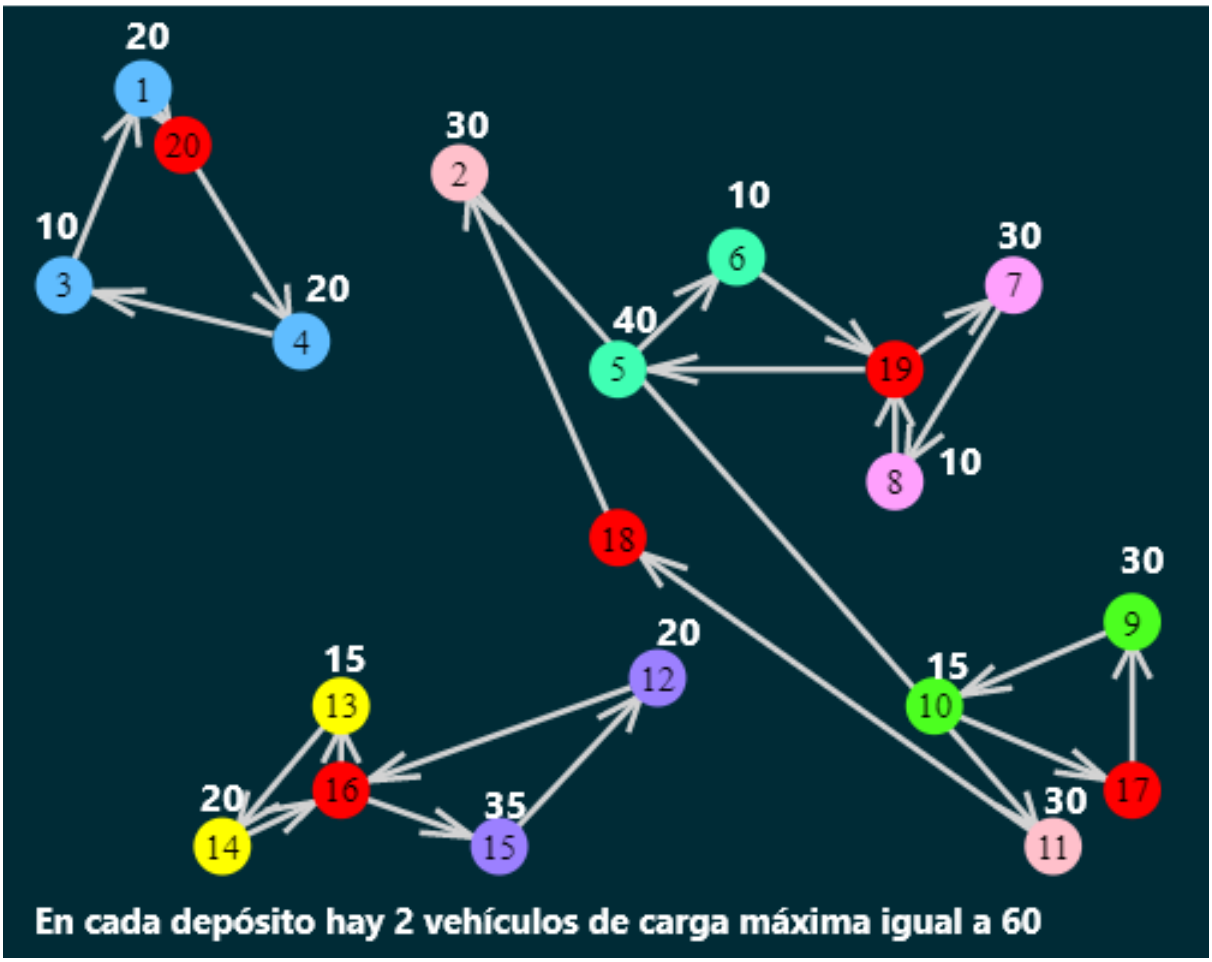


Figura 3.5: Resultados obtenidos por la aplicación de la heurística HDD a la instancia **DataPrueba**, los conjuntos de nodos pintados con el mismo color representan nodos del mismo cluster cuyas demandas serán satisfechas por un vehículo de la flota, los depósitos estan representados con el color rojo, el número mostrado en la parte superior de cada nodo cliente representa su demanda. La función objetivo que resulta de la aplicación de la heurística proporciona el valor **165.15** utilizando 7 vehículos (de la flota de 10 vehículos)

3.7.7. La heurística de clusterización por distancias, demandas y zonas (HDDZ)

Finalmente presentamos una variante de las ideas anteriores que está basado en las ubicaciones de los clientes alrededor de los depósitos, la idea central de la heurística es definir circunferencias alrededor de cada depósito con la particularidad de

que cualesquiera dos circunferencias no se intersecten, esto se garantiza de la siguiente manera, si D_1, D_2, $\cdots$, D_m son los depósitos y d la función distancia euclideana, entonces basta considerar como radio de las circunferencias el valor

$$r = \frac{\min\limits_{1 \leq i,j \leq m} \{d(D_i, D_j)\}}{2} \tag{3.7}$$

Los clientes o nodos que quedan dentro de cada circunferencia o en su frontera automaticamente serán asignados al depósito respectivo. De todas maneras quedarán algunos clientes o nodos sin pertenecer a alguna circunferencia, en este caso cada nodo será asignado a la circunferencia mas cercana. De esta manera todos los nodos fueron asignados a alguna circunferencia.

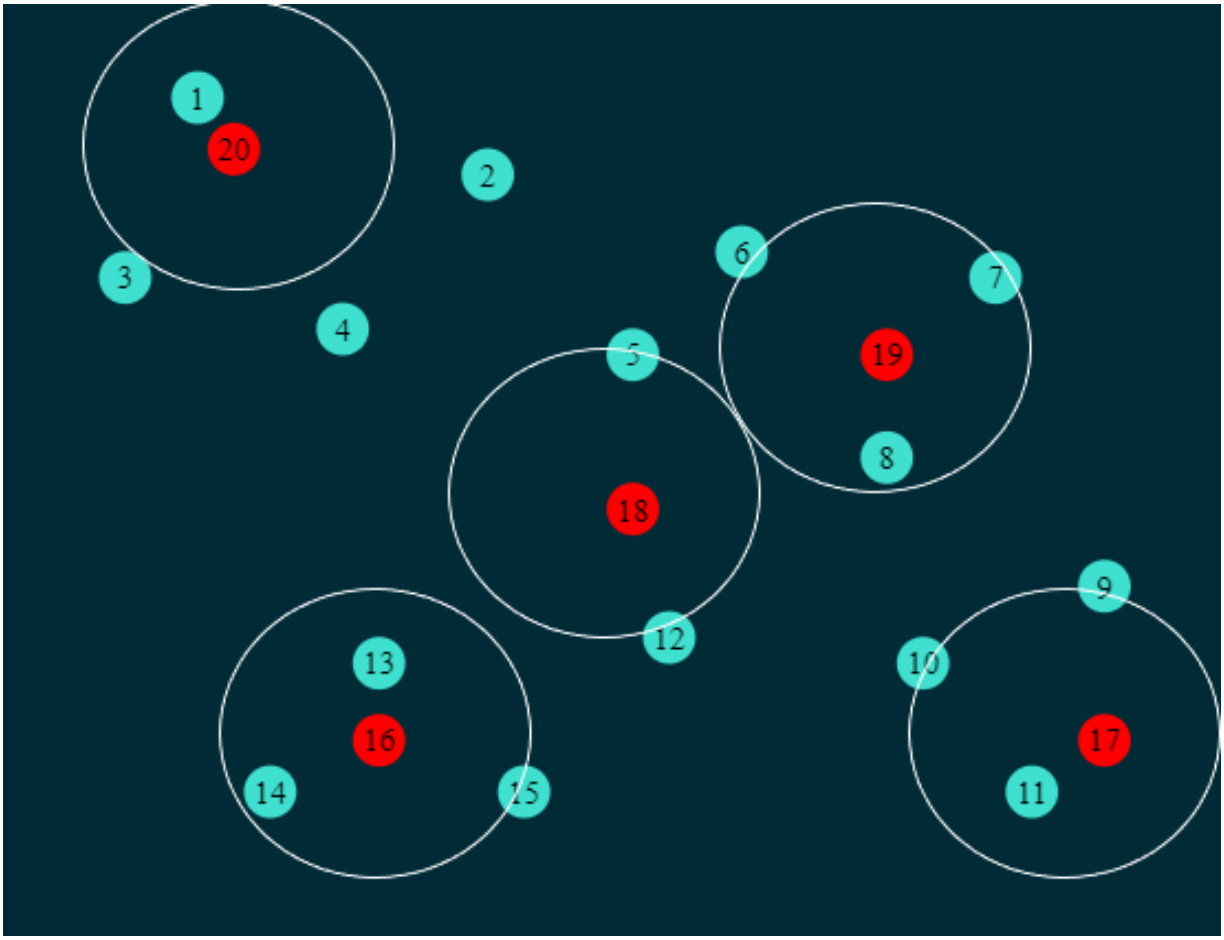


Figura 3.6: Se aplica la heurística descrita a la misma instancia **DataPrueba**. A los nodos que no están dentro o en la frontera de alguna circunferencia como por ejemplo los nodos 3 ; 4 y 2 se les asignará la circunferencia mas cercana, en este caso el depósito 20.

A partir de lo anterior se tiene un grupo de clientes para cada depósito, en este caso las rutas son establecidas solo teniendo en cuenta el vecino mas cercano y la capacidad o carga máxima de cada uno de los vehículos en cada depósito. Se muestra el algoritmo

Algoritmo 4 Algoritmo de clusterización por Distancias, Demandas y Zonas

Entrada: Ubicación y demanda de clientes y ubicación y capacidad de depósitos.

Salida: Conjunto de Clústeres

1: Calcula $r = \dfrac{\min\limits_{1 \leq i,j \leq m} \{d(D_i, D_j)\}}{2}$

2: **for** Cada depósito i ($1 \leq i \leq m$) **hacer:**

3: Sea C_i el grupo de clientes asignado al depósito D_i por estar ubicado dentro de la circunferencia de centro D_i y radio R. Cada C_i es un cluster

4: **end for**

5: Sea C el conjunto de nodos NO asignados a un cluster

6: **mientras** $C \neq \phi$ **hacer:**

7: Extraer un elemento p perteneciente a C

8: Sea C_k ($1 \leq k \leq m$) el cluster tal que $d(p, C_k) = \min\limits_{1 \leq i \leq m} \{d(p, C_i)\}$

9: $C_k = C_k \cup \{p\}$

10: **fin de mientras**

11: Retorna Conjunto de Clústeres $C_1, C_2, \cdots, C_m$

A continuación para cada cluster obtenido se construyen las rutas utilizando un algoritmo voráz que consiste en partir del depósito con un vehículo disponible y dirigirse al nodo o cliente más cercano satisfacer la demanda de dicho cliente y continuar con el siguiente nodo con la misma idea de mas cercano siempre que el vehículo aun puede satisfacer su demanda, en caso contrario el vehículo deberá volver al depósito e intentar construir otra ruta para los nodos aun no ruteados. Se muestra el algoritmo correspondiente

Algoritmo 5 Ruteo de vehículos dentro de cada cluster

Entrada: Clústeres $C_1, C_2, \cdots, C_m$

Salida: Rutas

for Cada cluster $C_1, C_2, \cdots, C_m$ **hacer:**
  mientras (Exista vehículo disponible) y (aun hay clientes no atendidos) **hacer:**
    Tomar un vehículo disponible y partir desde el depósito.
    salir=Falso
    mientras no salir **hacer:**
      Sea p el nodo mas cercano a la ruta trazada
      si El vehículo puede abastecer la demanda del nodo p **entonces**
        Avanzar hacia el nodo p trazar la ruta y actualizar demanda.
      sino
        Regresa al depósito
        salir= Verdadero
      fin de si
    fin de mientras
    Actualizar el conjunto de vehículos
    Actualizar el conjunto de clientes atendidos
  fin de mientras
  si (No existe vehículo) y (aun hay clientes sin atender) **entonces**
    Transferir los clientes no atendidos al cluster mas cercano
  fin de si
end for
Retorna las rutas descritas

Los dos algoritmos anteriores (que forman parte de la heurística descrita en esta sección) fueron implementados en el lenguaje de programación JULIA 1.0.5 obteniendo el siguiente resultado para la instancia de prueba **DataPrueba**, se muestra en la siguiente figura las rutas obtenidas.

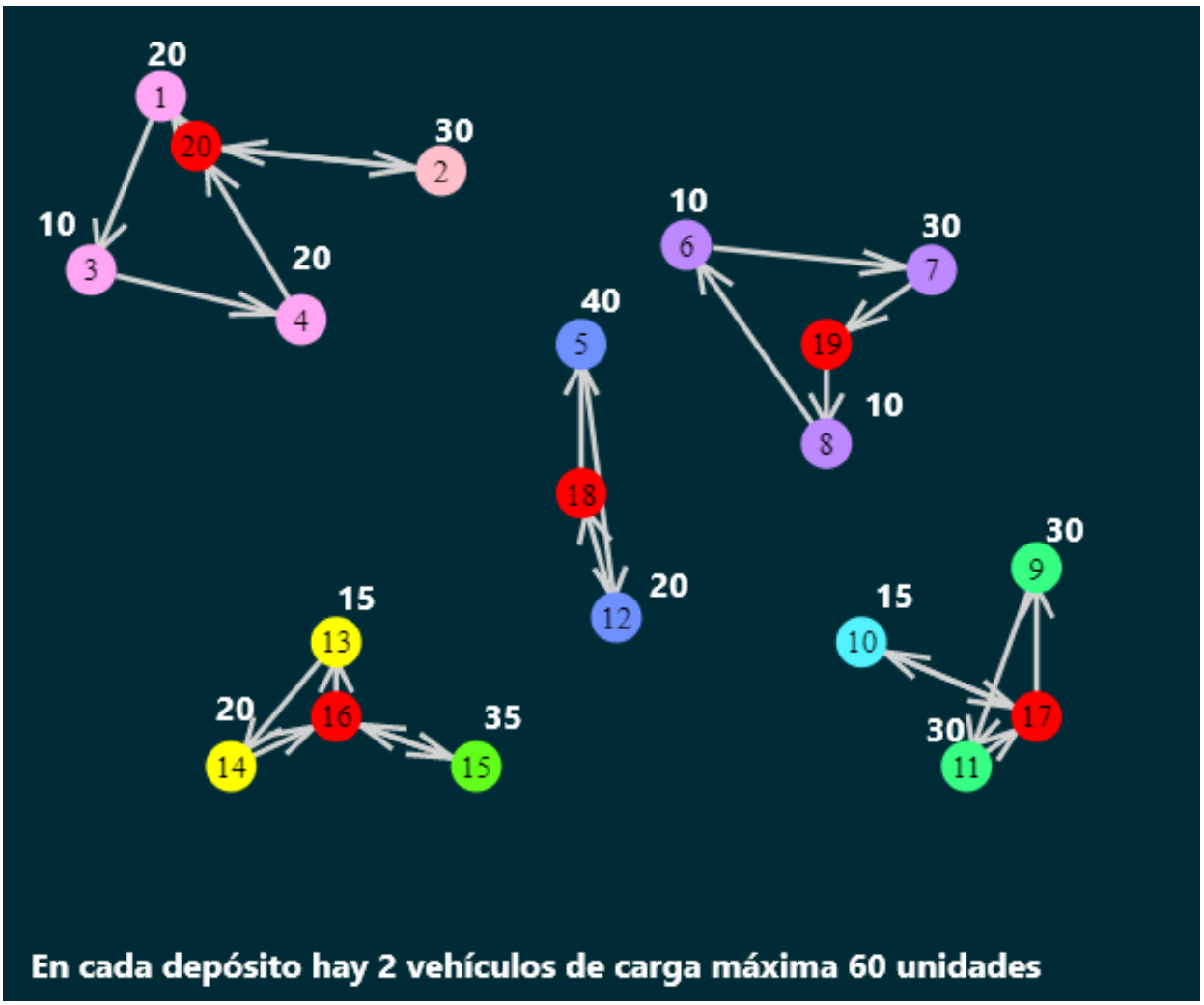


Figura 3.7: Resultados obtenidos con la aplicación de la heurística HDDZ aplicado a la instancia **DataPrueba**, la función objetivo que resulta de la aplicación de la heurística HDDZ proporciona el valor **134.11**(la heurística HDD aplicada a la misma instancia había proporcionado el valor **165.15**) utilizando 8 vehículos (de la flota de 10 vehículos).

