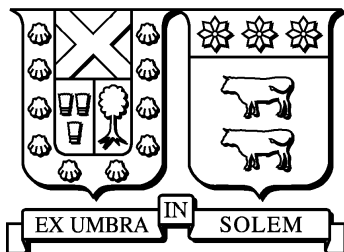


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



“ACERCAMIENTOS BASADOS EN BÚSQUEDA
LOCAL PARA EL PROBLEMA DE
RECOLECCIÓN DE LECHE CON MEZCLA”

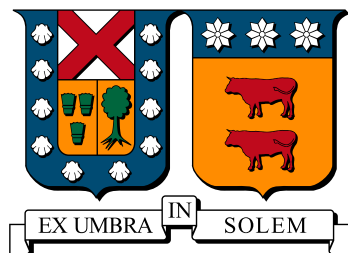
JORGE ANDRÉS VILLAGRÁN MUÑOZ

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ELIZABETH MONTERO U.

OCTUBRE 2019

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



**“ACERCAMIENTOS BASADOS EN BÚSQUEDA
LOCAL PARA EL PROBLEMA DE
RECOLECCIÓN DE LECHE CON MEZCLA”**

JORGE ANDRÉS VILLAGRÁN MUÑOZ

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO**

PROFESOR GUÍA: ELIZABETH MONTERO U.
PROFESOR CORREFERENTE: JOSÉ LUIS MARTÍ L.

OCTUBRE 2019

MATERIAL DE REFERENCIA, SU USO NO INVOLUCRA RESPONSABILIDAD DEL AUTOR O DE LA INSTITUCIÓN

Agradecimientos

Quiero agradecer enormemente a mis padres Jorge Villagrán Arancibia y Sandra Muñoz Gomez que me brindaron la oportunidad para poder estudiar en la Universidad, con el apoyo económico y emocional que me permitió concentrarme solo en mis estudios. Fueron 9 largos años en los que recibí todas las facilidades y oportunidades para poder crecer personal y profesionalmente.

Durante mi estancia en la universidad, conocí a gente increíble, muchos de ellos ahora los considero mis amigos (integrantes de tarao). Con ellos pase grandes momentos durante esas largas jornadas de estudio, clases tediosas y carretes varios.

El desarrollo de este trabajo de memoria no se hubiera concretado sin el apoyo de mi profesora guía: Elizabeth Montero, que me orientó con la mejor disposición y energía en lo que es el último hito que me separa de ser un estudiante a un profesional.

Resumen

Esta memoria aborda el problema del enrutamiento de vehículos para la recolección de leche considerando la posibilidad de mezcla tanto en ruta como en planta. Esta es una variante del problema de enrutamiento de vehículos (VRP por sus siglas en inglés) que modela las rutas que debe seguir una flota de camiones recolectores de leche, desde un conjunto de granjas productoras hacia una planta de procesamiento, teniendo en cuenta que en cada granja se produce leche de distinta calidad y precio. La mezcla de leche durante el transporte permite generar rutas con un menor costo, pero con una disminución del beneficio por parte de la planta. Por su parte, el beneficio aumenta en base a la calidad de cada tipo de leche. El objetivo de este problema es encontrar un conjunto de rutas óptimas, una para cada camión, que permita la recolección de leche de todos los productores a un costo mínimo, satisfaciendo las cuotas por cada tipo de leche que impone la planta, sin sobrepasar la capacidad de cada camión. Para resolver este problema, se plantea un modelo de programación entera mixta y dos técnicas de búsqueda local. Un acercamiento basado en *hill-climbing* y uno basado en *iterated local search*. Los acercamientos son evaluados utilizando cuatro conjuntos de instancias. El último basado en la distribución de 500 granjas lecheras en el sur de Chile.

Abstract

This work addresses the problem of routing vehicles for milk collection considering the possibility of mixing both on the road and in the plant. This is a variant of the vehicle routing problem (VRP) that models the routes that a fleet of milk collection trucks must follow, from a set of producing farms to a processing plant, taking into account that in Each farm produces milk of different quality and price. The mixing of milk during transport allows routes to be generated with a lower cost, but with a decrease in the benefit of the plant. For its part, the benefit increases based on the quality of each type of milk. The objective of this problem is to find a set of optimal routes, one for each truck, which allows the collection of milk from all producers at a minimum cost, satisfying the quotas for each type of milk imposed by the plant, without exceeding the capacity of each truck. To solve this problem, a mixed integer programming model and two local search techniques are proposed. An approach based on *hill-climbing* and one based on *iterated local search*. The approaches are evaluated using four sets of instances. The latter based on the distribution of 500 dairy farms in southern Chile.

Índice de Contenidos

Agradecimientos	III
Resumen	IV
Abstract	V
Índice de Contenidos	VI
Lista de Tablas	IX
Lista de Figuras	XI
Introducción	1
1. Definición del Problema	3
1.1. Formulación Matemática	5
1.2. Parámetros	6
1.3. Variables	6
1.4. Restricciones	7
1.4.1. Función Objetivo	10
1.5. Resumen	10

2. Estado del Arte	12
2.1. Ruteo de Vehículos en General	13
2.2. Ruteo de Vehículos con Múltiples Productos	17
2.3. Ruteo de Vehículos con Múltiples Productos y Mezclas	19
2.4. Recolección de Leche	20
2.5. Resumen	22
3. Propuesta	25
3.1. Representación	26
3.2. Función de Evaluación	27
3.3. Generación de Solución Inicial	28
3.4. Hill Climbing	30
3.5. Iterated Local Search	37
3.5.1. Perturbación	39
3.6. Resumen	40
4. Implementación	42
4.1. Objetivos	42
4.2. Instancias de Prueba	43
4.2.1. Instancias Pequeñas	43
4.2.2. Instancias reales	44
4.2.3. Entorno de Experimentación	49
4.3. Resultados	49
4.3.1. Hill Climbing	50
4.3.2. Iterated Local Search	57
4.3.3. Análisis de Convergencia	64

4.3.4. Instancias Reales	72
4.4. Conclusiones	75
5. Conclusiones	78
Conclusiones	78
Bibliografía	81

Índice de tablas

4.1. Instancias de Augerat [2]	45
4.2. Instancias de Taillard [50] y Fisher [19]	46
4.3. Instancias de Reinelt [44]	46
4.4. Instancia de Paredes-Belmar. [36] con 100 camiones, más variantes que consideran 88, 77 y 72 camiones.	48
4.5. Evaluación de parámetros de penalización con <i>hill-climbing</i>	52
4.6. Evaluación de parámetros de penalización con <i>hill-climbing</i>	53
4.7. Evaluación de parámetros de penalización con <i>hill-climbing</i>	53
4.8. Resumen con un compilado de los resultados de las pruebas realizadas con <i>hill-climbing</i> para determinar el mejor valor de los parámetros de penalización.	53
4.9. Evaluación de naturaleza de los procesos de búsqueda con <i>hill-climbing</i>	55
4.10. Evaluación de naturaleza de los procesos de búsqueda con <i>hill-climbing</i>	56
4.11. Evaluación de naturaleza de los procesos de búsqueda con <i>hill-climbing</i>	56
4.12. Resumen con un compilado de los resultados de las pruebas realizadas con <i>hill-climbing</i> para determinar la naturaleza de los procesos de búsqueda que otorga los mejores resultados.	57
4.13. Evaluación del valor del parámetro de perturbación con <i>iterated local search</i>	59

4.14. Evaluación del valor del parámetro de perturbación con <i>iterated local search</i> .	60
4.15. Evaluación del valor del parámetro de perturbación con <i>iterated local search</i> .	60
4.16. Resumen con un compilado de los resultados de las pruebas realizadas con <i>iterated local search</i> para la determinar valor óptimo del parámetro de perturbación.	60
4.17. Evaluación de enfoques de perturbación con <i>iterated local search</i>	62
4.18. Evaluación de enfoques de perturbación con <i>iterated local search</i>	63
4.19. Evaluación de enfoques de perturbación con <i>iterated local search</i>	63
4.20. Resumen con un compilado de los resultados de las pruebas realizadas con <i>iterated local search</i> para la determinar la solución a la cual aplicarle la perturbación.	63

Índice de figuras

1.1. Esquema de ejemplo de una instancia con 13 productores, una planta y 3 camiones.	5
3.1. Esquema representativo de una solución para el MCwB, con n camiones. . .	26
4.1. Región del sur de Chile con la distribución de todos los productores de la instancia real. Los círculos, triángulos y cuadrados corresponden a los productores diferenciados por tipo de leche, mientras que el diamante corresponde a la planta.	47
4.2. Histograma de producción de leche de los productores en litros.	48
4.3. Convergencia para la instancia <i>a60</i>	65
4.4. Convergencia para la instancia <i>c50</i>	66
4.5. Convergencia para la instancia <i>c75</i>	67
4.6. Boxplot con los resultados de la instancia <i>c75</i>	68
4.7. Convergencia para la instancia <i>tai75A</i>	69
4.8. Boxplot con los resultados de la instancia <i>tai75A</i>	70
4.9. Gráfico de convergencia para la instancia <i>tai75D</i>	71
4.10. Boxplot con los resultados de la instancia <i>tai75D</i>	71

4.11. Convergencia de la instancia <i>real500-100</i> utilizando <i>hill-climbing</i> e <i>iterated local search</i>	72
4.12. Convergencia de las instancias <i>reales</i> utilizando <i>iterated local search</i>	74
4.13. Convergencia de las instancias <i>reales</i> utilizando <i>iterated local search</i>	75
4.14. Convergencia de la instancias <i>real500-72</i> y <i>real500-100</i> utilizando <i>iterated local search</i> por un tiempo de 26.500 segundos.	76

Introducción

El traslado de leche desde los productores hasta las plantas de tratamiento conlleva un costo considerable a las empresas que utilizan la leche como materia prima. La leche es un producto perecedero, es decir, necesita estar refrigerado a una cierta temperatura para detener o retardar el crecimiento de bacterias, por lo que se requieren camiones con almacenamiento refrigerado que son mucho más costosos que un camión tradicional. Los productores de leche muchas veces se encuentran muy lejos unos de otros, separados por áreas rurales bastante extensas que hace que el transporte de leche sea costoso. Muchas veces estos productores se agrupan en pequeñas cooperativas las cuales se tratan como una entidad consolidada con un solo comprador, haciendo, por lo tanto, que se tenga que recoger la leche de todos los miembros de la cooperativa, incluso cuando algunos estén más alejados de la plantas de tratamiento. Junto con esto, es importante saber que cada granja produce leche de distinta calidad o grado, que debe ser almacenada en compartimentos distintos o ser recogida por distintos camiones. Cada tipo de leche se utiliza para producir un determinado producto en la planta de tratamiento, por ende, tiene un beneficio asociado.