Finalmente y para efectos de comparación y contrastación se halla la solución exacta (en este caso es posible hacerlo, dado que el problema tiene tamaño pequeño), el modelo de la solución exacta fue implementado en el lenguaje de programación JULIA 1.0.5, se muestra en la siguiente figura la solución exacta, que vendría a denominarse la clusterización exacta. Las rutas obtenidos (de manera óptima mediante JULIA 1.0.5) definen los clusteres, se observa similitud en alguno de ellos entre la solución por heurística y la solución exacta.

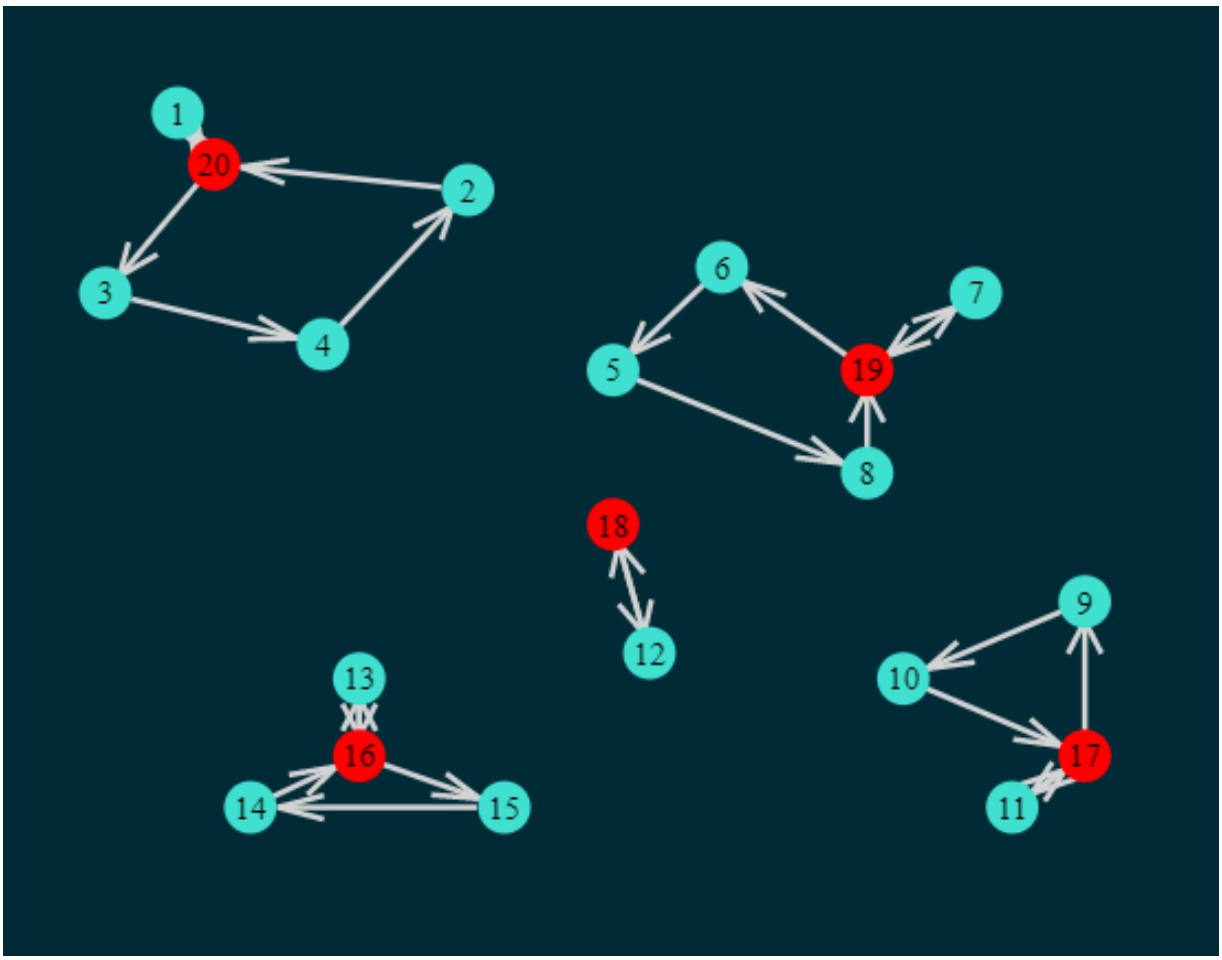


Figura 3.8: Datos mostrados para la misma instancia estudiada y los mismos depósitos y clientes de la figura anterior, los depósitos estan representados de color rojo y sus respectivos clusteres de color turquesa alrededor del depósito, el valor objetivo en este caso el óptimo es **116.70** en este caso se usan 9 vehículos (del total de 10 vehículos)
.

Finalmente elaboramos una tabla comparativa con el desempeño de las heurísticas **HDD** y **HDDZ** para la instancia creada **DataPrueba** la cual tiene $m = 5$ depósitos, $n = 15$ clientes, asimismo otras características de la instancia son la existencia de 2 vehículos por depósito, es decir $TV = 10$ (tamaño de flota) y para ésta instancia en particular (tamaño pequeño) es posible hallar la solución exacta, la cual es $116{,}70$ hallada en 6.33 segundos. Todos estos datos y resultados obtenidos por la ejecución de las heurísticas y el modelo exacto son resumidas en las tablas 3.1 para la solución exacta y 3.2 para la solución aproximada dada por las heurísticas.

solución	costo	Nro de Vehículos	tiempo(seg)
exacta	116.70	9	6.33

Tabla 3.1: Resultados obtenidos para la solución exacta.

Heurística	costo	Nro de Vehículos	tiempo(seg)
HDD	165.15	7	1.09
HDDZ	134.11	8	0.79

Tabla 3.2: Resultados obtenidos por la aplicación de las heurísticas HDD y HDDZ.

Debemos anotar que si bien la heurística HDDZ proporciona mejores resultados que la heurística HDD, el resultado dependerá de la ubicación geométrica de los depósitos, por ejemplo si hay dos depósitos muy cercanos limitará el radio de las circunferencias.

3.8. Procedimiento para la recolección de datos

En el caso de la toma de datos, para la ejecución de las pruebas que permitirán la contrastación la hipótesis planteada, los datos serán tomados de instancias existentes en la WEB como por ejemplo en: `https://neo.lcc.uma.es/vrp/vrp-instances/multiple-depot-vrp-instances/` llamadas las **instancias de CORDEAU** o en el siguiente enlace `https://github.com/fboliveira/MDVRP-Instances`. Estas instancias son utilizadas por toda la comunidad académica que se dedica a estudiar el problema MDVRP, existen además las mejores soluciones encontradas hasta el momento, BKS (Best Know Solution) la cual usamos como referencia en las pruebas numéricas. La comunidad académica mencionada se encarga de estudiar este tipo de modelos ya han aplicado sus algoritmos o heurísticas y nos permitirán validar la heurística que estamos proponiendo mediante la comparación de los resultados. Estas instancias aparecen en un formato específico y estandarizado (archivo texto) a partir de la cual se extrae datos respecto a cantidad de depósitos, clientes y vehiculos (en cada depósito y en total), ubicación de clientes y depósitos(mediante sus coordena-

das cartesianas), demanda de clientes, máxima capacidad de vehículos y capacidad máxima de depósitos. Existen un total de 33 instancias separados en dos grupos, tal como se decribe a continuación:

Instancias tipo p: Estas instancias tienen la particularidad de presentar datos enteros en las coordenadas, se tiene un total de 23 instancias

$$p01 \quad , \quad p02 \quad , \quad p03 \quad , \quad p04 \quad , \quad \cdots \quad , \quad p21 \quad , \quad p22 \quad , \quad p23$$

Instancias tipo pr: En este caso tenemos 10 instancias con la particularidad de que las coordenadas tienen valores reales,

$$pr01 \quad , \quad pr02 \quad , \quad \cdots \quad , \quad pr09 \quad , \quad pr10$$

Se muestra a continuación el contenido de una de las instancias mencionadas (la instancia $p01$), asimismo se describe las particularidades del archivo.

```
 2  4  50  4
 0  80
 0  80
 0  80
 0  80
 1  37  52  0   7  1  4  1  2  4  8
 2  49  49  0  30  1  4  1  2  4  8
 3  52  64  0  16  1  4  1  2  4  8
 4  20  26  0   9  1  4  1  2  4  8
 5  40  30  0  21  1  4  1  2  4  8
 6  21  47  0  15  1  4  1  2  4  8
 7  17  63  0  19  1  4  1  2  4  8
 8  31  62  0  23  1  4  1  2  4  8
 9  52  33  0  11  1  4  1  2  4  8
10  51  21  0   5  1  4  1  2  4  8
11  42  41  0  19  1  4  1  2  4  8
12  31  32  0  29  1  4  1  2  4  8
```

```
13  5 25 0  23 1 4 1 2 4 8
14 12 42 0  21 1 4 1 2 4 8
15 36 16 0  10 1 4 1 2 4 8
16 52 41 0  15 1 4 1 2 4 8
17 27 23 0   3 1 4 1 2 4 8
18 17 33 0  41 1 4 1 2 4 8
19 13 13 0   9 1 4 1 2 4 8
20 57 58 0  28 1 4 1 2 4 8
21 62 42 0   8 1 4 1 2 4 8
22 42 57 0   8 1 4 1 2 4 8
23 16 57 0  16 1 4 1 2 4 8
24  8 52 0  10 1 4 1 2 4 8
25  7 38 0  28 1 4 1 2 4 8
.........................
.........................
.........................
48 25 55 0  17 1 4 1 2 4 8
49 48 28 0  18 1 4 1 2 4 8
50 56 37 0  10 1 4 1 2 4 8
51 20 20 0   0 0 0
52 30 40 0   0 0 0
53 50 30 0   0 0 0
54 60 50 0   0 0 0
```

Para ésta instancia en particular se tiene la siguiente información:

- En la primera linea extraemos información respecto a: el tipo de problema MDVRP (2), número de vehículos en cada depósito (4), número de clientes (50) y número de depósitos (4)
- En las siguientes cuatro lineas tenemos información de la capacidad máxima de cada uno de los tipos de vehículos en cada depósito.
- En las siguientes lineas (enumeradas del 1 al 50) tenemos en las segundas y ter-

ceras columnas las coordenadas (x, y) de cada uno de los clientes y en la quinta columna la respectiva demanda del cliente, los otros valores se usan en otro tipo de modelo.

- En las últimas 4 lineas tenemos información respecto de los 4 depósitos en cuanto a su ubicación geografica, es decir sus coordenadas (x, y) (segunda y tercera columna), observe que en este caso no hay demandas pero tampoco información respecto a la capacidades máxima, es decir asumiremos que hay una capacidad ilimitada.

Si se quiere aplicar los resultados obtenidos (es decir la Heurística) a un caso concreto, se tiene que levantar información respecto a todos los parámetros que se requieren, ubicaciones de clientes y depósitos, demandas, capacidad máxima de los vehículos y depósitos, asi como el tipo de flota mixta u homogenea. Pero este caso está fuera del alcance del presente trabajo tal como se mencionó al formular las hipótesis.

3.9. Técnicas de procesamiento y análisis de los datos

3.9.1. Análisis e interpretación de datos

Obtenida una solución factible inicial mediante las heurísticas de construcción desarrolladas en la sección anterior **Heurística por demandas y distancias** y **Heurística por demandas, distancias y zonas**. El siguiente propósito será mejorar la solución inicial obtenida (que por cierto es un mínimo local), con el objetivo de abandonar dicha solución local y acercarse lo mas que pueda al mínimo global. Para ello en la siguiente sección definimos y desarrollamos las heurísticas de mejora.

3.9.2. Heurísticas de mejora

Una vez que se ha obtenido la solución inicial (que viene a ser una solución factible) mediante la heurística de construcción descrita anteriormente, se aplicará a la solución obtenida otra heurística llamada de mejora. Dentro de éstas se ha considerado las siguientes estrategias:

Intercambio de nodos

El proceso consiste en identificar dos clusteres C y T del conjunto total de clusteres (por heurística de construcción) que tienen la propiedad de ser los mas cercanos. Sea C_i el $i-$ésimo cluster obtenido $(i = 1; 2; 3; \cdots ; NV)$, es decir existen NV clusteres y en consecuencia se utilizarán NV vehículos del total de vehículos TV ($NV \leq TV$), entonces se encuentra bien definida $d(C_i, C_j)$ que representa la distancia entre los clústeres C_i y C_j donde d es la distancia euclideana, considerando para cada cluster su centroide (en el sentido geométrico). Si consideramos el problema matemático

$$\text{mín}\{d(C_i, C_j) \ : \ i, j \in 1; 2; 3; \cdots ; NV \quad i \neq j\}$$

Entonces los clusteres C y T han sido seleccionados de tal manera:

$$d(C, T) = \text{mín}\{d(C_i, C_j) \ : \ i, j \in 1; 2; 3; \cdots ; NV \quad i \neq j\} \tag{3.8}$$

Sean $p \in C$ y $q \in T$ los nodos (clientes) mas cercanos, luego es posible realizar las siguientes operaciones:

Cambio 1: El nodo p pasa a formar parte del cluster T, el cluster C contiene un nodo menos, las rutas cambian ligeramente.

Cambio 2: El nodo q pasa a formar parte del cluster C, el cluster T contiene un nodo menos, las rutas cambian ligeramente.

Intercambio: Ambos nodos se intercambian, es decir al final p pasa a formar parte del cluster T y de manera simultánea q pasa a formar parte del cluster C, ambos clusteres mantienen la misma cantidad de nodos, ambas rutas cambian también ligeramente

Con el fin de ilustrar las operaciones mencionadas, supongamos dada una solución factible, en particular centramos nuestra atención en el par de clusteres C y T siendo los mas cercanos y a la vez los nodos p y q el par de nodos mas cercano.

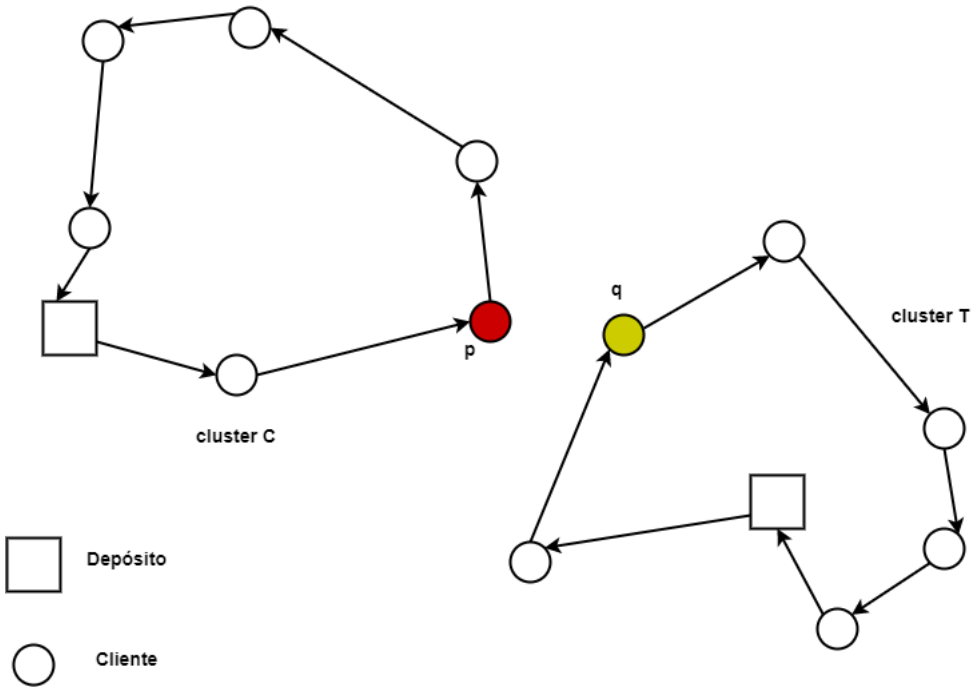


Figura 3.9: Configuración inicial del par de clusteres C y T con los nodos p y q siendo los mas cercanos

En primer lugar veamos el efecto de la aplicación de la heurística de mejora llamada **cambio 1**

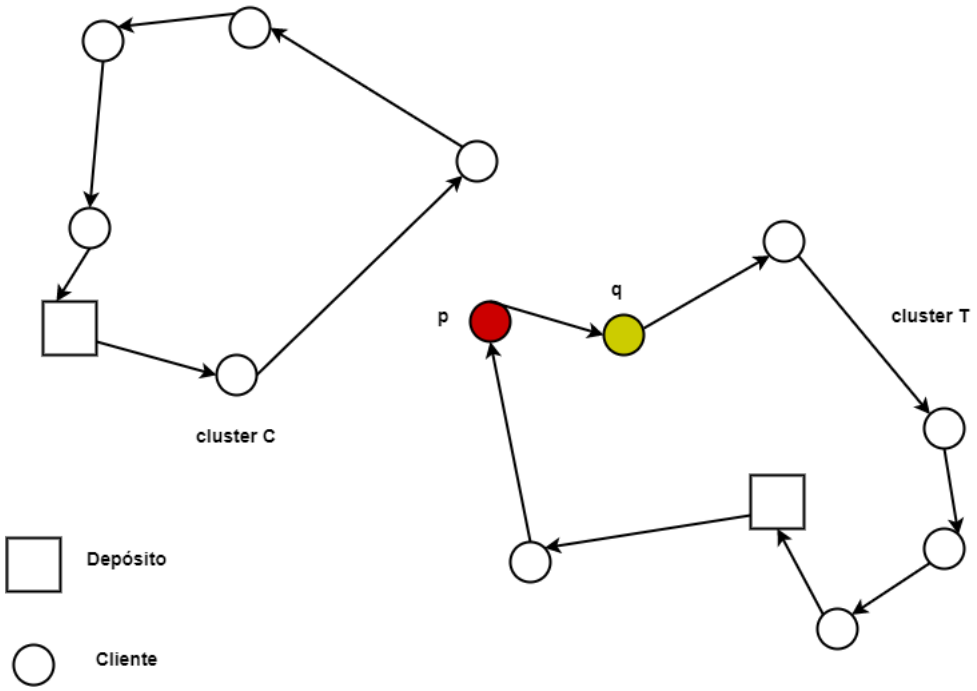


Figura 3.10: El nodo p ahora pertenece al cluster T

Asimismo, a continuación veamos el efecto de aplicación de la heurística de mejora **cambio 2**

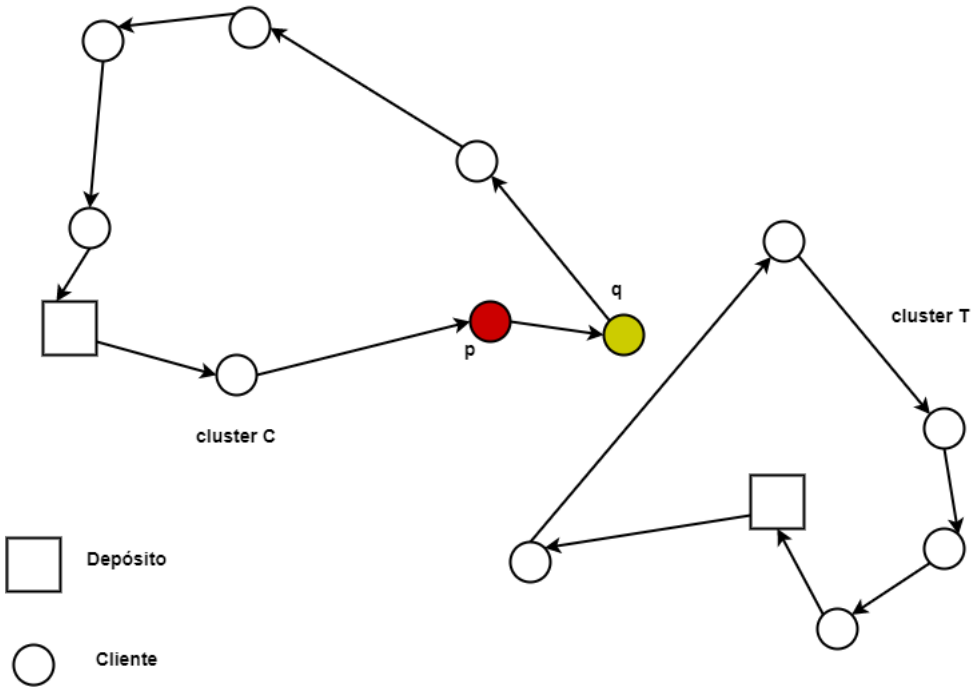


Figura 3.11: El nodo q ahora pertenece al cluster C

Debemos tener en cuenta que tanto la operación **cambio 1** como la operación **cambio 2** no son efectuadas arbitrariamente, sino bajo el cumplimiento de las siguientes condiciones:

- La suma de costos (o distancias) de los clusteres originales C y T es estrictamente mayor que la suma de costos o distancias de los clusteres C y T luego de realizar la operacion, en caso esta condicion no se satisfaga no se realiza el cambio.
- Cuando el nodo p o q según sea el caso pasa a formar parte del otro cluster se debe tener en cuenta que la capacidad máxima del vehículo aún no ha sido alcanzada.

Finalmente veamos el efecto de la operación intercambio, la cual también se realiza solo si las condiciones mencionadas lineas arriba se satisface.

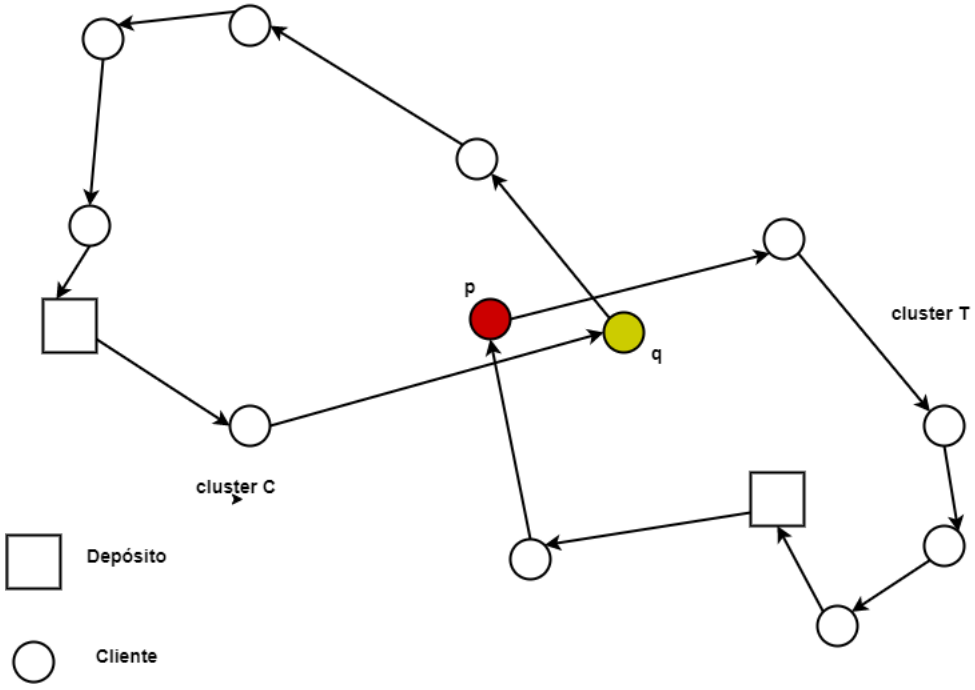


Figura 3.12: El nodo q ahora pertenece al cluster C y el nodo p ahora pertenece al cluster T

Existen otras estrategias de mejora, en el presente trabajo solo se han aplicado las heurísticas de mejora mencionadas anteriormente. También existe la heurística de cambio o intercambio de arcos, se ha optado en el presente trabajo aun no utilizarla.

En la siguiente subsección se desarrolla la heurística de mejoramiento que consiste en **partir** un cluster o ruta, llamado también split.

Split

Esta estrategia consiste en redistribuir los nodos que pertenecen a un cluster con la característica de tener uno, dos o tres (es un criterio tomado por el autor) nodos muy alejados respecto al centroide del cluster. La idea básica es tomar un cluster que tiene un grupo de nodos muy alejado del centroide y entonces formar un subcluster que deberá ser anexado a otro cluster cercano, siempre que las condiciones de demanda sean satisfechas en general que la factibilidad de la solución se mantenga. En este caso el grupo de nodos es incorporado a otro cluster, el cluster mencionado es aquel cuyo centroide se encuentra mas cerca del grupo de nodos considerado. Supongamos que se tiene los clusteres cercanos C y T, donde el cluster C contiene dos nodos p y q adyacentes que tienen la particularidad de estar muy alejados, tal como se muestra en la figura.

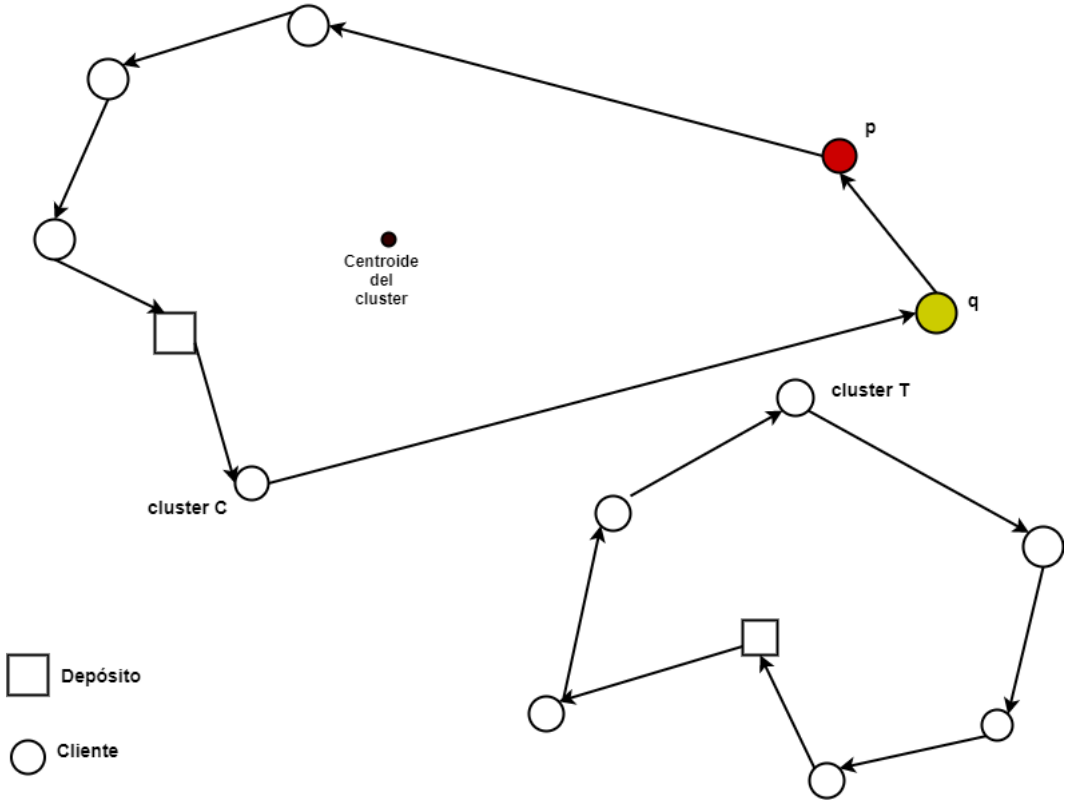


Figura 3.13: El cluster C contiene dos nodos p y q alejados del centroide

En este caso se incorpora ambos nodos al cluster T que se supone es el mas cercano a los nodos p y q visto como un conjunto. Obteniendo la siguiente configuración para los clusteres C y T.