La recolección de leche de distintas calidades desde diversos productores conlleva una tarea difícil de planificar, debido a que además de necesitar camiones con distintos compartimentos hay que procurar ocupar la mayor cantidad de espacio posible en cada uno para una determinada ruta. Una alternativa a los compartimentos es utilizar distintos camiones, donde cada uno se encargue de recoger un tipo de leche determinado. Esto puede aumentar los costos, debido a que se necesitan más camiones, algunas veces más de un camión para un mismo tipo de leche y la elección de rutas ineficientes. Esto debido a las grandes distancias

que deban recorrer los camiones para ir a buscar la leche de una cierta calidad. Una alternativa a esto es permitir la mezcla de leche de distintas calidades. Esto reduce la ganancia, ya que la mezcla de leche de distintas calidades produce como resultado leche con calidad igual a la menor de las mezcladas, pero disminuye los costos de transporte. Lo interesante es que, en muchos casos, la ganancia obtenida en los costos del transporte es mayor que la pérdida ocasionada por la mezcla de leche de diferentes calidades. Dado esto, es posible generar mejores rutas donde se tenga en cuenta la posibilidad de que un camión recoja leche de calidades distintas durante un recorrido, permitiendo obtener mejores soluciones que las de un problema que no considere la posibilidad de mezcla de leche de distintas calidades.

El problema de recolección de leche puede ser considerado como un caso especial del VRP que considera el servicio de recogida en los productores y entrega en la planta. Este tipo de problemas se ha tratado en varias industrias, especialmente fábricas de productos avícolas y lácteos. La variante que se trata en este documento es la del problema de recolección de leche con mezcla (MCwB por sus siglas en inglés) que considera la posibilidad de mezclar leche de distintas calidades, minimizando los costos de transporte y así maximizar la ganancia total.

Para comenzar, se realizará una definición formal del problema, detallando sus principales características y exponiendo el modelo matemático asociado. Esto se presenta en el capítulo 1. En el capítulo 2 se presenta una revisión del estado del arte relacionado del problema estudiado, comenzando por el ruteo de vehículos clásico, pasando por la recolección multi-producto para finalmente estudiar los acercamientos de ruteo multi-producto con mezcla. En el capítulo 3 se presentan las propuestas de resolución heurística basadas en búsqueda local desarrolladas en este trabajo de memoria. Un acercamiento basado en *hill-climbing* enfocado a resolver 3 conjuntos de instancias clásicas del estado del arte y uno basado en *iterated local search* enfocado en resolver un conjunto que fue construido a partir de la información de distribución de 500 granjas lecheras ubicadas en la región de Los Lagos del sur de Chile. Estos resultados se presentan en el capítulo 4. Por último, en el capítulo 5 se presentan las conclusiones que se obtuvieron de la realización de este trabajo de memoria.

Capítulo 1

Definición del Problema

El problema de enrutamiento de vehículos para la recolección de leche considerando mezcla busca encontrar rutas para un conjunto de camiones que, partiendo desde una planta, recogen leche de una cierta calidad desde un conjunto de productores y la llevan de regreso a la planta para su posterior procesamiento. El beneficio obtenido por la entrega de un tipo de leche en la planta aumenta con la calidad de la leche recolectada. El objetivo del problema es encontrar un conjunto de rutas óptimas que permitan satisfacer las cuotas de leche de la planta de procesamiento. Cada camión toma como punto de partida la planta, luego visita a un conjunto de productores donde a cada uno se le recoge el total de su producción de leche y finalmente se regresa a la planta procesadora desde donde comenzó su itinerario. En cada tramo de la ruta se incurre en un gasto y por cada granja visitada ingresa leche al depósito del camión. El beneficio por cada litro de leche que satisfaga las cuotas de la granja contribuye a la ganancia. El gasto incurrido en el transporte de la leche por parte de cada camión contribuye al costo. El beneficio total menos el costo total contribuye a la ganancia, que es lo que se intenta maximizar en este problema, mediante la búsqueda de rutas óptimas y mezcla de la leche tanto en los camiones como en la misma planta. La mezcla de leche de distintas calidades da como resultado leche con una calidad igual a la menor de las mezcladas. Si se realiza mezcla en los camiones, esto puede provocar una disminución de los beneficios e incluso provocar que no se cumplan las cuotas de leche de la planta si no se hace correctamente. Se realiza de todas maneras, pues, puede disminuir los costos considerablemente

debido al hallazgo de rutas más óptimas.

Para la planificación de las rutas, se cuenta en primera instancia, con una cierta flota de camiones fija, cada uno con un solo compartimento y capacidad fija. Es por esto, que además de la restricción de las cuotas de leche de la granja, hay una adicional respecto a la capacidad de los camiones durante la recolección de leche en una ruta. Se cuenta con una matriz de distancias entre todos los nodos, la que incluye la planta y todos los productores. Para cada productor se conoce la cantidad y tipo de leche que produce. Además se conoce los requerimientos de leche de cada tipo de la planta. Durante la búsqueda de rutas óptimas, el algoritmo será el encargado de determinar cuantos camiones se usarán, por lo que no necesariamente se usarán todos los disponibles. En la ruta, cada productor es visitado una vez y durante su visita el camión encargado debe recoger toda su leche. Es en este proceso donde se realiza la mezcla en los camiones. Esto ocurre cuando la leche del productor actual es distinta a la acumulada en el compartimento del camión.

En la figura 1.1 se muestra un ejemplo de solución al MCwB. Se cuenta con un conjunto de 13 productores, que producen 3 tipos de leche distintas. Cada productor es presentado por un círculo y su color de relleno corresponde al tipo de leche que producen, específicamente, amarillo para leche de tipo A (la de mayor calidad), celeste para leche de tipo B (leche de calidad media) y rosado para leche de tipo C (leche de menor calidad). La planta está representada por un cuadrado al centro del diagrama. Las rutas están representadas por las flechas que unen a los distintos nodos y se distinguen en colores para diferenciar a cada uno de los 3 camiones utilizados. El primer camión (flechas naranjas) parte de la planta, luego visita al nodo 4, luego al 2, 1, 5 y regresa a la planta con la leche de estos productores. Este camión al final llega solo con leche de tipo A, ya que todos los productores que visitó producían dicha leche. El segundo camión (flechas verdes) parte de la planta, luego visita al nodo 7, 3, 10, 13, 12, 9 y regresa a la planta. Los productores 3 y 13 se encuentran distantes del resto de productores que producen leche de tipo A, por lo que se requiere de un costo en ruta significativo para que el primer camión vaya por su leche. En cambio, dichos productores se encuentran cerca de los que producen leche de tipo B, por lo que el costo necesario para que el segundo camión pase por ellos es bastante bajo. Al llegar a la planta, el camión trajo consigo leche de calidad B que resulta de la mezcla de leche recolectada al pasar por productores

que producen leche de tipo A y B. El tercer camión (fechas azules) parte de la planta, visita al nodo 11, 8, 6 y regresa a la planta con leche de tipo C, ya que todos los productores que visito producen dicha leche.

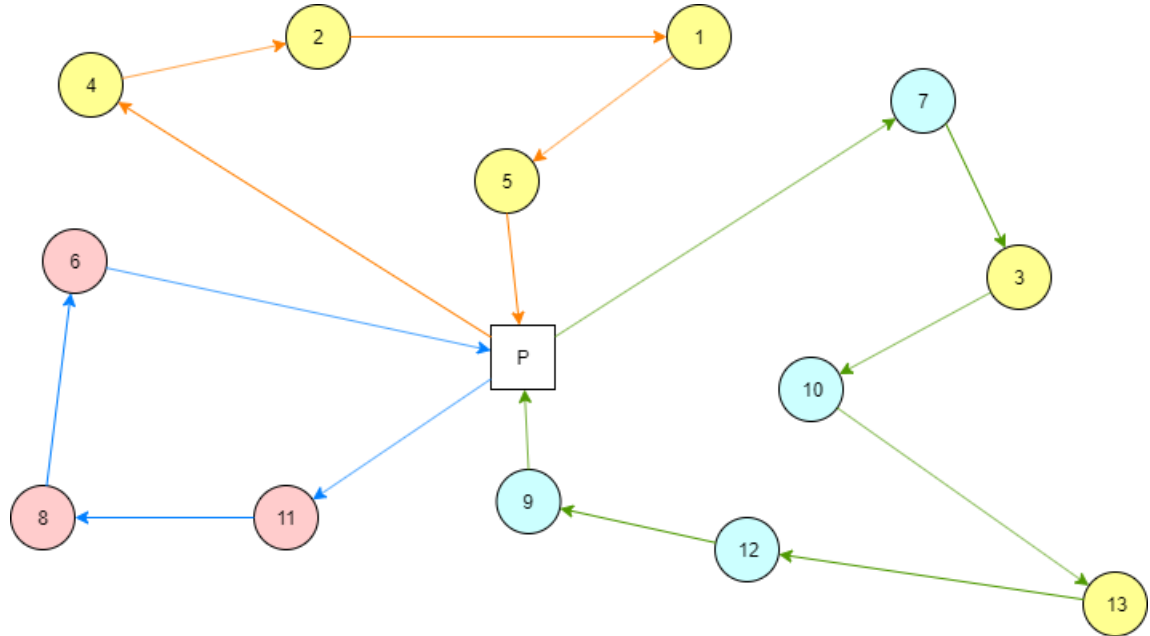


Figura 1.1: Esquema de ejemplo de una instancia con 13 productores, una planta y 3 camiones.

A continuación se presenta un modelo matemático que permitirá definir el problema de recolección de leche con mezclas. El modelo matemático de programación lineal entera mixta fue extraído de [36] que fue uno de los trabajos completos que trato este problema.

1.1. Formulación Matemática

Se formula un modelo de programación entera mixta (Mixed Integer linear Programming, MIP por sus siglas en inglés). El modelo considera cinco conjuntos de variables, donde tres de ellas son binarias. Primero se listan los parámetros del modelo, seguido de las variables de decisión que considera, las restricciones claramente explicadas y la función objetivo a maximizar.

1.2. Parámetros

Para la definición del modelo matemático se consideran los siguientes parámetros.