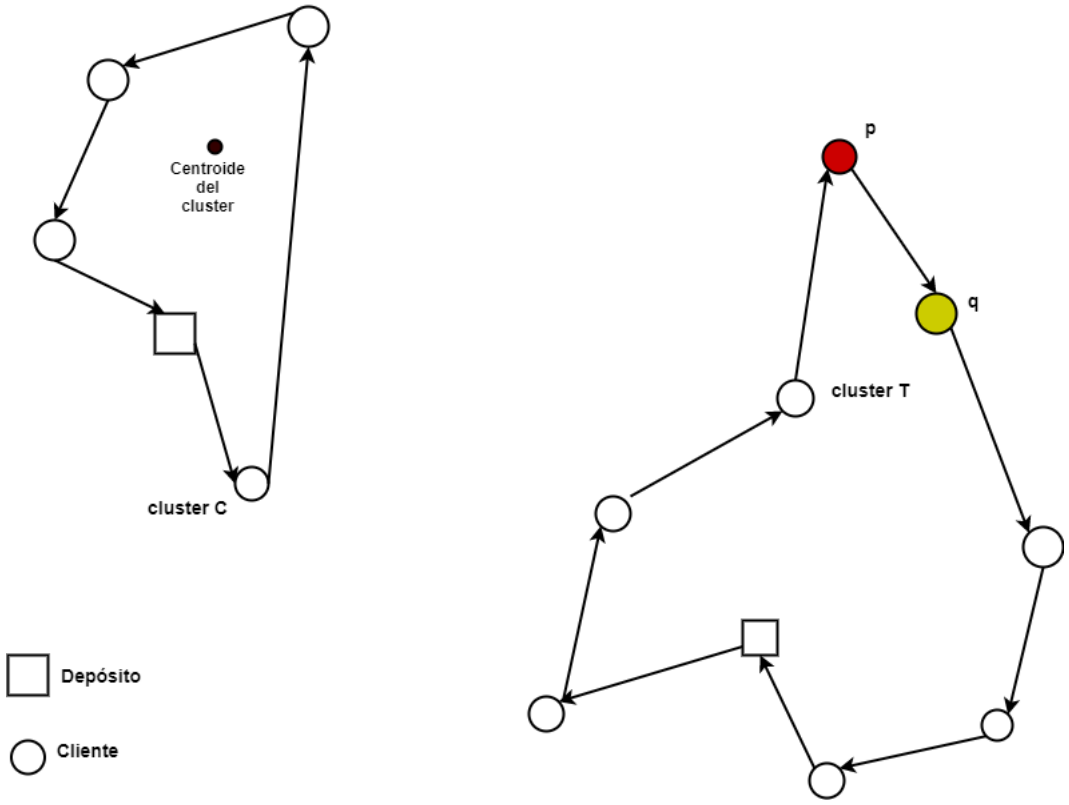


Figura 3.14: Nueva configuración de los clusteres C y T, se observa mayor homogeneidad

En este caso también los cambios están sujetos al cumplimiento de las condiciones antes mencionadas: es decir se debe mejorar la función objetivo cuando los nodos p y q se incorporan al cluster T debe satisfacerse la condición de demanda.

En términos de programación estas estrategias de mejora fueron implementadas en el lenguaje JULIA, para cada instancia hay efectivamente una secuencia de mejoras de la solucion inicial aplicando iterativamente las estrategias antes indicadas. Se debe mencionar que la secuencia de mejoras consume un tiempo adicional la cual tambien fue consignada en las tablas (4.1 y 4.3). Por ejemplo para la instancia p01 cuya solución inicial obtenida se muestra en la siguiente figura, las rutas estan indicadas por colores y se está señalando la ruta o cluster de color amarillo que es la que sufrirá

modificación por la aplicación de UNA iteración del procedimiento de mejora.

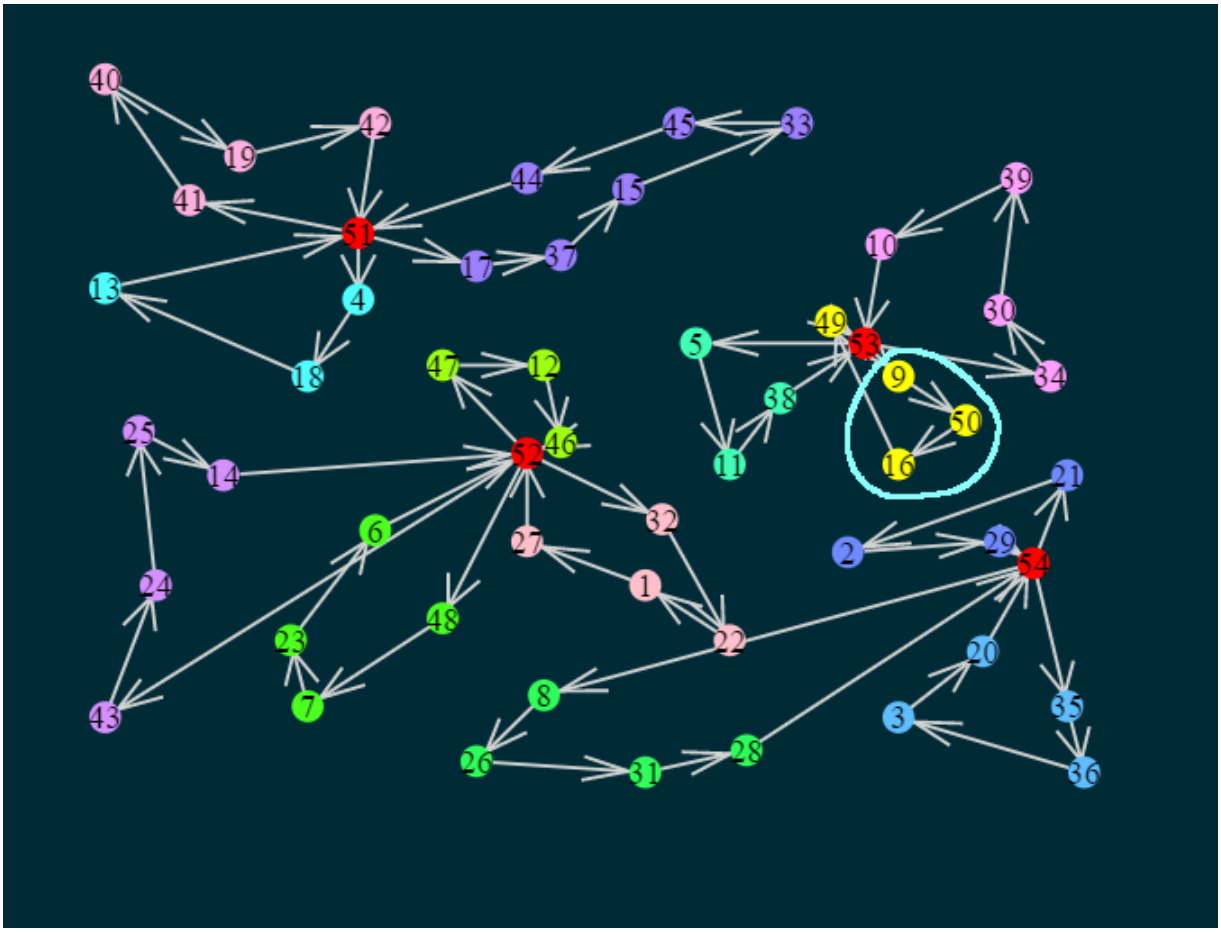


Figura 3.15: Instancia p01 con 50 clientes, 4 depósitos, 4 vehículos por depósito c/u con capacidad 80. Se usan 13 vehículos de un total de 16. Tiempo utilizado = 0.86 segundos y valor objetivo = 643.69

Luego de aplicarle el procedimiento de mejora (UNA vez) se obtiene una mejor solución, cabe mencionar que se la heurística intenta aplicar cualquiera de los procedimientos de mejora (4 en total), se aplica una o una combinación de ellas siempre bajo la condición de mejorar la función objetivo, la aplicación de una mejora se muestra en la siguiente figura:

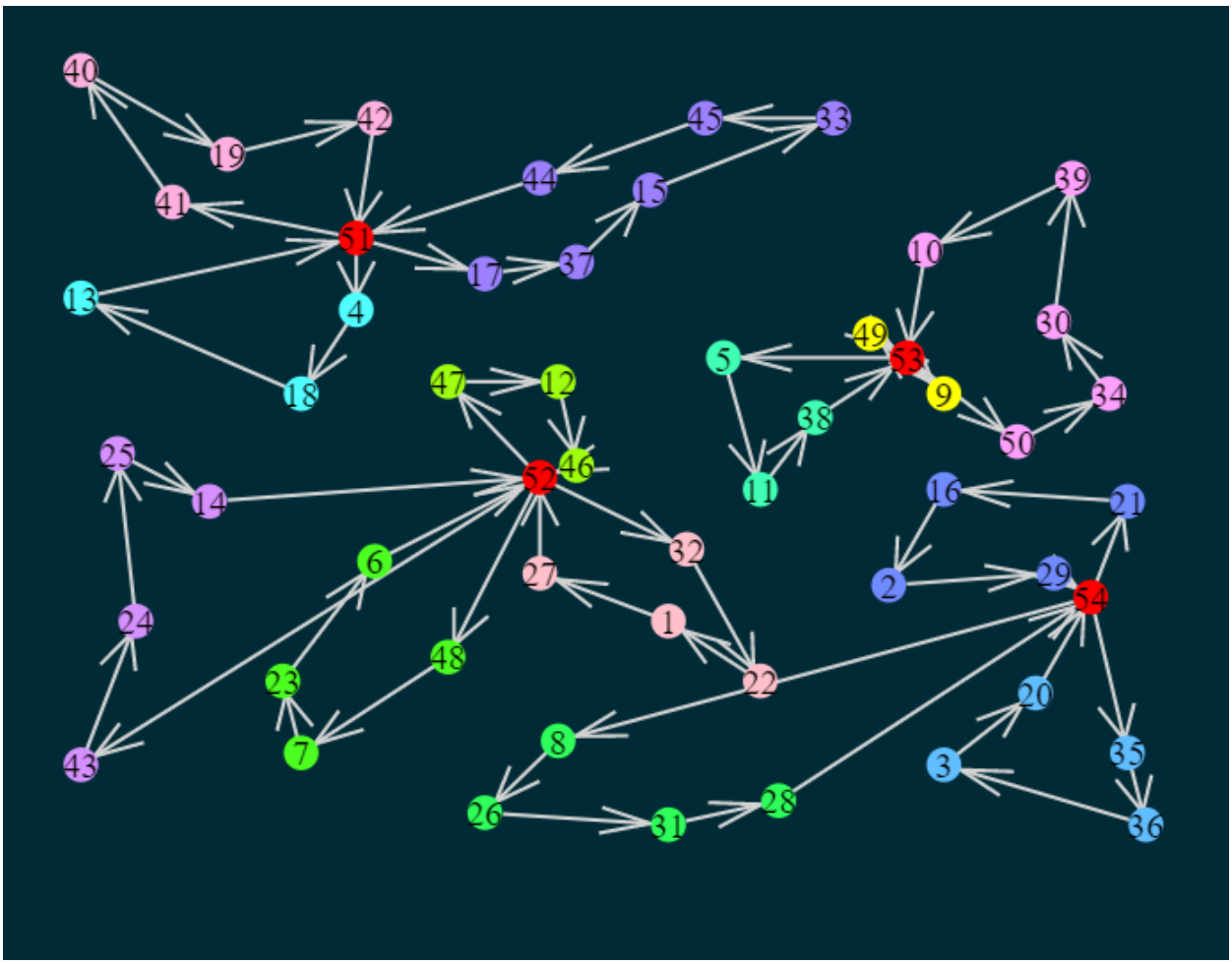


Figura 3.16: Instancia p01 con 50 clientes, 4 depósitos, 4 vehículos por depósito c/u con capacidad 80. Valor objetivo mejorado = 633.237

La aplicación sucesiva de la heuristica de mejora permite encontrar una mejor solución consumiendo un tiempo adicional pero ya una mejora significativa. Se muestra a continuacion la secuencia de mejoras para la misma instancia dada.