- A : Conjunto de arcos que representan caminos entre productores de leche.
- A^0 : Conjunto de arcos que representan caminos entre planta y productores de leche.
- N : Conjunto de productores, $N = 0, \dots, n$. Se consideran en total n productores.
- N_0 : Conjunto de productores y la planta.
- K : Conjunto de camiones
- T : Conjunto de calidades de leche
- N^t : Conjunto de productores de leche de calidad $t \in T$.
- D^t : Resultado de la mezcla de leche de calidad r con leche de calidad t .
- IT : Conjunto de pares ordenados (i, t) de productores i y leche de calidad t , donde cada cliente produce solo una calidad de leche.
- Q^k : Capacidad de cada camión k
- q_i^t : Cantidad de leche t producida por el productor i
- c_{ij}^k : Costo del viaje de cada camión k sobre el arco $(i, j) \in A \cup A^0$
- α^t : Ingreso por unidad leche de calidad t
- P^t : Requerimientos de leche de calidad t de la planta

1.3. Variables

Para este problema se considera un conjunto de cinco variables de decisión. Las primeras tres variables son de tipo binaria.

- $x_{ij}^k = \begin{cases} 1 & \text{Si el camión } k \text{ viaja directamente del nodo } i \text{ al nodo } j \\ 0 & \text{en caso contrario} \end{cases}$
- $y_i^{kt} = \begin{cases} 1 & \text{Si el camión } k \text{ recoge leche de calidad } t \text{ del productor } i \\ 0 & \text{en caso contrario} \end{cases}$
- $z^{kt} = \begin{cases} 1 & \text{Si el camión } k \text{ entrega leche de calidad } t \text{ a la planta} \\ 0 & \text{en caso contrario} \end{cases}$

Además se tiene dos variables adicionales que controlan la cantidad de leche entregada por cada camión a la planta y el total posterior a la mezcla.

- w^{kt} = Volumen de leche de calidad t que el camión k entrega a la planta.
- v^{tr} = Volumen de leche de calidad t entregada a la planta, mezclada para su uso como leche de calidad r .

1.4. Restricciones

Las restricciones para este problema se definen a continuación:

- La restricción 1.1 limita la cantidad de leche que puede recolectar cada camión de acuerdo a su capacidad. Se considera una flota heterogénea.

$$\sum_{t \in T} \sum_{i \in N: (i,j) \in IT} q_i^t y_i^{kt} \leq Q^k, \quad \forall k \in K \quad (1.1)$$

- La restricción 1.2 establece que la recolección de la leche de cada productor debe ser realizada por exactamente un camión. Esto implica que se debe recolectar la leche de todos los productores y que un productor no puede ser visitado más de una vez.

$$\sum_{k \in K_i} y_i^{kt} = 1, \quad \forall i \in N, t \in T : (i, j) \in IT \quad (1.2)$$

- La restricción 1.3 establece que cada camión debe tener como máximo una ruta la cual comienza desde la planta.

$$\sum_{j:(0_k,j,k) \in AK} x_{0_k j}^k \leq 1 \quad \forall k \in K \quad (1.3)$$

- La restricción 1.4 permite controlar el flujo para el orden de las visitas de los nodos por parte de cada camión.

$$\sum_{i:(i,j,k) \in AK} x_{ij}^k = \sum_{h:(j,h,k) \in AK} x_{jh}^k \quad \forall k \in K, j \in N_0 \quad (1.4)$$

- La restricción 1.5 establece que cada camión que visita cada granja debe detenerse y recoger su leche.

$$\sum_{p:(p,i,k) \in AK} x_{pi}^k = y_i^{kt} \quad \forall k \in K, i \in N, t \in T : (i,t) \in IT \quad (1.5)$$

- Las restricciones 1.6, 1.7, 1.8 y 1.9 establecen las reglas de mezcla de leche. La restricción 1.6 controla el tipo de leche de cada camión de acuerdo a cada una de las granjas que ha visitado. La restricción 1.7 controla que cada camión entrega en planta solo un tipo de leche. La restricción 1.8 mide la cantidad de leche entregada de cada tipo de acuerdo a la capacidad de los camiones que recolectaron cada tipo de leche. Por último, la restricción 1.9 mide la cantidad efectivamente recolectada de cada tipo de leche considerando las granjas visitadas por cada camión.

$$z^{kt} \leq 1 - \sum_{\substack{r \in D' : r \neq t \\ (i,r) \in IT}} y_i^{kr} \quad \forall k \in K, i \in N, t \in T \quad (1.6)$$

$$\sum_{t \in T} z^{kt} \leq 1 \quad \forall k \in K \quad (1.7)$$

$$w^{kt} \leq z^{kt} Q^k \quad \forall k \in K, t \in T \quad (1.8)$$

$$w^{kt} \leq \sum_{r:t \in D^r} \sum_{h \in N^r} q_h^r y_h^{kr} \quad \forall k \in K, t \in T \quad (1.9)$$

- La restricción 1.10 se encarga de forzar que cada camión se lleve toda la leche producida por cada granja a la planta.

$$\sum_{k \in K} \sum_{t \in T} w^{kt} = \sum_{(i,t) \in IT} q_i^t \quad (1.10)$$

- La restricción 1.11 se encarga de equilibrar la cantidad de leche de cada calidad que llega a la planta y la cantidad de leche de cada calidad restante después de la mezcla en la planta.

$$\sum_{r \in D^t} v^{tr} = \sum_{k \in K} w^{kt} \quad \forall t \in T \quad (1.11)$$

- La restricción 1.12 se encarga de controlar la satisfacción de las cuotas de leche de cada tipo.

$$\sum_{t \in T} v^{tr} \geq P^r \quad \forall r \in D^t \quad (1.12)$$

- La restricción 1.13 evita mezclas de leche prohibidas.

$$y_i^{kt} + y_i^{kr} \leq 1 \quad \forall (t, r) \in PM; (i, t), (j, t) \in IT \quad (1.13)$$

- La restricción 1.14 evita la aparición de sub-ciclos en las rutas de cada camión.

$$\sum_{i \in S} \sum_{j \in S} x_{ij}^k \leq |S| - 1 \quad \forall S \subseteq N, k \in K \quad (1.14)$$

- Las restricciones 1.15, 1.16 y 1.17 controlan la naturaleza de las variables de decisión del modelo planteado. La restricción 1.15 controla la naturaleza binaria de las variables asociadas a los tipos de leche recolectados y mezclados en ruta. La restricción 1.16 controla la naturaleza de la variable binaria que controla las secuencias de visitas de los camiones. La restricción 1.17 controla la naturaleza no negativa de las variables de volúmenes de leche entregados y mezclados en planta.

$$y_i^{kt}, z^{kt} \in \{0, 1\} \quad \forall i \in N, k \in K_i, t \in T : (i, t) \in IT \quad (1.15)$$

$$x_{ij}^k \in \{0, 1\} \quad \forall (i, j, k) \in AK \quad (1.16)$$

$$w^{kt}, v^{tr} \geq 0 \quad \forall k \in K; t, r \in T, r \in D^t \quad (1.17)$$

Todas estas restricciones se trabajan como restricciones duras, es decir, toda solución al problema debe cumplir con todas las restricciones impuestas.

1.4.1. Función Objetivo

Para el problema se considera como función objetivo la maximización del beneficio. Esto considera la diferencia entre la ganancia producida por la leche recolectada de acuerdo a su tipo y los costos de transporte de las rutas construidas. La primera parte de la expresión 1.18 muestra las ganancias y la segunda parte los costos de transporte.

$$Máx \sum_{i \in T} \sum_{r \in T} \alpha^r v^{ir} - \sum_{(i,j,k) \in AK} c_{ij}^k x_{ij}^k \quad (1.18)$$

Cabe destacar que no se esta considerando el costo que debe pagar la empresa por la adquisición de leche ya que este es un costo fijo que depende netamente de los requerimientos de leche de la planta y no de la ruta o como se mezcla la leche en los camiones.

1.5. Resumen

El MCwB busca encontrar rutas óptimas para un conjunto de camiones que parten desde una planta, luego recolectan leche de distintos productores y finalmente regresan a la planta para la entrega de esta. La leche recolectada se clasifica según su calidad y durante el recorrido se permite la mezcla de esta, de tal forma que la leche resultante de la mezcla es la de menor calidad de entre las mezcladas. La calidad de la leche entregada es directamente proporcional a su calidad, por lo que, durante la planificación de las rutas, además de intentar minimizar el costo de estas, se intenta maximizar el beneficio mediante la combinación de rutas cortas y mezcla según corresponda. Con esto se espera la obtención de rutas que se vean beneficiadas de una disminución en los costos a cambio de una disminución en el beneficio producido por la mezcla. Para el problema se requiere una matriz de distancia entre productores y planta, tipo y cantidad de leche de cada productos, requerimientos de la planta y capacidad de los camiones. En la figura 1.1 se muestra un ejemplo práctico de una

instancia con 13 productores y una planificación básica que considera la mezcla para obtener rutas más cortas. Matemáticamente se considera un modelo de programación entera mixta con 5 conjuntos de variables, de las cuales 3 de ellas son binarias. Se cuenta con 17 restricciones, entre las que se incluyen, restricción por no exceder la capacidad de los camiones, cumplimiento de los requerimientos de la planta, mezclas de leche prohibidas, visita de un producto por un solo camión, recolección de toda la leche del productor en una visita y naturaleza de las variables. La función objetivo busca maximizar el beneficio, el cual se calcula como la ganancia por cada litro de leche entregado a la planta menos los costos de las rutas.

Capítulo 2

Estado del Arte

Los problemas de enrutamiento de vehículos (VRP por sus siglas en inglés Vehicle Routing Problem) comprenden un conjunto de problemas de optimización combinatoria en los que un conjunto de clientes deben ser atendidos por una cierta cantidad de vehículos ya sea para envío o recogida de alguna mercancía. El VRP constituye una generalización del problema del vendedor viajero (TSP por sus siglas en inglés Travelling Salesman Problem), en la que se tiene un solo vehículo para atender a todos los clientes. Si bien ambos son NP-hard los VRP en general son más difíciles de resolver que los TSP del mismo tamaño, tal como se mencionan en JK Lenstra, AHGR Kan [26]. El problema que se trata en este documento constituye un caso específico del VRP, en el cual se permite la mezcla de mercancías de diferente tipo al interior de los vehículos de transporte.

El presente estado del arte se organiza según lo descrito a continuación. Primero se presenta una revisión de la literatura en torno al VRP, junto con sus variantes más representativas. Luego se tratará con una variante muy particular, el MCVRP (Multi-Compartment Vehicle Routing Problem) en el que los clientes pueden solicitar la recogida o entrega de mercancías de diferente tipo. En este caso particular es posible considerar dos opciones. En el primero caso las mercancías deben mantenerse separadas durante el transporte para lo que se utilizan camiones con múltiples compartimentos. La segunda opción considera una extensión que permite, además, la mezcla al interior de los camiones. Finalmente se trata con el problema de la recolección de leche con un único compartimento, en la que se considera la mezcla de

ésta, tanto durante el transporte como en la planta.

2.1. Ruteo de Vehículos en General

El clásico problema de enrutamiento de vehículos capacitados (CVRP por sus siglas en inglés Capacitated Vehicle Routing Problem), conocido simplemente como VRP, fue propuesto por primera vez en la literatura por Dantzig y Ramser en el año 1959 [13]. El problema es uno de los temas más estudiados en el campo de la investigación de operaciones al día de hoy. Dantzig y Ramser encontraron rutas óptimas para una flota de camiones que entrega gasolina desde un terminal a granel a una gran cantidad de estaciones de servicio suministradas. Más tarde en 1964 Clarke and Wright [11] generalizaron este problema a un problema de optimización lineal y desarrollaron un procedimiento iterativo para el enrutamiento de una flota de camiones de diferentes capacidades desde un depósito central a varios puntos de entrega.

El VRP clásico puede ser ampliado de muchas maneras, ya sea introduciendo nuevos aspectos o características o modificando las ya existentes. Un ejemplo de ello es el Heterogeneous Fleet Vehicle Routing Problem (HFVRP) que en vez de considerar una flota homogénea de vehículos, considera una flota heterogénea, con camiones o vehículos de distintas capacidades. En el trabajo de M Gendreau, G Laporte [21] utilizan una heurística de búsqueda tabú para resolver el HVRP. Por otro lado, PHV Penna, A Subramanian [37] fueron los primeros en proponer una metaheurística de búsqueda local iterada (ILS por sus siglas en inglés) para resolver el HVRP. A lo largo de los años se han propuesto un gran cantidad de modelos diferentes que agregan cada vez más complejidades del mundo real al modelo básico propuesto por Dantzig y Ramser y Clarke and Wright. Una de ellas es el VRP con ventanas de tiempo (VRPTW por sus siglas en inglés Vehicle Routing Problem with Time Windows), propuesto por primera vez por Solomon y Desrosiers [49] en 1988, donde se asume que las visitas a los clientes deben ocurrir en un cierto intervalo de tiempo, probablemente distinto para cada uno de ellos. Al utilizar ventanas de tiempo como restricción para la visita de los clientes se puede optar por tres enfoques. El primero donde las ventana de tiempo se tratan como restricción dura (VRPHTW por sus siglas en inglés Vehicle Routing Problem

with Hard Time Windows) y la visita de los clientes debe respetarse estrictamente. El segundo donde se trata la ventana de tiempo como restricción blanda (VRPSTW por sus siglas en inglés Vehicle Routing Problem with Soft Time Windows), agregando penalización por no cumplimiento de la ventana de tiempo en la visita de los clientes. Figliozzi, Miguel Andres (2010) [18] explica que este enfoque puede ser visto como una relajación del VRPHTW. El tercero tiene un enfoque mixto entre los dos primeros. Figliozzi [17] explica que es necesario introducir este enfoque mixto en situaciones donde el número de rutas necesarias para las ventanas de tiempo excede el número de vehículos disponibles o cuando el despachador tiene información cualitativa de la importancia relativa de las restricciones de ventanas de tiempo entre los clientes.

Con la llegada de las tecnologías en tiempo real, como Sistemas de Transformación Inteligente (ITS por sus siglas en inglés Intelligent Transformation Systems), los Sistemas Avanzados de Gestión de Flotas (AFMS por sus siglas en inglés Advanced Fleet Management Systems) y los Sistemas de Posicionamiento Global (GPS por sus siglas en inglés Global Positioning System), han hecho que los VRP dinámicos (DVRP por sus siglas en inglés Dynamic Vehicle Routing Problem), también llamados VRP en línea o en tiempo real, sean un tema relevante en los últimos años. Los DVRP propuestos por Psaraftis [42] en 1995 hacen adaptaciones a los VRP estáticos para convertirlos en problemas de enrutamiento dinámico.

Una revisión exhaustiva del DVRP fue presentada recientemente por Pillac [38]. El dinamismo se considera principalmente con respecto a las solicitudes de los clientes. Pillac, Guéret y Medaglia [39] consideran las decisiones de enrutamiento dinámico como una consecuencia de que la demanda real solo se revela al llegar a los clientes. Esto, aunque la información estocástica se conoce de antemano. Por otro lado, los tiempos de viaje dinámicos son tratados por Liao y Hu [27]. Para la actualización en tiempo real desarrollaron un algoritmo de búsqueda tabú que considera las posibles variaciones dinámicas en los tiempos de viaje.

Los VRP periódicos (PVRP por sus siglas en inglés Periodic Vehicle Routing Problem) propuestos por Mourgaya y Vanderbeck [31] en 2006 contemplan un horizonte de operación de una cierta cantidad de días, durante los cuales, cada cliente debe ser visitado una vez. Se utiliza cuando la planificación se realiza durante un período determinado y las entregas

al cliente pueden ser realizadas en diferentes días dentro de ese período, tal como proponen Campbell y Wilson [6]. La primera aplicación práctica del PVRP fue la recolección de residuos municipales por Teixeira, J., Antunes [53].

El VRP con recogida y entrega simultánea (VRPPD por sus siglas en inglés Vehicle Routing Problem with Pickup and Delivery), considera la distribución y recolección simultánea de mercancías a los clientes. Las mercancías necesitan ser recogidas de una determinada ubicación y entregadas en su destino. El servicio de recogida y entrega debe realizarse en el mismo vehículo, por lo que el lugar de recogida y la ubicación de entrega deben incluirse en la misma ruta. Esta extensión del problema de enrutamiento de vehículos capacitados fue propuesta por Tasan y Gen [52]. Un problema relacionado es el VRPB (Vehicle Routing Problem with Backhauls), donde un conjunto de vehículos que parte desde un depósito central debe entregar y recoger mercancías de diferentes clientes dentro de la misma ruta. El proceso de carga de un vehículo en el depósito para dar servicio a diferentes puntos de entrega se denomina *linehauls* y el proceso de recolección de nuevas mercancías en los puntos de recogida para luego ser entregados en el depósito se denomina *backhauls*. Con esta distinción se agrega la restricción de que primero se deben llevar a cabo los *backhauls* y luego los *linehauls*. Pradenas, Oportus y Parada [40], usando una búsqueda dispersa, analizaron problemas de la literatura con hasta 100 clientes distribuidos al azar.

En el VRP abierto (OVRP por sus siglas en inglés Open Vehicle Routing Problem) los vehículos no están obligados a regresar al depósito central después de visitar al último cliente. Si regresan, deben visitar a los mismos clientes en el orden inverso. Además, el OVRP a menudo tiene dos objetivos de optimización: minimizar el número de vehículos utilizados y, dado este número de vehículos, minimizar la distancia total recorrida. En la práctica, el OVRP se produce cuando la flota de vehículos no es propiedad de la propia empresa o cuando la flota de vehículos disponible no puede satisfacer la demanda de sus clientes, de modo que, parte de las actividades de distribución se contraten a un proveedor externo. Una variante parecida es el enrutamiento del vehículo con viajes múltiples (VRPM por sus siglas en inglés Vehicle Routing Problem with Multiple Trips), donde se relaja la restricción de que el vehículo debe regresar al depósito luego de terminar el despacho a los clientes. El

primer estudio que incluye la idea de viajes múltiples para VRP se debe al trabajo de Fleischmann [20]. Acá se permite que los vehículos puedan realizar más de un viaje en un período de planificación y que realicen viajes más largos, pagando alguna penalización. Su algoritmo de solución es un algoritmo basado en poblaciones que utiliza enfoques de búsqueda tabú y empaquetado de bandejas. Sen y Bülbül [46] en 2008 estudian y comparan varios trabajos entorno al VRPM.

El enrutamiento de vehículo capacitado acumulativo (CCVRP por sus siglas en inglés Cumulative Capacitated Vehicle Routing Problem) es una variación del conocido CVRP cuyo objetivo tradicional de minimizar la distancia total o el tiempo recorrido por los vehículos se reemplaza minimizando la suma de los tiempos de llegada a los clientes. Lysgaard y Wohlk [30] publicaron el primer algoritmo exacto para resolver el CCVRP, basado en *branch-and-cut-and-price* para obtener soluciones óptimas al problema.

Los VRP con entrega dividida (SDVRP por sus siglas en inglés Split Delivery Vehicle Routing Problem), corresponden a una clase de problemas de enrutamiento donde cada cliente puede ser atendido por más de un vehículo. El SDVRP fue introducido formalmente por Dror y Trudeau [16] y ha recibido una atención cada vez mayor con el tiempo. Archetti y Speranza [1] describen las propiedades y variantes del SDVRP, y hacen un análisis de varias propuestas de enrutamiento que consideran las entregas divididas.

Todos estos trabajos consideran un solo depósito, pero hay situaciones más realistas donde es necesario considerar que la distribución de los productos es realizada desde varios depósitos hasta los clientes finales (MDVRP por sus siglas en inglés Multi-Depot Vehicle Routing Problem) tal como se menciona por Crevier, Cordeau, y Laporte [12] en 2007.

El problema de ruteo multiproducto es una versión del VRP que considera la recolección/entrega de productos de diferente tipo. A continuación se describen en detalle acercamientos para esta versión del VRP que tiene mucho en común con el problema de enrutamiento con mezclas estudiado en este trabajo de memoria.