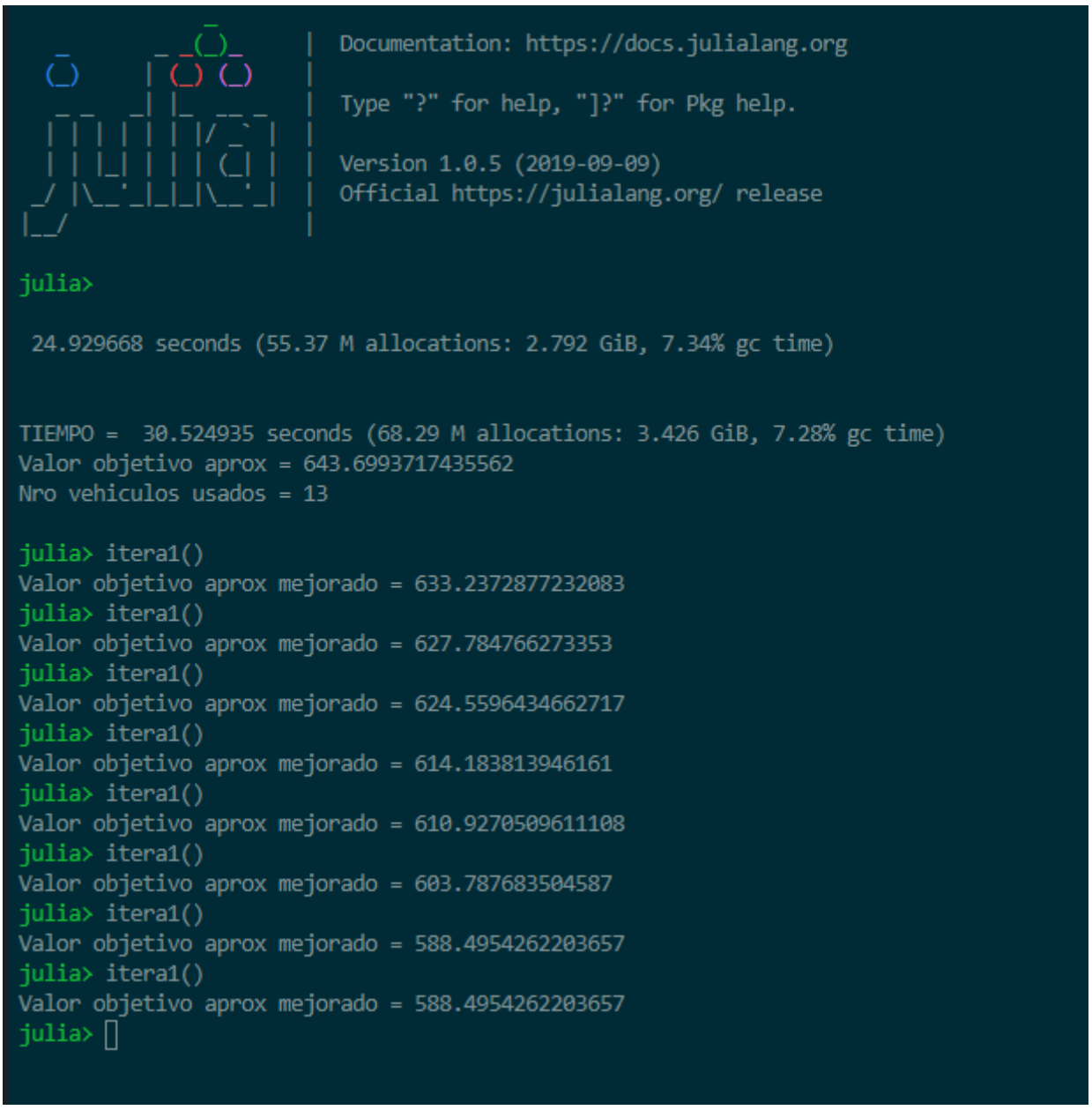


Figura 3.17: secuencia de mejoras aplicada a la instancia p01. En este caso fueron aplicadas 7 veces la operación de mejora. El procedimiento de mejora se realiza mientras dos mejoras sucesivas sean diferentes.

Luego de agotar todas las opciones de mejora se obtiene la mejor solución mediante la heurística de clusterización y adicionalmente la aplicacion de la heurística de mejora. La configuración obtenida se muestra en la siguiente figura.

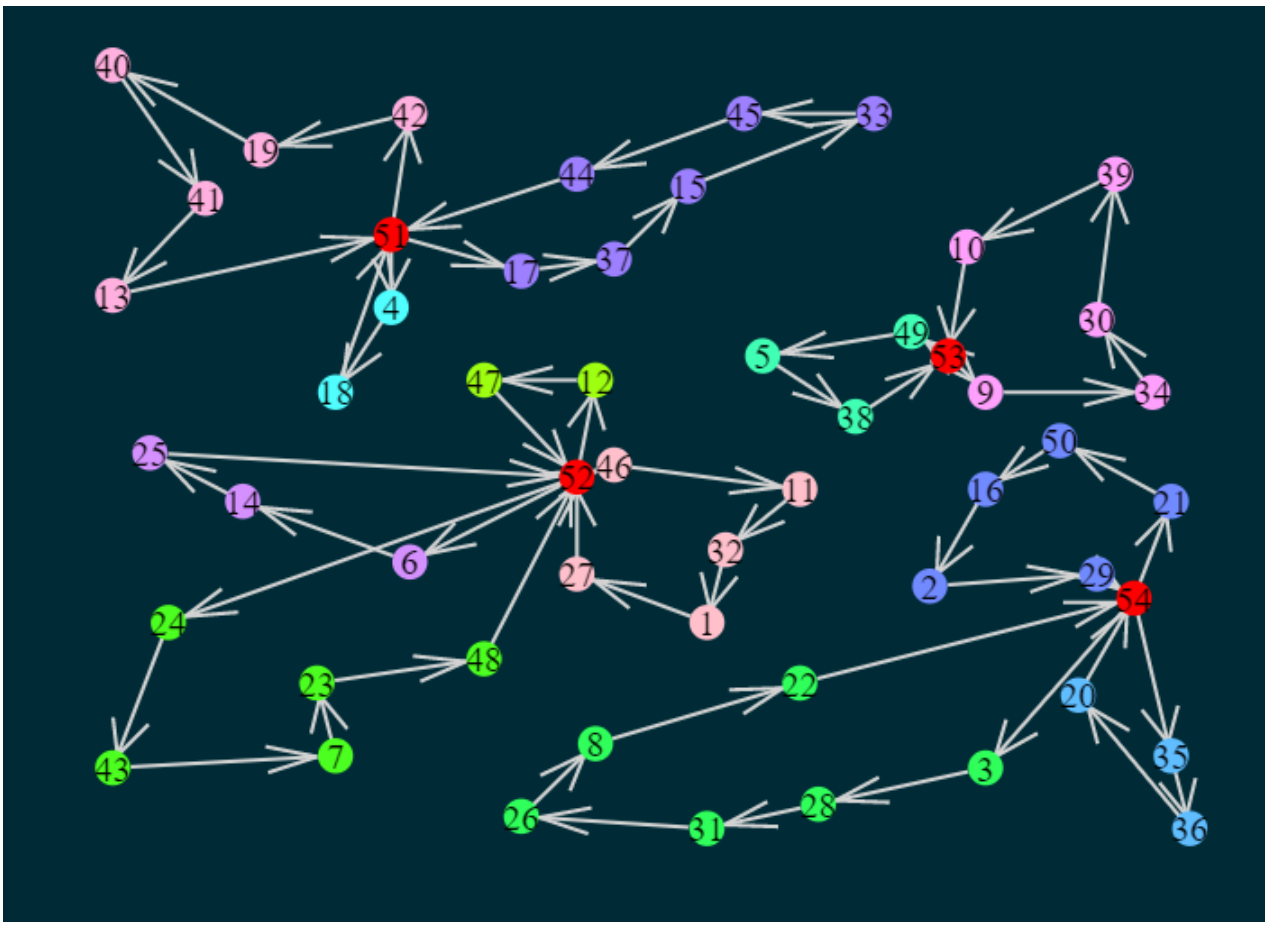


Figura 3.18: Instancia p01 con 50 clientes, 4 depósitos, 4 vehículos por depósito c/u con capacidad 80. Luego de aplicar la heurística de clusterización por **demandas y distancias** y las respectivas mejoras (6 en total). Se mejoró el valor objetivo desde desde el valor inicial **643.69**(2.48 seg) hasta el valor mejorado **588.49**(17.78 seg) utilizando 12 vehículos de un total de 16

De la misma manera se aplicó la Heurística de clusterización por **demandas, distancias y zonas** obteniendo la siguiente configuración.

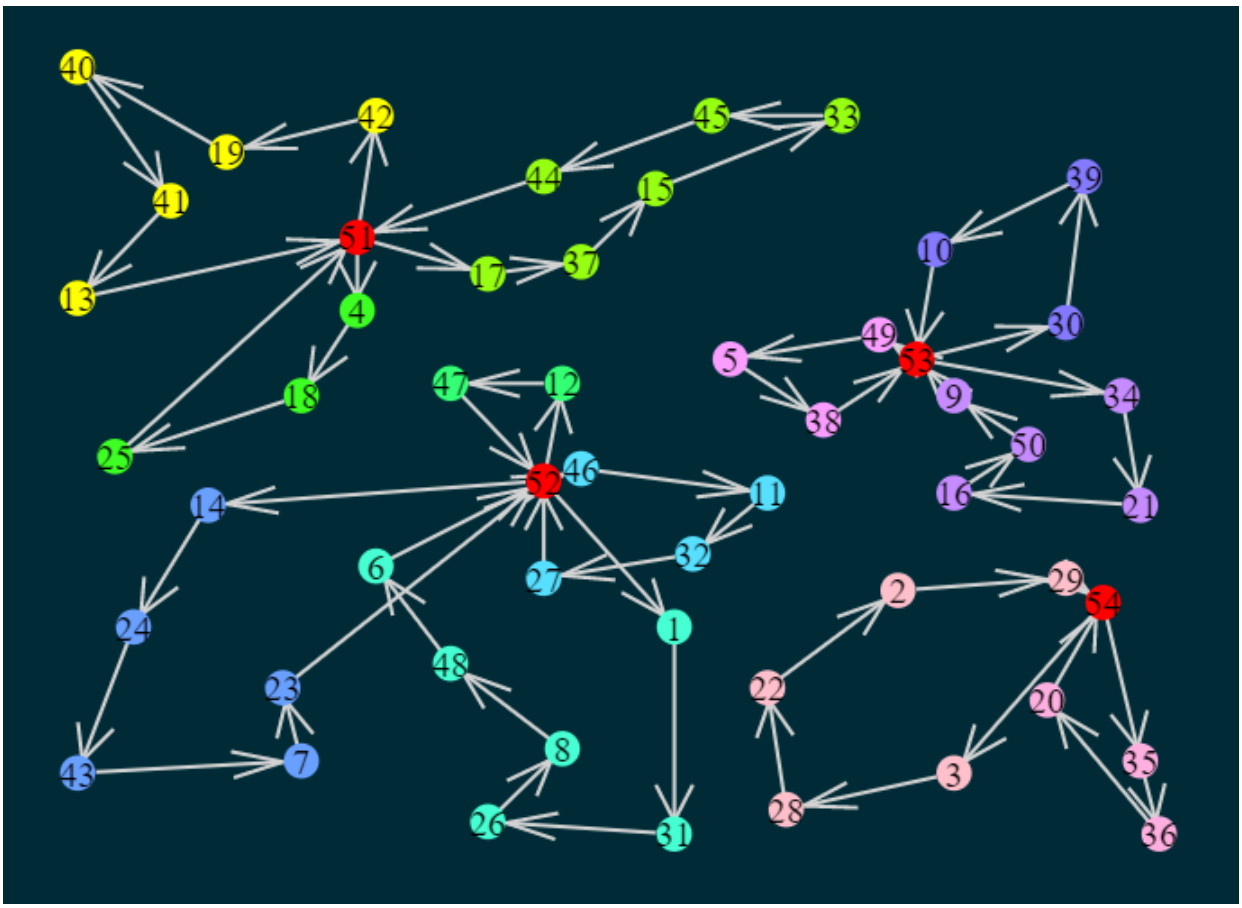


Figura 3.19: Resultado que se obtiene al aplicar la heurística de clusterización por **demandas, distancias y zonas** a la instancia p01 y las respectivas mejoras (20 en total). Se mejoró el valor objetivo desde desde el valor inicial **768.92**(2.43 seg) hasta el valor mejorado **600.08**(66.66 seg) utilizando 12 vehículos de un total de 16

Finalmente se muestra para la misma instancia, la ejecución del modelo exacto implementada en el lenguaje JULIA 1.0.5. La solución hallada igualmente es una mejor aproximación dado que el tiempo a esperar sería excesivamente alto.

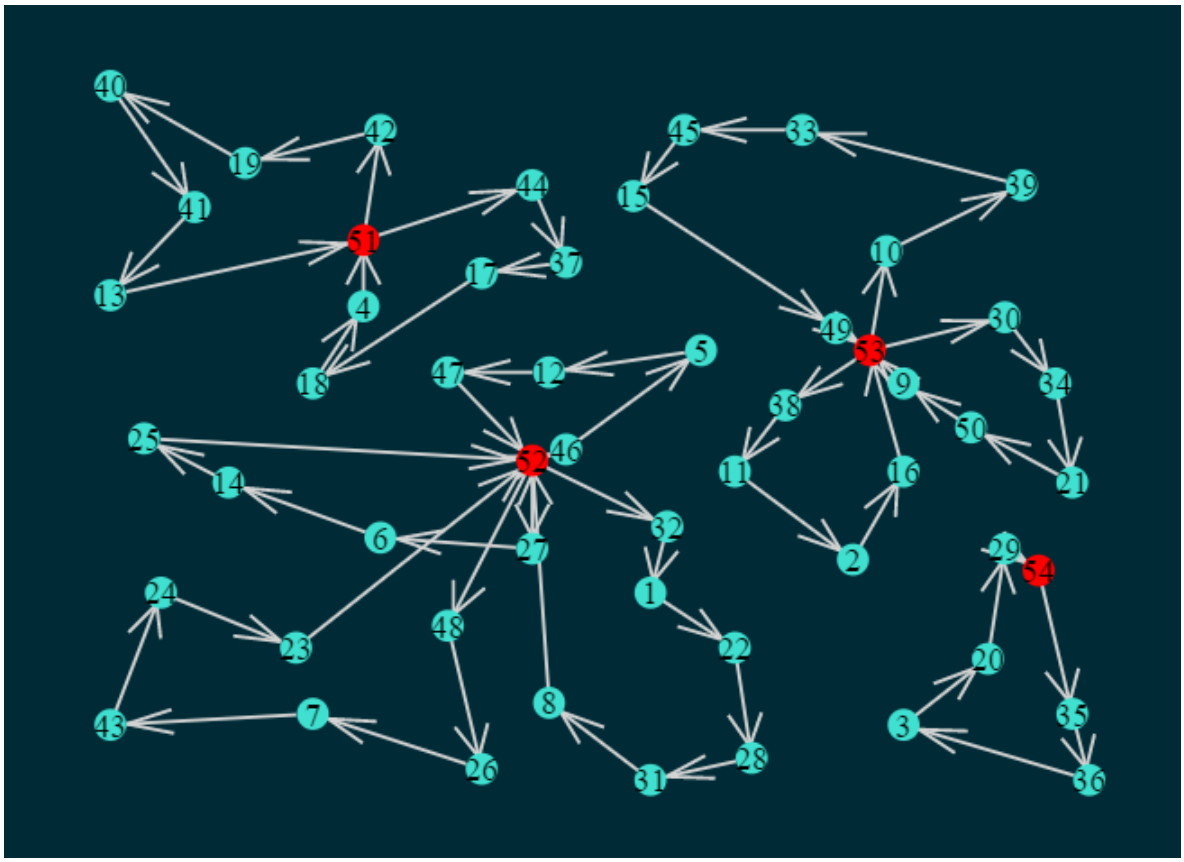


Figura 3.20: Solución exacta para la instancia p01 ejecutada hasta un tiempo = 17856 seg = 4.96 horas, Mejor Valor objetivo logrado = 584.43 (cortada al 27.2 % de gap), mejor solución conocida = 576.87, se usan 10 vehículos de un total de 16.

Para culminar el presente capítulo consolidamos los resultados obtenidos en una tabla comparativa solo para la instancia p01, preparándonos para en siguiente capítulo donde se consolidarán los resultados de ambas heurísticas presentadas para todas las 23 instancias. En la siguiente tabla mostramos las características de la instancia p01

cantidad de depósitos	Nro de clientes	BKS Mejor sol. conocida	Vehículos por depósito	Total de vehículos
4	50	576.87	4	16

Tabla 3.3: Características de la instancia p01

Luego de aplicar las dos heurísticas de clusterización por **demandas y distancias** y **demandas, distancias y zonas** a la misma instancia p01 se obtuvieron los resultados

que se muestran en la siguiente tabla

Heurística de clusterización	solución inicial		solución mejorada			
	dist.	t_{ini}	dist.	t_{acum}	NV	Mejoras
HDD	643.69	0.85	588.49	4.83	12	6
HDDZ	768.92	2.43	600.08	66.66	12	20

Tabla 3.4: Comparación de los resultados obtenidos por ambas heurísticas de clusterización aplicados a la instancia p01

En este caso la primera heurística tiene mejor desempeño que la segunda dado que inicia mas cerca del BKS y se aproxima mejor hacia dicho valor con 6 operciones de mejora y también en un mejor tiempo acumulado. En el caso del desempeño de la segunda heurística inicia lejos del BKS y logra aproximarse mediante 20 operaciones de mejora, ambos utilizan 12 vehículos de la flota de 16 vehículos.

Capítulo 4

RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados obtenidos como resultado de la implementación y ejecución en el lenguaje de programación JULIA 1.0.5 de las heurísticas de construcción y las sucesivas mejoras aplicadas a las soluciones iniciales, el reporte es mostrado en tablas completas donde se compara los resultados obtenidos por las heurísticas desarrolladas y la mejor solución conocida hasta el momento BKS, las corridas se realizaron para las 23 instancias existentes en la literatura desde p01 hasta p23. La discusión se genera en torno a la comparación de los resultados obtenidos y la satisfacción o cumplimiento de la hipótesis planteada.

En el presente capítulo se presentan los resultados obtenidos a partir de la implementación y ejecución en el lenguaje de programación JULIA 1.0.5 de las heurísticas de construcción y las sucesivas mejoras aplicadas a las soluciones iniciales, el reporte es mostrado en tablas completas donde se compara los resultados obtenidos por las heurísticas desarrolladas y la mejor solución conocida hasta el momento BKS, las corridas se realizaron para las 23 instancias existentes en la literatura desde p01 hasta p23.

4.1. Notación utilizada

En cada una de las tablas mostradas consignamos amplia información respecto a los resultados obtenidos a partir de la ejecución de la implementación en el lenguaje de programación JULIA 1.0.5

N: Cantidad de clientes.

M: Cantidad de depósitos.

K: Flota de vehículos en cada depósito, expresado en cantidad de vehículos en cada depósito.

T: Número total de vehículos (flota completa de vehículos).

Q: Capacidad máxima de cada vehículo.

BKS: Mejor solución conocida a la fecha dentro de la comunidad de investigación MDVRP, en este caso se toma como referencia lo publicado en (Shi et al., 2020, hi et al., 2020)

dist: Es el costo o distancia obtenida para la solución inicial obtenida por la heurística de clusterización, en el caso de la solución mejorada es la distancia o costo, la cual evidentemente es menor que el valor inicial.

t_{ini}**:** Es el tiempo en segundos que tarda la heurística para encontrar la solución inicial.

t_{acum}**:** Es el tiempo en segundos considerando el tiempo para hallar la solución inicial ademas del tiempo empleado en la mejora de la solución inicial.

NV/T: Es la cantidad de vehículos utilizados en la solución final mejorada, respecto a la flota total representado por T.

CM: Representa la cantidad de mejoras adicionales realizadas una vez obtenida la solución inicial.

HDD: Denotamos con estas siglas a la heurística de clusterización por distancias y demandas.

HDDZ: Estas siglas denotan a la heurística de clusterización por distancias, demandas y zonas.

HM: Representamos de esta manera a las heurísticas de mejora (las implementadas fueron el vecino mas cercano y splitt)

4.2. Resultados para la heurística HDD

En la siguiente tabla se muestra las características de cada una de las 23 instancias, como son cantidad de clientes(N), cantidad de depósitos(M), cantidad de vehículos en cada depósito(K) y la carga máxima de cada vehículo(Q). Asimismo para cada instancia se consigna la mejor solución conocida hasta el momento o BKS (Best Know Solution).

Para cada instancia y respecto a la solución inicial tenemos registrada la solución obtenida por aplicación de la heurística en cuanto a distancia total del ruteo y tiempo de cómputo. Respecto a la solución mejorada tenemos registrado la mejor distancia obtenida, el tiempo de cómputo acumulado que incluye al tiempo empleado en hallar la solución inicial asimismo la cantidad de vehículos que utiliza la solución hallada respecto al total de vehículos (NV/T) y finalmente la cantidad de operaciones de mejora realizados.

Características de las Instancias					MEJOR SOLUC.	Soluc. Inicial por **HDD**		Solución Mejorada **HDD+HM**			
Nro	N	M	K	Q	BKS	dist.	t_{ini}	dist.	t_{acum}	NV/T	CM
p01	50	4	4	80	576.87	643.69	0.85	588.49	10.54	12/16	6
p02	50	4	2	160	473.53	630.10	0.86	585.87	14.90	6/8	5
p03	75	5	3	140	641.19	720.27	0.93	689.38	10.74	11/15	5
p04	100	2	8	100	1001.59	1232.25	0.70	1145.16	45.16	14/16	5
p05	100	2	5	200	750.03	996.45	0.74	831.44	139.86	9/10	27
p06	100	3	6	100	876.50	1119.44	0.73	1017.71	27.29	17/18	12
p07	100	4	4	100	881.97	1137.23	0.71	997.12	54.32	14/16	10
p08	249	2	14	500	4372.78	5727.57	0.90	5181.44	564.37	27/28	45
p09	249	3	12	500	3858.66	4647.73	0.90	4346.48	391.33	28/36	30
p10	249	4	8	500	3631.11	4602.74	0.92	4202.72	241.26	27/32	29
p11	249	5	6	500	3546.06	4393.01	0.91	4154.11	790.99	27/30	27
p12	80	2	5	60	1318.95	2707.42	0.79	1770.32	131.23	8/10	34
p13	80	2	5	60	1318.95	2707.42	0.77	1770.32	78.14	8/10	34
p14	80	2	5	60	1360.12	2707.42	0.78	1770.32	92.28	8/10	34
p15	160	4	5	60	2505.42	5008.94	0.91	4430.36	108.80	15/20	12
p16	160	4	5	60	2572.23	5008.94	0.89	4430.36	89.24	15/20	12
p17	160	4	5	60	2709.09	5008.94	0.88	4430.36	44.93	15/20	12

Tabla 4.1: Solución factible inicial obtenida por la heurística de construcción **HDD** basada en clusterización y la aplicación sucesiva de las heurísticas de mejora **HM**. En la tabla se muestra la mejor solución conocida (BKS) hasta el momento, la distancia total obtenida por la solución, el tiempo que le toma a la heurística de construcción y el tiempo total que incluye el tiempo utilizado para hallar la solución inicial. También se muestra la cantidad de vehículos utilizados respecto a la flota total y la cantidad de operaciones de mejora.

Características de las Instancias					MEJOR SOLUC.	Soluc. Inicial por **HDD**		Solución Mejorada **HDD+HM**			
Nro	N	M	K	Q	BKS	dist.	t_{ini}	dist.	t_{acum}	NV/T	CM
p18	240	6	5	60	3702.85	8269.59	1.00	7226.22	141.81	23/30	29
p19	240	6	5	60	3827.06	8269.59	1.05	7226.22	147.63	23/30	29
p20	240	6	5	60	4058.07	8269.59	1.07	7226.22	152.04	23/30	29
p21	360	9	5	60	5474.84	12921.27	1.39	9387.03	892.71	34/45	89
p22	360	9	5	60	5702.16	12921.27	1.40	9387.03	875.81	34/45	89
p23	360	9	5	60	6078.75	12921.27	1.43	9387.03	871.09	34/45	89

Tabla 4.2: Continuación de la tabla anterior.

4.3. Resultados para la heurística HDDZ

En este caso se muestran los mismos parámetros que fueron mostrados en la tabla anterior para la heurística precedente.

Características de las Instancias					MEJOR SOLUC.	Soluc. Inicial por **HDDZ**		Solución Mejorada **HDDZ+HM**			
Nro	N	M	K	Q	BKS	dist.	t_{ini}	dist.	t_{acum}	NV/T	CM
p01	50	4	4	80	576.87	768.92	0.58	600.08	23.16	12/16	20
p02	50	4	2	160	473.53	539.53	0.72	499.20	37.22	6/8	4
p03	75	5	3	140	641.19	973.06	0.70	789.85	85.33	12/15	23
p04	100	2	8	100	1001.59	1380.91	0.70	1109.79	219.59	16/16	47
p05	100	2	5	200	750.03	1216.47	0.73	859.23	3693	9/10	76
p06	100	3	6	100	876.50	1420.52	0.73	952.09	254.88	17/18	54
p07	100	4	4	100	881.97	1248.00	0.62	949.98	130.26	17/16	43
p08	249	2	14	500	4372.78	7307.09	0.89	5342.53	16647	26/28	150
p09	249	3	12	500	3858.66	6672.99	1.86	4628.92	10964	27/36	150
p10	249	4	8	500	3631.11	5784.78	1.29	4365.26	11689	27/32	129
p11	249	5	6	500	3546.06	5018.83	0.96	4089.62	15833	27/30	100
p12	80	2	5	60	1318.95	1805.68	0.86	1622.90	502.16	8/10	12
p13	80	2	5	60	1318.95	1805.68	0.84	1622.90	487.31	8/10	12
p14	80	2	5	60	1360.12	1805.68	0.83	1622.90	484.41	8/10	12
p15	160	4	5	60	2505.42	3611.37	1.03	3173.32	2088.92	16/20	30
p16	160	4	5	60	2572.23	3611.37	1.50	3173.32	1994.39	16/20	30
p17	160	4	5	60	2709.09	3611.37	0.98	3173.32	2008.59	16/20	30

Tabla 4.3: Solución factible inicial obtenida por la heurística de construcción **HDDZ** basada en clusterización y la aplicación sucesiva de las heurísticas de mejora **HM**. En la tabla se muestra la mejor solución conocida (BKS) hasta el momento, la distancia total obtenida por la solución, el tiempo que le toma a la heurística de construcción y el tiempo total que incluye el tiempo utilizado para hallar la solución inicial. También se muestra la cantidad de vehículos utilizados respecto a la flota total y la cantidad de operaciones de mejora.

Características de las Instancias					MEJOR SOLUC.	Soluc. Inicial por **HDDZ**		Solución Mejorada **HDDZ+HM**			
Nro	N	M	K	Q	BKS	dist.	t_{ini}	dist.	t_{acum}	NV/T	CM
p18	240	6	5	60	3702.85	5417.05	1.26	4826.31	4104.53	24/30	38
p19	240	6	5	60	3827.06	5417.05	1.42	4826.31	3978.39	24/30	38
p20	240	6	5	60	4058.07	5417.05	1.23	4826.31	4007.41	24/30	38
p21	360	9	5	60	5474.84	8125.58	1.87	7311.57	10646	36/45	49
p22	360	9	5	60	5702.16	8125.58	12.98	7311.57	34417	36/45	49
p23	360	9	5	60	6078.75	8125.58	11.90	7311.57	34459	36/45	49

Tabla 4.4: continuación de la tabla anterior

A continuación mostramos la información consignadas en las tablas anteriores en cuanto al desempeño de las heurísticas desarrolladas en el capítulo anterior pero esta vez de manera gráfica. Se muestra la evolución a partir de la solución inicial (denotada por inicial) obtenida por las heurísticas HDD y HDDZ hasta el valor final (denotado por final) obtenida por susecivas aplicaciones de las heurísticas de mejoramiento, estos resultados son comparadas con las de la mejor solución que se conoce hasta el momento conocida como BKS.