2.2. Ruteo de Vehículos con Múltiples Productos

La mayor parte de la literatura sólo considera un tipo de producto. Sin embargo, en muchas aplicaciones prácticas del VRP, hay más de un tipo de producto involucrado. De acá nace el problema de enrutamiento de vehículos multiproductos (MPVRP por sus siglas en inglés Multi-Product Vehicle Routing Problem), que permite reducir costos mediante el transporte de diferentes productos en un mismo vehículo dentro de un ciclo de entrega (Liu, Lei y Park [28]). Cada cliente, ubicado en una zona geográfica determinada, tiene un conjunto de demandas, una para cada producto diferente. Cada tipo de producto tiene un tamaño diferente, mientras que el camión de entrega tiene un tamaño de carga dado. El problema es similar al de entrega dividida (SDVRP por sus siglas en inglés Split Delivery Vehicle Routing Problem), donde el pedido de un cliente puede ser entregado por múltiples vehículos mediante envíos parciales. La diferencia es que en el SDVRP las entregas parciales corresponden al mismo tipo de producto, mientras que en el MPVRP cada envío parcial debe contener un tipo de producto distinto. Cuando se trabaja en un marco de enrutamiento de vehículos multiproductos, en muchos casos, los diferentes tipos de productos no pueden mezclarse en el mismo compartimento durante el transporte. Teniendo en cuenta la eficiencia del transporte, en vez de dividir las entregas en varios envíos y utilizar varios vehículos, se puede dividir la capacidad de un vehículo en varios compartimentos para dar servicio a diferentes tipos de productos simultáneamente. De acá nacen los problemas de enrutamiento de vehículos con múltiples compartimentos (MCVRP por sus siglas en inglés Multi-Compartment Vehicle Routing Problem), acuñados por Brown y Graves en 1981 [4].

Kaabachi, Yahyaoui, Krichen y Dekdouk [23] utilizaron el MCVRP en el contexto de la distribución de combustible para el reabastecimiento de una gasolinera. Cada una solicita un cantidad conocida de uno de tres hidrocarburos: *Super*, *Regular* y *Diesel*. Para la distribución se utilizaron camiones cisterna con compartimentos de capacidades fijas dedicados a cada tipo de hidrocarburo. Kaabachi propusieron un algoritmo híbrido de colonias de abejas artificiales para resolver 28 instancias con un rango de 50 a 200 clientes. Se obtuvieron los óptimos globales en tiempos inferiores a 2 segundos, en comparación con el paquete de software de optimización CPLEX que tomó alrededor de 55 segundos para algunas instancias. La versión ejecutada con CPLEX, separa el problema en 3, es decir, considera un VRP

por cada tipo de hidrocarburo y la utilización de un camión. Luego se suman las distancias recorridas por los tres camiones.

Muyldermans y Pang [33] en 2010 presentaron un procedimiento de tres fases para resolver el MCVRP. En primer lugar, se obtiene una solución inicial factible utilizando el algoritmo de ahorro Clarke y Wright [11]). Esta solución se mejora posteriormente en una fase de búsqueda local que explora movimientos conocidos: *2-opt*, *cross*, *exchange* y *relocate*. Aplica los mecanismos de las listas de vecinos y el marcado para acelerar las búsquedas. Por último, se ejecuta un metaheurística de búsqueda local guiada [54]. Con su procedimiento y análisis concluyeron que en muchos casos la corecogida con múltiples compartimentos es mejor que la recolección simple, cuando el número de mercancías es mayor, la capacidad del vehículo aumenta, los artículos son menos voluminosos, los clientes solicitan todas las mercancías, la densidad del cliente es menor y el depósito está más céntrico respecto al área de distribución. Para obtener estas conclusiones utilizaron 39 instancias repartidas 3 conjuntos con una cantidad de clientes que va de los 50 a los 483. Los vehículos utilizados cuentan con 2, 3 o 4 compartimentos. Lograron encontrar el óptimo global para la instancia más grande (483 clientes) en aproximadamente 11 minutos en comparación con los 34 minutos de Memetic Algorithm y los 107 minutos de Tabu Search con los cuales se compararon.

Dependiendo de la industria en donde se trabaje, hay ocasiones donde se introducen nuevas variables al modelo. Este es el caso de una empresa de distribución de la cadena de frío, donde además de distribuir distintos productos que no se pueden mezclar, cada uno debe ser transportado a una temperatura determinada y los tiempos de viaje y entrega deben ser estrictos. De acá nace el problema de enrutamiento de vehículos multi-compartimento con ventanas de tiempo (MCVRPTW por sus siglas en inglés Multi-Compartment Vehicle Routing Problem with Time Windows) tratado por Chen, Liu y Langevin [9]. Para resolver el problema proponen el algoritmo de búsqueda en grandes vecindarios adaptivos (*Adaptive Large Neighborhood Search – ALNS*), donde en cada iteración se elige una heurística entre varias para destruir y reparar las diversas rutas. Por otro lado, Chen y Shi [8] proponen y comparan dos enfoques de solución, con uno de ellos la optimización híbrida de enjambre de partículas (HPSO por sus siglas en inglés Hybrid Particle Swarm Optimization) con Simulated annealing, y el otro es optimización por enjambre de partículas (Particle Swarm

Optimization – PSO). Trabajaron con instancias de 10 a 100 clientes. En la más grande obtuvieron el óptimo global en menos de 50 segundos.

Reed, Yiannakou y Evering [43] utilizaron un sistema de colonias de hormigas (ACS por sus siglas en inglés Ant Colony System) para resolver el problema del enrutamiento de vehículos capacitados y algunas variantes como el MCVRP asociado con la recolección de residuos de reciclaje domésticos. Los vehículos utilizados suelen tener cuatro compartimentos separados, usualmente, vidrio, papel, plástico y metal. Lograron encontrar el óptimo global en una instancia de 50 nodos en menos de 3 segundos y una de 199 nodos en 40 minutos.

Silvestrin y Ritt [48] estudian el problema de enrutamiento de vehículos con múltiples compartimentos. Para resolver el MCVRP proponen un algoritmo de búsqueda tabú iterada (ITS por sus siglas en inglés Iterated Local Search) en la que se aplica repetidamente una perturbación para escapar de un mínimo local seguida de una búsqueda tabú para encontrar otro mínimo local. Resolvieron instancias de 50 a 100 clientes en menos de 10 minutos. Para instancias de alrededor de 400 clientes obtuvieron el óptimo global en aproximadamente 1 hora.

2.3. Ruteo de Vehículos con Múltiples Productos y Mezclas

En el despliegue de vehículos con múltiples compartimentos para la recogida de residuos separados, cuando los clientes a visitar generan diferentes tipos de residuos, las empresas de recogida tienen la flexibilidad de desplegar vehículos tradicionales sin particionar, que recogen un tipo de residuo a la vez (*recogida separada*); pueden desplegar vehículos con múltiples compartimentos para recoger diferentes tipos de residuos (*corecogida*); o utilizar alguna de las modalidades anteriores, pero permitiendo que un camión o un compartimento de éste pueda albergar varios tipos de residuos. Bajo estas modalidades, no hay claridad en la literatura respecto a las circunstancias en la que la corecogida es mejor, respecto al largo de las rutas, en comparación con la colección separada o la mezcla de residuos. Mientras que las rutas para la colección separada se pueden diseñar utilizando un algoritmo CVRP estándar,

la evaluación de la eficiencia del ruteo de la colección compartida requiere un procedimiento de solución MCVRP y la mezcla requiere de la incorporación de nuevas restricciones respecto a que tipos de residuos pueden mezclarse y como afecta la mezcla a la calidad de las soluciones.

Paredes-Belmar, Bronfman y Marianov [34] proponen un enfoque de enrutamiento de vehículos multiproductos para reducir los costos y el riesgo durante el transporte de cargas de distintos tipos de material peligroso (*HAZardous MATerial* – *HAZMAT*). Cada material tiene un nivel de riesgo que es directamente proporcional al tamaño de la población expuesta. En este enfoque se disminuyen los riesgos en comparación al estándar en el que se utiliza varios camiones para transportar un solo material. La reducción del riesgo se debe al hecho de que un camión cargado con múltiples productos expone a la misma población que también estaría expuesta por el material más riesgoso que se transporta. Dicho de otro modo, el riesgo asociado a un camión es el del material con mayor riesgo que éste lleva, independientemente de cuántos materiales menos riesgosos se transporten simultáneamente. Se supone que cada cliente o nodo generador produce un solo tipo de material y que los desechos incompatibles no pueden ser transportados juntos. Paredes-Belmar diseñaron una metodología que incluye el precálculo de los mejores caminos entre todos los nodos relevantes de la red para diferentes riesgos de residuos y la formulación de un modelo de programación entera para resolver una instancia real con 167 fábricas distribuidas a lo largo de 7 zonas. Para la zona más complicada (32 fábricas) se encuentra un buen óptimo local en 50,000 segundos en algunas pruebas, en tan solo 300 segundos para otras. Para la segunda más complicada (36 fábricas) se encuentra en 24,000 segundos en algunas prueba, mientras que en otras en tan solo 14 segundos. Para las demás zonas, los tiempos van de 1 a 100 segundos.

2.4. Recolección de Leche

En 1994, Sankaran y Ubgade [45] fueron los primeros en abordar el problema de la recolección de leche como un caso especial de VRP con una flota de vehículos homogénea. Butler [5] proponen el uso de camiones con múltiples compartimentos. Igbaria [51] y Dayarian [14] formularon el problema de enrutamiento de vehículos con una flota fija heterogénea

(HFFVRP por sus siglas en inglés Heterogeneous Fixed Fleet Vehicle Routing Problem). Prasertsri y Kilmer [41] introducen al problema ventanas de tiempo. Hoff y Løkketangen [22] consideran el problema de enrutamiento del vehículo con recolecciones y entregas (VRPPD por sus siglas en inglés Vehicle Routing Problem with Pickup and Delivery) donde el mismo cliente puede requerir tanto una entrega como un retiro. Claassen y Hendriks [10] utilizaron un enfoque basado en investigación de operaciones que permite resolver varios objetivos en el problema de recolección de leche de cabra en los Países Bajos. Todos estos autores representaron el problema de recolección de leche considerando un solo tipo de leche. Hoff y Løkketangen [22] resuelven el problema de ruteo de remolque (TTVRP por sus siglas en inglés Truck and Trailer Vehicle Routing Problem) utilizando un algoritmo de búsqueda tabú para una instancia real de Noruega. En el TTVRP los camiones no recogen leche directamente en las granjas, sino que los mismos productores entregan la leche en puntos de recogida mientras un camión hace un recorrido secundario a granjas distantes. Caramia y Guerriero [7] resuelven el TTVRP para la recolección de 4 tipos de leche (con distintas calidades), sin considerar su mezcla. Para ello se utilizaron camiones con compartimientos que permitieron separar los distintos tipos. Primero aplicaron búsqueda local que asigna productores a cada compartimento y luego se minimiza la cantidad de camiones a utilizar. Mumtaz [32] resuelven un problema de la recolección de leche de un solo tipo, la cual es recogida exclusivamente en centros de acopio donde los productores van a dejar la leche. Para la resolución se utiliza una heurística orientada a instancias pequeñas. Basnet [3] proponen un algoritmo exacto y un procedimiento heurístico para asignar camiones a rutas predefinidas en Nueva Zelanda, cuyo objetivo es minimizar el tiempo en que el último camión entrega su carga a la planta.

En 2015, Lahrichi [24] utilizó búsqueda tabú para resolver el problema de recolección de leche. Los resultados obtenidos para instancias pequeñas reflejan mejoras poco significativas. Dooley [15] y Sethanan y Pitakaso [47] utilizaron algoritmos genéticos para resolver el problema considerando múltiples calidades de leche. Sethanan y Pitakaso [47] en su trabajo futuro propone la mezcla de leche en los distintos puntos de recogida.

Finalmente, en 2016 G. Paredes-Belmar [36] presentan por primera vez el problema de la recolección de leche con mezcla (MCwB por sus siglas en inglés Milk Collection with

Blending), que permite combinar leche de distintas calidades al momento de la recolección. Esto disminuye los ingresos debido a la calidad de la leche, pero reduce los gastos de transporte. Ellos proponen un heurística de tres etapas diseñada para resolver una instancia con 500 productores ubicados en el sur de Chile. Consideran la mezcla de 3 tipos de leche y 100 camiones para la recogida de esta. Obtuvieron los mejores resultados, dividiendo el terreno en 13 zonas geográficas con tiempos cercanos a las 16 horas. En 2017, publican otro trabajo [35] donde se establecen puntos de recolección para el MCwB, donde varios productores pequeños alejados de la planta podrían almacenar la leche producida en dichos puntos. De esta forma los camiones podrían recolectar el total de leche de un grupo en una sola parada. Se utiliza un algoritmo de *branch-and-cut* para resolver el problema de la optimalidad y un procedimiento heurístico de tres fases que incluye un algoritmo basado en colonia de hormigas. En su trabajo resolvieron una instancia con 500 productores (la misma que en su trabajo anterior) mediante 4 esquemas, entre los cuales se combina el utilizar o no mezcla de leche y el utilizar o no puntos de recogida. Los mejores resultado se obtienen cuando se considera tanto la mezcla de leche como puntos de recogida. En este caso se obtienen mejores resultados que en su trabajo anterior con tiempos cercanos a 34 horas.

2.5. Resumen

El problema de enrutamiento de vehículos (VRP por sus siglas en inglés Vehicle Routing Problem) comprenden un conjunto de problemas de optimización combinatoria en los que un conjunto de clientes deben ser atendidos por una cierta cantidad de vehículos ya sea para envío o recogida de alguna mercancía. Fue propuesto por primera vez en la literatura por Dantzig y Ramser en el año 1959 [13]. Este puede ser visto como un generalización al problema del vendedor viajero (TSP por sus siglas en inglés Travelling Salesman Problem). Debido a su uso en varias aplicaciones de transporte se fueron desarrollando variantes, que agregan complejidades del mundo real y permiten utilizarla en un gran número de situaciones.

A continuación se agrupan la variantes del VRP por sus características:

Problemas del VRP en general:

- **CVRT**: Capacitated Vehicle Routing Problem
- **VRPTW**: Vehicle Routing Problem with Time Windows
- **VRPHTW**: Vehicle Routing Problem with Hard Time Windows
- **VRPSTW**: Vehicle Routing Problem with Soft Time Windows
- **DVRP**: Dynamic Vehicle Routing Problem
- **PVRP**: Periodic Vehicle Routing Problem
- **VRPPD**: Vehicle Routing Problem with Pickup and Delivery
- **OVRP**: Open Vehicle Routing Problem
- **VRPM**: Vehicle Routing Problem with Multiple Trips
- **CCVRP**: Cumulative Capacitated Vehicle Routing Problem
- **SDVRP**: Split Delivery Vehicle Routing Problem

Problemas que incluyen el ruteo de múltiples productos:

- **MDVRP**: Multi-Depot Vehicle Routing Problem
- **MPVRP**: Multi-Product Vehicle Routing Problem
- **SDVRP**: Split Delivery Vehicle Routing Problem

Problemas que incluyen el ruteo de múltiples productos y mezclas:

- **MCVRP**: Multi-Compartment Vehicle Routing Problem
- **MCVRPTW**: Multi-Compartment Vehicle Routing Problem with Time Windows

Problemas que tratan la recolección de leche:

- **HFFVRP**: Heterogeneous Fixed Fleet Vehicle Routing Problem

- **VRPPD:** Vehicle Routing Problem with Pickup and Delivery
- **TTVRP:** Truck and Trailer Vehicle Routing Problem
- **MCwB:** Milk Collection with Blending

Capítulo 3

Propuesta

Para resolver el problema de enrutamiento de vehículos con mezcla (MCwB) se proponen dos algoritmos basados en búsqueda local. Un algoritmo basado en *hill-climbing* con reinicio y una variante de este esquema llamado *iterated local search*. En ambos casos, se comienza el proceso de búsqueda con un generador de soluciones aleatorias factibles que construya soluciones que cumplan con la restricción de no superar la capacidad de los camiones para una ruta y con las cuotas de la planta para cada tipo de leche. Mediante la aplicación de sucesivos movimientos se mejora la calidad de las soluciones hasta obtener el óptimo local de la zona donde se está buscando. Un algoritmo *iterated local search*, en vez de generar una solución aleatoria factible en cada reinicio, aplica una perturbación a la solución actual. Esto produce una menor exploración del espacio de búsqueda que en *hill-climbing*, pero puede ser suficientemente significativa para explorar otros espacios de búsqueda cercanos a la solución actual que permita escapar del óptimo local actual. En ambos casos se utiliza la misma representación de las soluciones, función de evaluación y procesos de inicialización de soluciones. En esta sección se explican primero las componentes en común de los acercamientos. Se explica en detalle la representación utilizada, luego se describe la función de evaluación y el proceso de inicialización. A continuación, se explican a fondo las componentes de los algoritmo *hill-climbing* e *iterated local search* implementados para resolver el MCwB.

3.1. Representación

El MCwB tiene como objetivo encontrar rutas óptimas para un conjunto de camiones que deben visitar ciertas granjas para recoger la leche que producen y llevarla a la planta. Dado esto, una solución al MCwB debe contener un listado de rutas, una para cada camión, en la que cada una muestre las secuencias de granjas visitadas. En el algoritmo, esto se presenta mediante un conjunto de vectores. Cada uno de estos vectores internos representa la ruta realizada por un camión. Cada ruta esta compuesta por un listado de números enteros, representando al identificador único de cada granja. Cada solución posee n rutas, donde n corresponde a la cantidad de camiones disponibles. La ruta del primer camión se muestra en la parte superior del vector de rutas, luego se añade la ruta del siguiente camión y así sucesivamente hasta llegar a la n -ésima ruta, tal como se muestra en la figura 3.1.

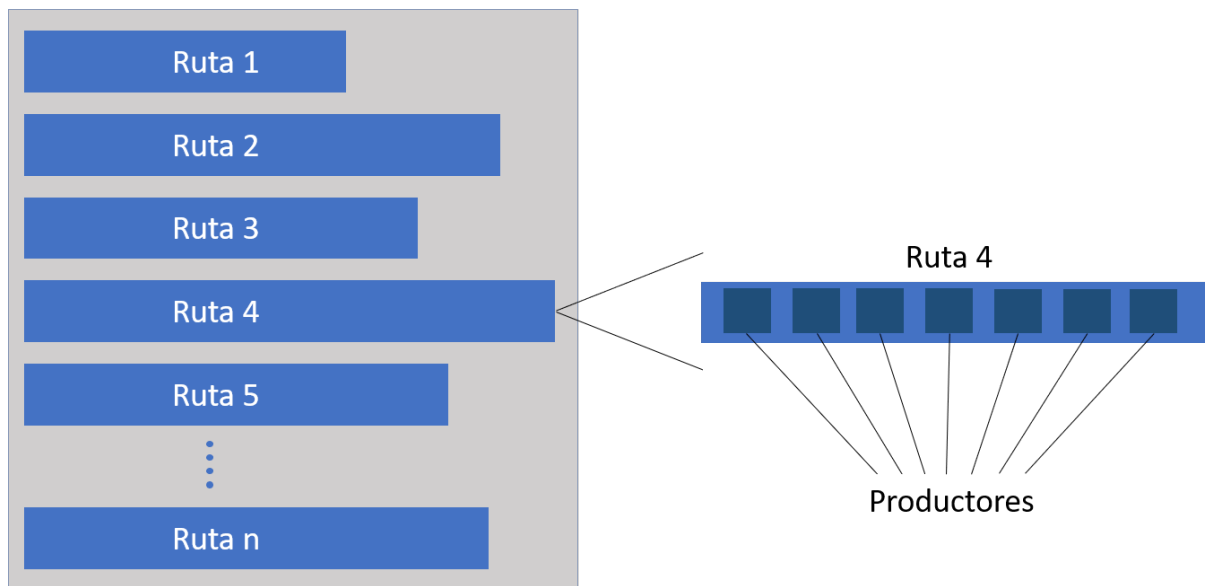


Figura 3.1: Esquema representativo de una solución para el MCwB, con n camiones.

Por temas de eficiencia de espacio de almacenamiento se ha omitido el nodo 0 (la planta) al comienzo y final de cada ruta, ya que cada ruta siempre comienza y termina en la planta. Así, en cada ruta solo se listan las granjas atendidas por cada camión en el orden en que son visitadas. Cabe destacar que el largo de cada ruta es variable, y depende principalmente de la capacidad de cada camión y de como el algoritmo asigne granjas a visitar en cada momento.

Cuanto mayor sea la capacidad de un camión mayor podrá ser la cantidad de nodos visitados. Al contrario, cuanto mayor sea la leche que producen las granjas ya visitadas por un camión, menor será la cantidad de granjas que se podrán visitar posteriormente. Dada esta variabilidad en el largo es que se optó por una representación de vector de vectores y no a una simplemente matriz. Esta representación permite ocupar menos espacio que una representación matricial, ya que con una matriz se debería dejar establecido desde un principio el máximo largo de una ruta, dejando espacio sin ocupar en rutas más cortas.