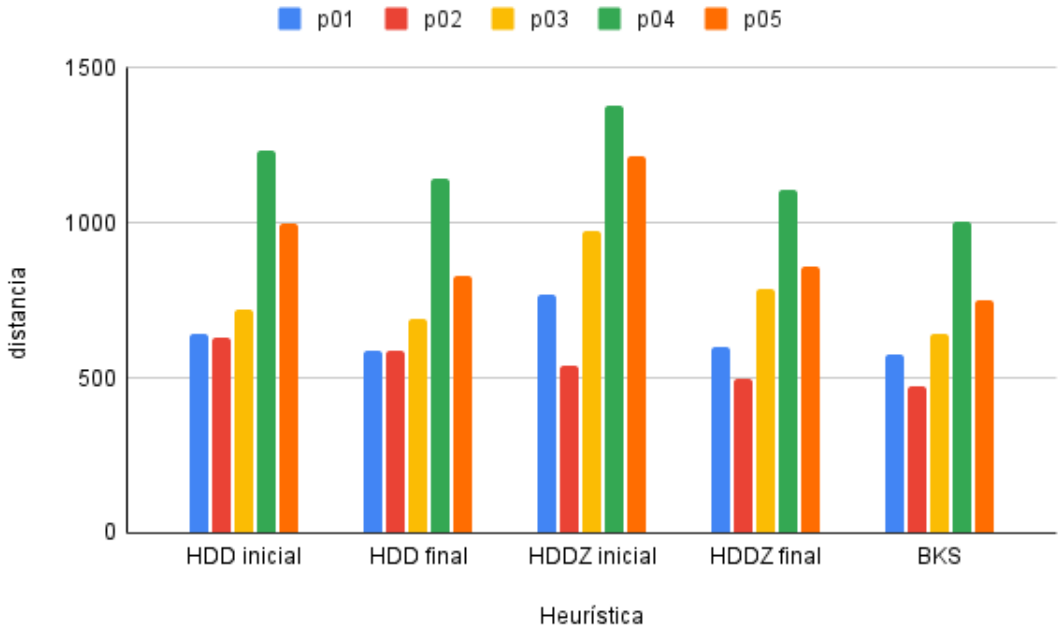


Figura 4.1: Comparación de los desempeños de las heurísticas HDD y HDDZ en su aplicación a las primeras 5 instancias p01 hasta p05

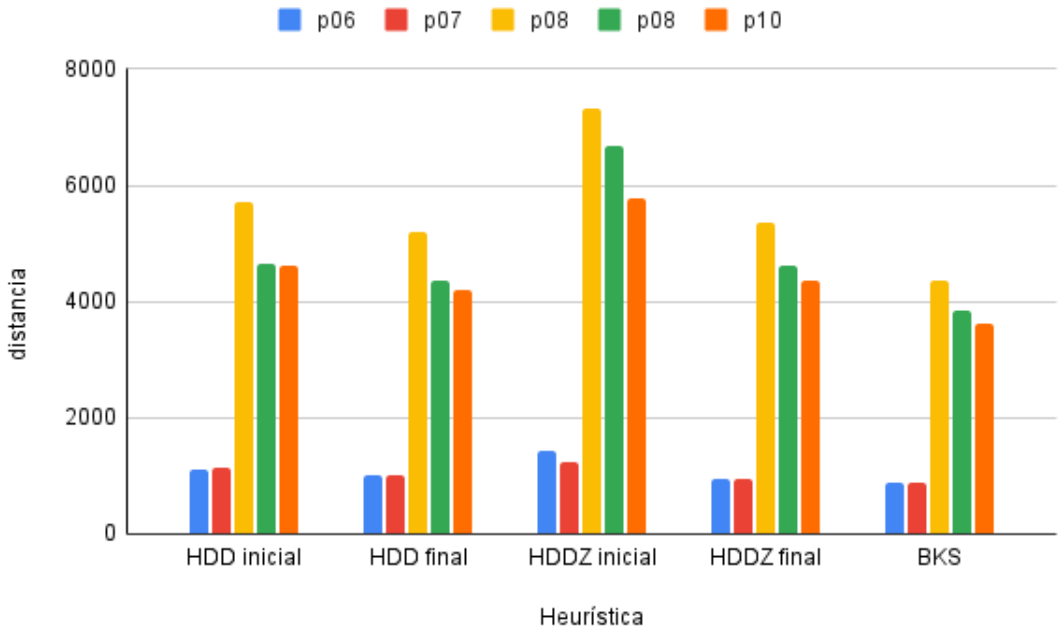


Figura 4.2: Comparación de los desempeños de las heurísticas HDD y HDDZ en su aplicación a las instancias desde p06 hasta p10

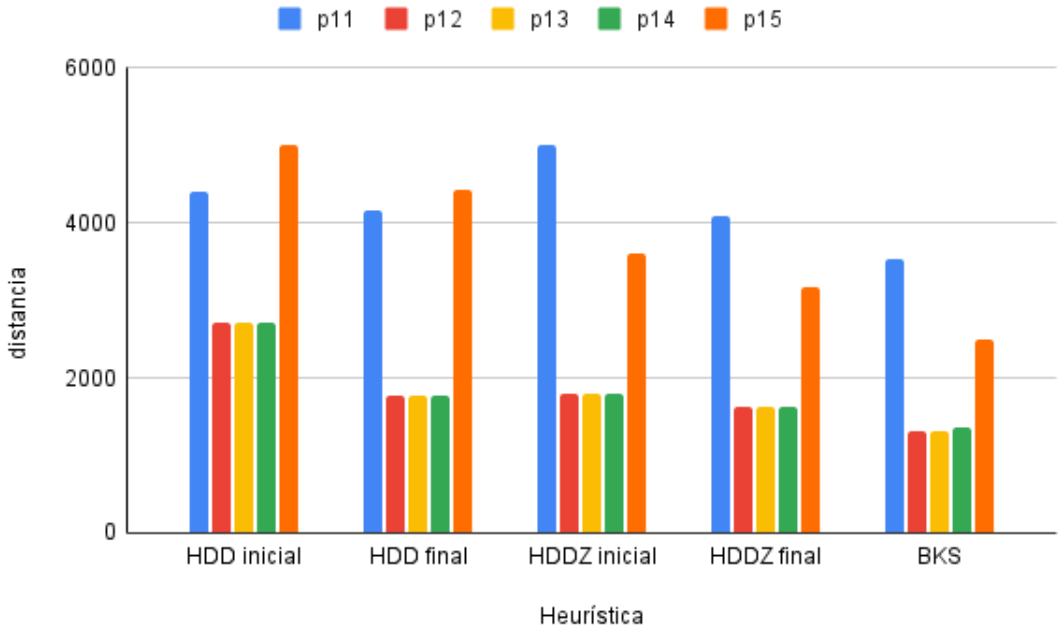


Figura 4.3: Comparación de los desempeños de las heurísticas HDD y HDDZ en su aplicación a las instancias desde p11 hasta p15

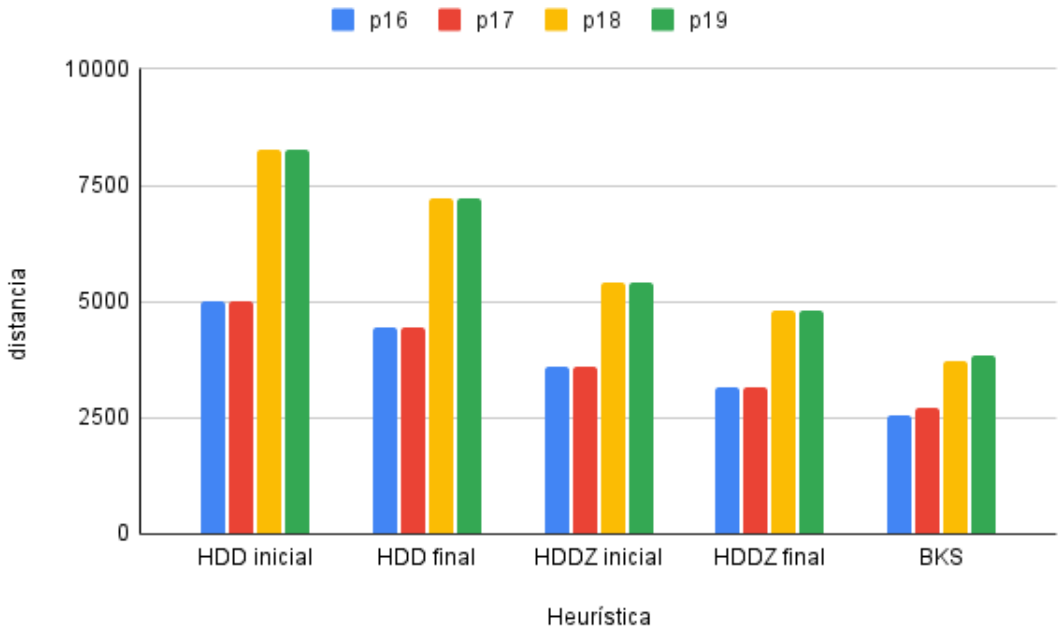


Figura 4.4: Comparación de los desempeños de las heurísticas HDD y HDDZ en su aplicación a las instancias desde p16 hasta p19.

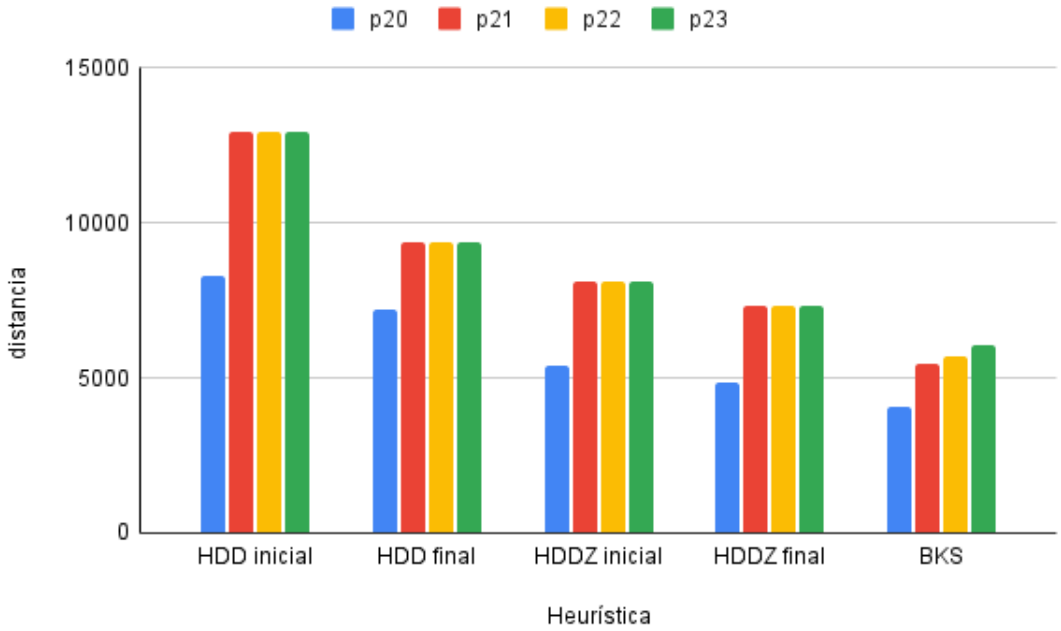


Figura 4.5: Comparación de los desempeños de las heurísticas HDD y HDDZ en su aplicación a las instancias desde p20 hasta p23.

El desempeño promedio se muestra en la siguiente gráfica donde se observa la evolución de la heurística desde la solución inicial hasta la mejora sucesiva.

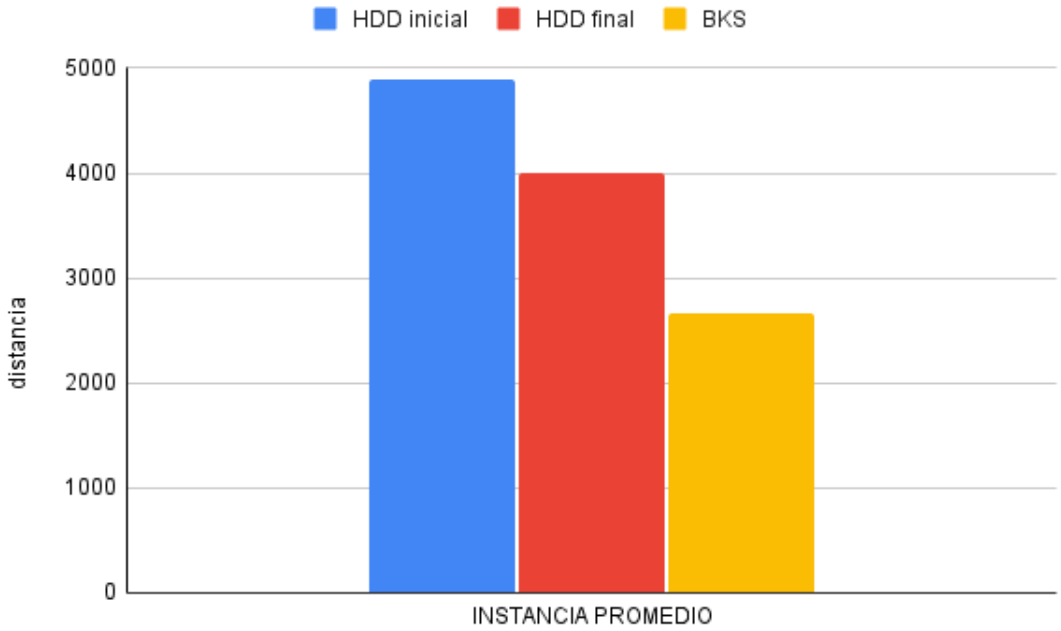


Figura 4.6: Desempeño promedio de la heurística de clusterización por distancias y demandas **HDD**

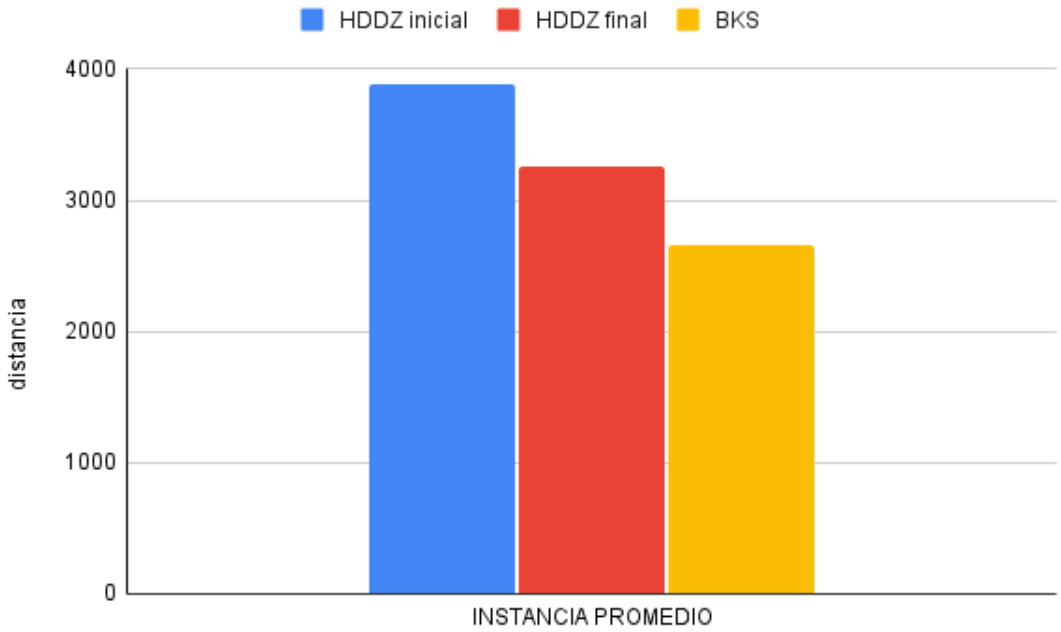


Figura 4.7: Desempeño promedio de la heurística de clusterización por distancias, demandas y zonas **HDDZ**

Finalmente mostramos la comparación del desempeño promedio entre ambas heurísticas.