A continuación se muestra el ejemplo de la representación de una solución aleatoria para una instancia en la que se cuenta con tres camiones y se tienen nueve granjas por visitar:

$$Solution = [[5, 8, 9, 6], [4, 2], [3, 7, 1]]$$

El primer camión parte en la planta (nodo 0), visita a las granjas 5, 8, 9 y 6 en ese orden y regresa a la planta. El segundo camión comienza su recorrido en la planta, visita a las granjas 4 y 2 en ese orden y regresa a la planta. Por último, el tercer camión comienza su recorrido en la planta, visita a las granjas 3, 7 y 1 en ese orden y regresa a la planta. Es importante recordar que cada solución debe visitar a todos los nodos del problema.

3.2. Función de Evaluación

La calidad de las soluciones se determina mediante la función de evaluación que suma el beneficio por cada litro de leche que fue entregado a la planta según sus requerimientos y resta el costo de la ruta de cada camión. Adicionalmente se consideran dos penalizaciones. Una penalización por cada litro de leche que no es entregado a la planta según sus requerimientos, multiplicado por su factor de penalización ($litros_cuota * f_cuotas$). La segunda penalización viene dada por cada litro de exceso en la capacidad de los camiones para una ruta determinada, multiplicado por su factor de penalización ($litros_exceso * f_capacidad$). Estas penalizaciones tienen por objetivo dar la posibilidad al algoritmo de trabajar con soluciones infactibles durante el proceso de búsqueda. A continuación se muestra la formula para el cálculo de la calidad de una solución:

$$Quality = milk_profit - route_cost - litros_cuota * f_cuotas - litros_exceso * f_capacidad$$

Para aplicar las penalizaciones por incumplimiento de cuota, primero se verifica si en la solución falta leche de algún tipo para cumplir las cuotas de la planta. En caso afirmativo, se verifica si hay excedentes de leche de una calidad superior, para así mezclar lo que haga falta. Esta mezcla se considera siempre y cuando no provoque un incumplimiento de las cuotas de la leche de mayor calidad. En otras palabras, si hay escasez de leche de algún tipo, se busca excedentes en el tipo de leche de un nivel superior a la actual. Si la leche faltante para suplir la escasez es mayor que la leche disponible de exactamente un nivel superior que la actual, se completa con lo que este nivel de leche pueda aportar, y se procede a buscar la forma de completar la leche faltante en los niveles de calidad superiores. Cabe destacar que si la escasez se da en la leche de mayor calidad, el algoritmo no realizará ninguna acción, ya que no hay un tipo de leche de mayor calidad que pueda ser mezclada para suplir la falta.

3.3. Generación de Solución Inicial

La generación de la solución inicial consiste en asignar un conjunto de productores a distintos camiones en forma aleatoria, teniendo en consideración algunas restricciones que aseguren, como mínimo, que no se exceda la capacidad de los camiones e intentando agrupar en un mismo camión productores que produzcan un mismo tipo de leche. El pseudocódigo con la solución implementada para esta tarea se muestra en el algoritmo 1.

Primero se define el arreglo solución (*solution*) que contendrá un listado de las rutas de todos los camiones. Este será el que se retorne al finalizar la ejecución de la función. Como al principio todas las rutas estarán vacías, se inicializa con un número de vectores vacíos igual a la cantidad de camiones disponibles en la instancia. Luego se desordena el arreglo con la capacidad de los camiones, con eso nos aseguramos quitar cualquier sesgo en el proceso de asignación de productores a camiones. Este arreglo de camiones se guarda en la variable *trucks*. Se define un arreglo con el número de productores totales (*farms*). Este es el que identifica a los productores que aún no hayan sido asignados a un camión. En el ciclo *while* principal, se irán quitando elementos de este arreglo hasta que quede desocupado. En este momento, sera cuando el arreglo solución (*solution*) este completo con todos los productores asignados a un camión. Se define una tolerancia (*TOL*) que indica la cantidad de litros de

leche en que los camiones pueden exceder de su capacidad. Este comienza con un valor cero, pero como verá más adelante su valor va disminuyendo hacia valores negativos. Por último, se inicializa un contador (*truck_counter*) en cero que indica la cantidad de productores que han sido asignados a camiones de la solución.

Algorithm 1 *random feasible solution*

```

1: procedure RANDOM_FEASIBLE_SOLUTION()
2:   solution  $\leftarrow$  empty int vectors
3:   trucks  $\leftarrow$  mess up truck
4:   farms  $\leftarrow$  farms with production and quality
5:   TOL  $\leftarrow$  0
6:   truck_counter  $\leftarrow$  0
7:   while length farms > 0 do
8:     current_farm  $\leftarrow$  select random farm
9:     for trucks as truck do
10:      if truck solution capacity – current_farm production >= TOL and
        milk quality solution == current_farm quality then
11:        add current_farm to truck from solution
12:        remove current_farm from farms
13:        truck_counter ++
14:        current_farm  $\leftarrow$  select random farm
15:      end if
16:    end for
17:    if truck_counter == 0 then
18:      TOL – = 10
19:      continue
20:    else
21:      truck_counter  $\leftarrow$  0
22:    end if
23:  end while
24:  return solution
25: end procedure

```

En el ciclo *while* principal, se verifica en todo momento que la variable *farms* tenga elementos, ya que en cada iteración se irán quitando los productores hasta dejarlo vacío. Primero elige un productor en forma aleatoria (*current_farm*), el cual, será el candidato a ser asignado a la ruta de un camión. Luego comienza un ciclo *for*, que busca entre todos los camiones, uno que permita recolectar toda la leche del productor seleccionado y cuya calidad sea igual a la leche que ya hay dentro (si la hubiera). Solo en este momento se agrega dicho productor al camión perteneciente a la solución. Además se quita la granja de la variable *farms* y se incrementa el contador *truck_counter* en uno. Finalmente se selecciona otro productor en forma aleatoria (*current_farm*) y se repite el ciclo hasta que no queden más productores en el arreglo *farms*. Antes de terminar una iteración del ciclo se hace una comprobación importante. Si *truck_counter* es igual cero, significa que el productor actual no pudo insertarse en ningún camión, ya sea por falta de capacidad o tipo de leche incompatible. Cuando esto sucede, la variable *TOL* se decrementa en un factor de 10. Posterior a ello, se ejecuta la sentencia *continue*. En caso contrario, se reinicia la variable *truck_counter* a cero.

Como último detalle, comentar que la tolerancia se definió para dar más prioridad al cumplimiento de las cuotas de planta que a la capacidad de los camiones. Por ejemplo, si falta asignar un solo productor, pero hay capacidad para moverlo a un camión que contiene leche de mayor calidad. Moverlo a dicho camión conllevaría bajar la calidad de leche de todo ese camión, lo cual, es menos beneficioso que sobrepasar la capacidad del camión que ya tiene ese tipo de leche por unos pocos litros. La tolerancia ayuda a obtener rutas de mayor calidad al comienzo en casos bordes como que se acaba de describir.

3.4. Hill Climbing

Se implementa una solución basada en *hill-climbing* con Primera Mejora para la resolución del problema de enrutamiento de vehículos multiproducto con mezcla.

Un algoritmo *hill-climbing* consiste en un proceso que toma una solución inicial generada de forma aleatoria que luego busca mejorarla mediante una sucesión de movimientos. Los

movimientos aplicados a una solución generan un vecindario, del cual se selecciona el primer vecino cuya calidad sea mejor que la de la solución actual. Por esto el nombre primera mejora. Este vecino pasa a ser la nueva solución actual. Luego, el proceso se repite hasta que dada una solución, ningún vecino tiene una calidad superior a ésta. En dicho caso se esta ante un óptimo local. Llegado a este punto es posible repetir el proceso completo mediante la generación de otra solución aleatoria. En el libro *Local Search in Combinatorial Optimization* [25] publicado en 1997, se habla en forma extensa acerca de este algoritmo.

Algorithm 2 *hill-climbing*

```

1: procedure HILL_CLIMBING(max_quality, max_time, f_cuotas, f_capacidad)
2:   global_local, extra_local, intra_local
3:   best_solution  $\leftarrow$  random_feasible_solution()
4:   best_quality  $\leftarrow$  evaluate(solution)
5:   while best_solution < max_quality and current_time < max_time do
6:     global_local  $\leftarrow$  false
7:     while not global_local do
8:       extra_local  $\leftarrow$  extra_local_search()
9:       intra_local  $\leftarrow$  intra_local_search()
10:      global_local  $\leftarrow$  extra_local or intra_local
11:    end while
12:    if quality > best_quality then
13:      best_solution  $\leftarrow$  current_solution
14:      best_quality  $\leftarrow$  quality
15:    end if
16:    solution  $\leftarrow$  random_feasible_solution()
17:    quality  $\leftarrow$  evaluate(solution)
18:  end while
19:  return best_solution
20: end procedure

```

El pseudocódigo de la propuesta se muestra en el algoritmo 2. En términos generales posee la estructura de un algoritmo *hill-climbing*, pero utiliza dos procesos de generación

de vecindarios basados en los movimientos *extra-rutas* e *intra-rutas*. La estructura de la estrategia se explica a continuación.

El algoritmo recibe como parámetros el tiempo máximo de ejecución *end_time* y calidad máxima a encontrar *max_quality*. Ambos parámetros determinan los criterios de parada de la búsqueda local. Adicionalmente, se necesitan los factores de penalización por incumplimiento en cuotas de la planta (*f_cuotas*) y capacidad de los camiones (*f_capacidad*), tal como se explico en el apartado de *Función de Evaluación*.

El proceso comienza con la declaración de las variables *global_local*, *extra_local* e *intra_rutas*. La variable *global_local* tiene por objetivo monitorear el estancamiento del proceso de búsqueda global. Las variables *extra_local* e *intra_local* permiten monitorear el estancamiento de las búsquedas locales *extra-rutas* e *intra-rutas*. Luego se declaran las variables *best_solution* que almacena la mejor solución encontrada en todo el proceso y *best_quality* que almacena su función de evaluación. Para inicializar *best_solution* se utiliza la función *random_feasible_solution()* que genera una solución aleatoria factible, que, en la mayoría de los casos, es una solución en la que no se excede la capacidad de ningún camión y en la que se cumplen todas las cuotas de la planta. La función *evaluate()* evalúa la calidad de dicha solución utilizando la matriz de costos, capacidad de los camiones, requerimientos de la planta, tipo y cantidad de leche que produce cada granja, y factores de penalización.

El primer ciclo iterativo (ciclo restart), que comienza en la línea 5, se mantiene activo mientras no se cumplan los criterios de parada. Es decir, que la calidad de la mejor solución encontrada hasta el momento sea inferior a *max_quality* y que el tiempo ejecución sea inferior a *end_time*. Se setea la variable *global_local* como *false* para detectar el estancamiento de este proceso.