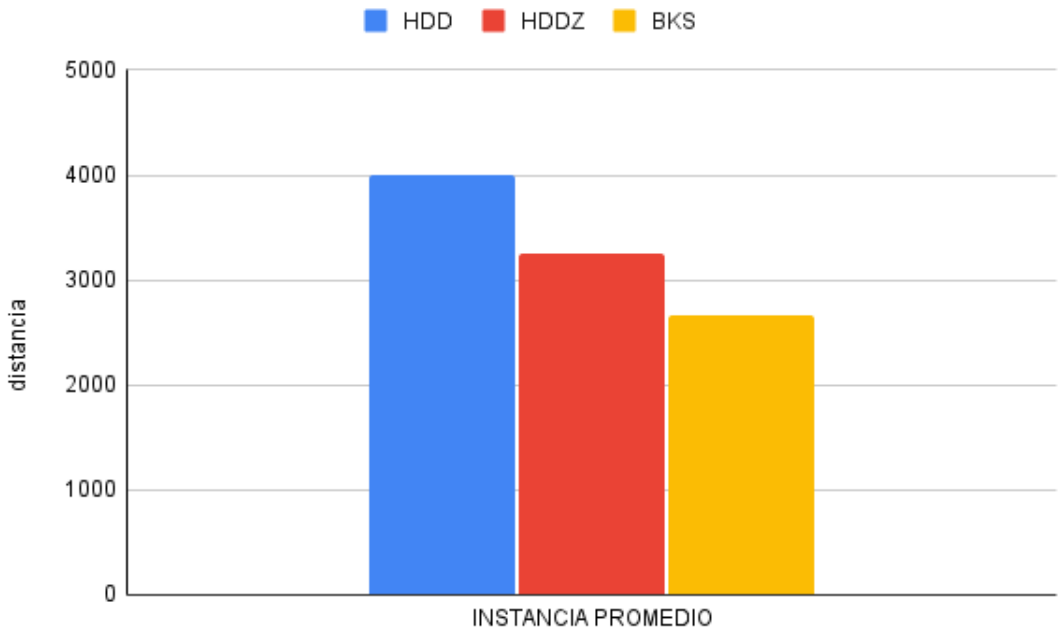


Figura 4.8: Desempeño promedio de las heurísticas de clusterización **HDD** y **HDDZ** respecto al valor **BKS**

Capítulo 5

CONCLUSIONES Y RECOMENDACIONES

En el presente capítulo se hace un listado de las conclusiones a las que se llega como resultado de la investigación asi como las recomendaciones para trabajos futuros basados en la presente investigación.

5.1. Conclusiones

Culminado el trabajo las principales conclusiones a las que hemos llegado son las siguientes: (para todos los casos HDD representa a la heurística por distancias y demandas mientras que HDDZ representa a la heurística por distancias, demandas y zonas)

1. Los resultados obtenidos se lograron a partir de la secuenciación de tres etapas que se tienen que desarrollar de manera consecutiva, estas son:

   CLUSTERIZACIÓN: En esta etapa los clientes son aglutinados en grupos denominados Clústeres, utilizando precisamente la heurística y su posterior implementación, al término de ésta etapa se tiene un grupo de clústeres y el grupo original de depósitos que no necesariamente son de la misma cardinalidad.

   MIP: Teniendo como insumo lo obtenido en la etapa anterior, en ésta etapa los clústeres obtenidos son asignados a determinados depósitos considerando

cercanías y las restricciones de flota de vehículos y capacidad del depósito, planteando y resolviendo para ello un problema de optimización matemática cuya solución óptima resuelve el problema de asignación.

TSP: Finalmente en cada uno de los clústeres obtenidos y asignados óptimamente en la etapa anterior se resuelve el problema del agente viajero TSP considerando el conjunto de clientes de cada clúster y el depósito asignado.

Es decir, podemos indicar que $HDD = CLUSTERIZACION + MIP + TSP$.

2. HDDZ proporciona mejores resultados que HDD.

3. HDD inicia lejos de BKS pero con las operaciones de mejora logra acercarse un tramo largo.

4. Por su lado HDDZ inicia cerca de BKS pero no mejora mucho.

5. Ambas heurísticas son eficientes (en términos de tiempo de ejecución) para hallar una solución inicial. A HDDZ le toma mas tiempo en mejorar la solución inicial respecto a HDD.

6. El lenguaje de programación JULIA y los solvers GUROBI y CPLEX son herramientas básicas imprescindibles para obtener resultados numéricos y asi medir la bondad de las heurísticas.

5.2. Recomendaciones

Considerando que el trabajo realizado aún no ha explorado todas las posibilidades dentro de las múltiples existentes y teniendo en cuenta que una heurística es algo asi como una intuición, nos permitimos dar algunas recomendaciones para futuros trabajos de investigación que podrian tener como base a la presente

1. Diseñar otras estrategias de clusterización que permitan encontrar mejores soluciones iniciales.

2. Aplicar otras estrategias de mejora que permitan escapar de un mínimo local a la solución inicial obtenida.

3. Buscar nuevas instancias de mayor tamaño para medir la bondad de la heurística

4. Aplicar los resultados a instancias reales como por ejemplo entrega por delivery y otros.

5. Incorporar estrategias de programación matemática (relajación Lagrangeana, fix and relax, optimización subgradiente) para analizar el modelo exacto y asi obtener cotas inferiores.

Referencias

Programación matemática: estructuras y algoritmos. *Journal of the Operational Research Society*.

Cabezas, X. A. F. (2014). Problemas del milenio: P vs np.

Cook, S. (2006). The p versus np problem. *The millennium prize problems*, pages 87–104.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Desrochers, M. and Laporte, G. (1991). Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36.

Held, Michael y Wolfe, P. y. C. H. P. Validación de optimización de subgrado. *Programación matemática*.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Lenstra, J. and Kan, A. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Llanes-Font, M., Isaac-Godínez, C. L., Moreno-Pino, M., and García-Vidal, G. (2014). De la gestión por procesos a la gestión integrada por procesos. *Ingeniería Industrial*, 35(3):255–264.

Lüer, A., Benavente, M., Bustos, J., and Venegas, B. (2009). El problema de rutas de vehículos: Extensiones y métodos de resolución, estado del arte. In *EIG*.

Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies*, 70:100–112.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Padberg, M. and Sung, T.-Y. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1):315–357.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Pichpibul, T. and Kawtummachai, R. (2012). An improved clarke and wright savings algorithm for the capacitated vehicle routing problem. *ScienceAsia*, 38(3):307–318.

Salhi, S., Imran, A., and Wassan, N. A. (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research*, 52:315–325.

Salhi, S. and Sari, M. (1997). A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. *European Journal of Operational Research*, 103(1):95–112.

Shi, Y., Lv, L., Hu, F., and Han, Q. (2020). A heuristic solution method for multi-depot vehicle routing-based waste collection problems. *Applied Sciences*, 10(7):2403.

Stodola, P. (2018). Using metaheuristics on the multi-depot vehicle routing problem with modified optimization criterion. *Algorithms*, 11(5):74.

Surekha, P. and Sumathi, S. (2011). Solution to multi-depot vehicle routing problem using genetic algorithms. *World Applied Programming*, 1(3):118–131.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Urruty, J.-B. H. and Lemaréchal, C. (1996). *Convex analysis and minimization algorithms*. Springer-Verlag.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Padberg, M. and Sung, T.-Y. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1):315–357.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Pichpibul, T. and Kawtummachai, R. (2012). An improved clarke and wright savings algorithm for the capacitated vehicle routing problem. *ScienceAsia*, 38(3):307–318.

Salhi, S., Imran, A., and Wassan, N. A. (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research*, 52:315–325.

Salhi, S. and Sari, M. (1997). A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. *European Journal of Operational Research*, 103(1):95–112.

Shi, Y., Lv, L., Hu, F., and Han, Q. (2020). A heuristic solution method for multi-depot vehicle routing-based waste collection problems. *Applied Sciences*, 10(7):2403.

Stodola, P. (2018). Using metaheuristics on the multi-depot vehicle routing problem with modified optimization criterion. *Algorithms*, 11(5):74.

Surekha, P. and Sumathi, S. (2011). Solution to multi-depot vehicle routing problem using genetic algorithms. *World Applied Programming*, 1(3):118–131.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Urruty, J.-B. H. and Lemaréchal, C. (1996). *Convex analysis and minimization algorithms*. Springer-Verlag.

Tesis doctoral

por Rosulo Pérez

Fecha de entrega: 14-abr-2022 08:35p.m. (UTC-0500)
Identificador de la entrega: 1811069886
Nombre del archivo: Tesis_de_Doctorado_Rosulo__2.pdf (1.99M)
Total de palabras: 27873
Total de caracteres: 135478

Tesis doctoral

INFORME DE ORIGINALIDAD

24%	21%	1%	7%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	studylib.es Fuente de Internet	10%
2	Submitted to Universidad Nacional de Trujillo Trabajo del estudiante	3%
3	docplayer.es Fuente de Internet	2%
4	sisbib.unmsm.edu.pe Fuente de Internet	1%
5	www.colibri.udelar.edu.uy Fuente de Internet	1%
6	repositorio.uns.edu.pe Fuente de Internet	1%
7	hdl.handle.net Fuente de Internet	1%
8	Submitted to Universidad Nacional del Santa Trabajo del estudiante	<1%
9	eprints.uanl.mx Fuente de Internet	<1%

10	www.investigacionyposgrado.uadec.mx Fuente de Internet	<1 %
11	bibliotecadigital.univalle.edu.co Fuente de Internet	<1 %
12	repositorio.unal.edu.co Fuente de Internet	<1 %
13	Submitted to Pontificia Universidad Catolica del Peru Trabajo del estudiante	<1 %
14	idoc.pub Fuente de Internet	<1 %
15	editorial.umariana.edu.co Fuente de Internet	<1 %
16	repositorio.uniagustiniana.edu.co Fuente de Internet	<1 %
17	ri.uaemex.mx Fuente de Internet	<1 %
18	Submitted to Associatie K.U.Leuven Trabajo del estudiante	<1 %
19	upcommons.upc.edu Fuente de Internet	<1 %
20	www.ptolomeo.unam.mx:8080 Fuente de Internet	<1 %
21	Submitted to EP NBS S.A.C.	

	Trabajo del estudiante	<1 %
22	qdoc.tips Fuente de Internet	<1 %
23	Submitted to University of Auckland Trabajo del estudiante	<1 %
24	futur.upc.edu Fuente de Internet	<1 %
25	Submitted to De Montfort University Trabajo del estudiante	<1 %
26	es.wikipedia.org Fuente de Internet	<1 %
27	repository.udistrital.edu.co Fuente de Internet	<1 %
28	1library.co Fuente de Internet	<1 %
29	pt.scribd.com Fuente de Internet	<1 %
30	repositorio.unasam.edu.pe Fuente de Internet	<1 %
31	Submitted to tec Trabajo del estudiante	<1 %
32	worldwidescience.org Fuente de Internet	<1 %

33	pure.rug.nl Fuente de Internet	<1 %
34	repositorioacademico.upc.edu.pe Fuente de Internet	<1 %
35	tangara.uis.edu.co Fuente de Internet	<1 %
36	www.cie.uva.es Fuente de Internet	<1 %
37	www.coursehero.com Fuente de Internet	<1 %
38	upc.aws.openrepository.com Fuente de Internet	<1 %
39	bibing.us.es Fuente de Internet	<1 %
40	www.dspace.uce.edu.ec Fuente de Internet	<1 %
41	es.unionpedia.org Fuente de Internet	<1 %
42	Submitted to Universidad Autónoma de Nuevo León Trabajo del estudiante	<1 %
43	es.scribd.com Fuente de Internet	<1 %
44	repositorio.pucp.edu.pe	

	Fuente de Internet	<1 %
45	Submitted to 95480 Trabajo del estudiante	<1 %
46	Submitted to Institute of Graduate Studies, UiTM Trabajo del estudiante	<1 %
47	personales.ac.upc.edu Fuente de Internet	<1 %
48	Román Anselmo Mora Gutiérrez. "Diseño y desarrollo de un método heurístico basado en un sistema socio-cultural de creatividad para la resolución de problemas de optimización no lineales y diseño de zonas electorales", Universidad Nacional Autonoma de Mexico, 2013 Publicación	<1 %
49	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	<1 %
50	Submitted to Universidad Sergio Arboleda Trabajo del estudiante	<1 %
51	revistas.unipamplona.edu.co Fuente de Internet	<1 %
52	www.nureinvestigacion.es Fuente de Internet	<1 %

53	www.scribd.com Fuente de Internet	<1 %
54	docplayer.biz.tr Fuente de Internet	<1 %
55	moam.info Fuente de Internet	<1 %
56	revistacompudata.com Fuente de Internet	<1 %
57	www.labc.usb.ve Fuente de Internet	<1 %
58	www.mobile.amena.com Fuente de Internet	<1 %
59	zaguan.unizar.es Fuente de Internet	<1 %
60	deepai.org Fuente de Internet	<1 %
61	nanopdf.com Fuente de Internet	<1 %
62	tesis.ipn.mx Fuente de Internet	<1 %
63	www.pld.org.do Fuente de Internet	<1 %
64	www.slideserve.com Fuente de Internet	<1 %

65	Submitted to Unviersidad de Granada Trabajo del estudiante	<1 %
66	Víctor Yepes Piqueras. "Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW.", 'Universitat Politecnica de Valencia', 2015 Fuente de Internet	<1 %
67	dspace.ucuenca.edu.ec Fuente de Internet	<1 %
68	dx.doi.org Fuente de Internet	<1 %
69	es.slideshare.net Fuente de Internet	<1 %
70	repositorio.puce.edu.ec Fuente de Internet	<1 %
71	repositorio.uniandes.edu.co Fuente de Internet	<1 %
72	uvadoc.uva.es Fuente de Internet	<1 %
73	www.atc.uniovi.es Fuente de Internet	<1 %
74	www.slideshare.net Fuente de Internet	<1 %

75	Varimna Singh, L. Ganapathy, Ashok K. Pundir. "chapter 20 An Improved Genetic Algorithm for Solving Multi Depot Vehicle Routing Problems", IGI Global, 2021 Publicación	<1 %
76	bdigital.unal.edu.co Fuente de Internet	<1 %
77	doku.pub Fuente de Internet	<1 %
78	issuu.com Fuente de Internet	<1 %
79	repositorio.unab.cl Fuente de Internet	<1 %
80	repositorio.unc.edu.pe Fuente de Internet	<1 %
81	www.imm.unavarra.es Fuente de Internet	<1 %
82	www.dspace.espol.edu.ec Fuente de Internet	<1 %
83	Mazzeo, S.. "An Ant Colony Algorithm for the Capacitated Vehicle Routing", Electronic Notes in Discrete Mathematics, 20041201 Publicación	<1 %