El segundo ciclo iterativo (línea 7) se mantiene activo mientras la variable *global_local* no sea verdadera. Esto se da cuando los procesos de búsqueda local *extra-rutas* e *intra-rutas* no encuentran una solución de mejor calidad que la actual. Ambos procesos de búsqueda se encapsulan en funciones *extra_local_search* e *intra_local_search* respectivamente y retornan un booleano que indica si hubo mejoramiento de la actual. Estos valores se guardan en las variables *extra_local* e *intra_local* que permiten detectar el estancamiento de estos procesos.

El procedimiento descrito se muestra entre las líneas 7 y 11 del algoritmo 2.

Al término del segundo ciclo iterativo, la solución resultante es comparada con la máxima encontrada hasta el momento (de todos los ciclos restarts que transcurrieron previamente). En caso de ser de mejor calidad, esta es guardada en la variable *best_solution*. Con esta última verificación se ha concluido un ciclo restart, y el algoritmo esta listo para generar una nueva solución aleatoria y repetir el proceso. El procedimiento descrito se muestra entre las líneas 12 y 15 del algoritmo 2.

Al término del primer ciclo iterativo, se genera una nueva solución aleatoria para reemplazar a la actual (que puede o no ser igual a la mejor solución), luego esta se evalúa con la función *evaluate()* (líneas 16 y 17) y se da comienzo a un nuevo ciclo siempre y cuando aún no se haya llegado a la máxima calidad o al tiempo límite.

Al finalizar el algoritmo, ya sea por que se llego a la máxima calidad o al tiempo máximo de ejecución, se retorna la mejor solución encontrada en todo el proceso (línea 19).

Como se mencionó anteriormente, al interior del segundo ciclo se encuentran los procesos de búsqueda local: búsqueda local *intra-rutas* y *extra-rutas*. En la búsqueda local *intra-rutas* se genera un vecindario compuesto por el movimiento de cada nodo de una ruta a la mejor posición de todas las demás rutas. En la búsqueda local *extra-rutas* se genera un vecindario compuesto por cambiar el orden de visitar los nodos dentro de una misma ruta entre cualquier par de nodos de la misma. Los procedimientos *intra-rutas* y *extra-rutas* se describen en detalle en las siguientes secciones. Ambos procesos de búsqueda se explicarán a detalle a continuación.

Extra-Rutas

En esta fase del algoritmo se busca el mejoramiento de la solución a través de la reasignación de granjas a otros camiones, es decir, para generar el vecindario se intenta mover la visita de un productor específico a cualquier otro camión que haga que la calidad de dicha solución sea superior a la actual. El conjunto de todas las posibilidades de reasignación de todas las granjas a todos los demás camiones conforma el vecindario entorno a la solución

actual.

El movimiento utilizado en el proceso de búsqueda *extra-rutas* mueve un elemento de la solución a diferentes rutas, permitiendo así que el largo de la ruta de los camiones varíe y que un productor sea visitado por otro camión sin que hayan productores que se dejen de visitar. Por esto, es importante que la generación de soluciones aleatoria considere desde un principio que todos los productores sean visitados.

Algorithm 3 Búsqueda Local Extra-Rutas

```
1: procedure EXTRA_LOCAL_SEARCH(solution)
2:   extra_local  $\leftarrow$  false
3:   while not extra_local do
4:     extra_local  $\leftarrow$  true
5:     neighborhood  $\leftarrow$  get_extra_neighborhood(solution)
6:     for neighbour in neighborhood in random order do
7:       neighbour_quality = evaluate(neighbour)
8:       if neighbour_quality > quality then
9:         solution  $\leftarrow$  neighbour
10:        quality  $\leftarrow$  neighbour_quality
11:        extra_local  $\leftarrow$  false
12:      break
13:    end if
14:  end for
15: end while
16: end procedure
```

El algoritmo 3 muestra el pseudocódigo del proceso de búsqueda local extra-rutas. El procedimiento recibe como parámetro la solución actual. En el primer paso se setea la variable *extra_local* como *false* y luego inicia el primer ciclo iterativo (línea 3 a la 15). Este se mantiene activo mientras la variable *extra_local* no sea verdadera. Dentro se setea la variable *extra_local* como *true*, haciendo que el propio proceso de búsqueda local la cambie a *false* para continuar dentro del ciclo. A continuación se genera el vecindario para la

búsqueda *extra-rutas* y se recorre con un ciclo *For* (línea 6 a la 14). En cada iteración se evalúa el vecino actual (*neighbour_quality*) y se compara su calidad con la solución actual (*quality*). En caso de que *neighbour_quality* sea mayor que *quality* se reemplaza la solución actual (*solution*) por el vecino actual (*neighbour*), la calidad actual (*quality*) por la calidad del vecino actual (*neighbour_quality*) y se setea como *false* la variable *extra_local*. Como el proceso de búsqueda local trabaja en modo primera mejora se ejecuta un *break* para terminar el recorrido en el vecindario actual y generar uno nuevo a partir de la nueva solución.

Intra-Rutas

En esta fase del algoritmo se busca el mejoramiento de la solución a través del reordenamiento de las granjas dentro de un camión, es decir, para cada ruta se realiza un reordenamiento de las visitas realizadas por dicho camión. El conjunto de todas las posibilidades de ordenamiento bajo el movimiento *2-OPT* conforma el vecindario entorno a la solución actual.

El movimiento utilizado en el proceso de búsqueda *intra-rutas* corresponde a *2-OPT* que dado dos elementos de una ruta invierte el orden en que se visitan los nodos (aplicando un orden contrario al actual). Este movimiento permite hacer un cambio sustancial en la ruta cuando los nodos seleccionados como pivotes están separados. Si bien, este movimiento conlleva un cambio estructural significativo en la distribución de granjas para una ruta determinada, no alteran demasiado los costos asociados a esta, ya que al interior de la ruta no habrá cambios significativos, solo los habrá en los extremos (suponiendo que se tiene un esquema de costos simétrico entre los distintos nodos).

El algoritmo 4 muestra el pseudocódigo del proceso de búsqueda local *intra-rutas*. El proceso recibe como parámetro la solución actual. En el primer paso se setea la variable *intra_local* como *false* y luego inicia el primero ciclo (línea 3 a la 14). Este se mantiene activo mientras la variable *intra_local* no sea verdadera. Apenas ingresa al ciclo se setea la variable *intra_local* como *true*, haciendo que el propio proceso de búsqueda local la cambie a *false* para continuar dentro del ciclo. A continuación se genera el vecindario para la búsqueda *intra-rutas* y se recorre con un ciclo *For* (línea 6 a la 13). En cada iteración se

evalúa el vecino actual (*neighbour_quality*) y se compara su calidad con la solución actual (*quality*). En este caso para la evaluación se utiliza la función *fast_evaluate* que permite encontrar la calidad de la solución a partir de los cambios producidos en una sola ruta, sin necesidad de recalcular los costos de todas las demás rutas. Esto es posible ya que el intercambio de nodos dentro de una misma ruta no altera las restricciones de capacidad de los camiones ni cuotas de la planta, solo plantea un cambio en el orden en que se recorren los nodos, haciendo que solo se produzcan cambios en los costos de dicha ruta. En caso de que la *neighbour_quality* sea mayor que *quality* se reemplaza la solución actual (*solution*) por el vecino actual (*neighbour*), la calidad actual (*quality*) por la calidad del vecino actual (*neighbour_quality*) y se setea como *false* la variable *intra_local*. Como el proceso de búsqueda local considera la versión primera mejora, adicionalmente se ejecuta un *break* para terminar el recorrido en el vecindario actual y generar uno nuevo a partir de la nueva solución.

Algorithm 4 Búsqueda Local Intra-Rutas

```

1: procedure INTRA_LOCAL_SEARCH(solution)
2:   intra_local  $\leftarrow$  false
3:   while not intra_local do
4:     intra_local  $\leftarrow$  true
5:     neighborhood  $\leftarrow$  get_intra_neighborhood(solution)
6:     for neighbour in neighborhood in random order do
7:       neighbour_quality = fast_evaluate(neighbour)
8:       if neighbour_quality > quality then
9:         solution  $\leftarrow$  neighbour
10:        quality  $\leftarrow$  neighbour_quality
11:        intra_local  $\leftarrow$  false
12:      break
13:    end if
14:  end for
15: end while
16: end procedure

```

3.5. Iterated Local Search

Una modificación al esquema *hill-climbing* considera cambiar la generación de una solución aleatoria factible en cada ciclo superior por la perturbación de la solución, esto es, cambiar un porcentaje de los nodos de la solución en forma aleatoria y sin tener consideraciones respecto al empeoramiento de la calidad o cumplimiento de las restricciones. Este esquema se llama *iterated local search* y consiste en un proceso de búsqueda local utilizada en investigación de operaciones, que a partir de una solución aleatoria, realiza sucesivos movimientos con la esperanza de que la nueva solución sea mejor que la inicial. Este proceso se repite iterativamente hasta que ningún vecino (generado mediante un movimiento a la solución actual) tenga una calidad mejor que la solución actual, en cuyo caso, se dice que se ha encontrado un óptimo local. El proceso continua, realizando una perturbación a la solución actual que permita escapar del óptimo local y encontrar zonas en las que la aplicación de nuevos movimientos llegue a encontrar otro óptimo local, (que ojala tenga una calidad mejor que la de éste último). En el escrito *Iterated Local Search* [29] publicado por HR Lourenço, OC Martin, T Stützle en 2003 explican a profundidad todas las características y alcances de ésta metaheurística.

La estructura general del acercamiento propuesto se muestra en el algoritmo 5. El algoritmo ILS recibe los mismos parámetros que en *hill-climbing*, pero adicionalmente requiere de un parámetro adicional *fp* conocido como factor de perturbación. En este acercamiento se considera que el factor de perturbación puede tomar valores entre 0 y 1. El detalle de este parámetro se explica en la sección 3.5.1.

Se inicia con la definición de las variables booleanas *global_local*, *extra_local* y *intra_local* que controlarán la repetición de los procesos de búsqueda. Luego se inicializa la variable *best_solution* con la función *random_feasible_solution()* que genera una solución aleatoria factible en la mayoría de los casos. Luego esta solución se evalúa y comienza el ciclo restart (línea 5 del algoritmo 5). Dentro se setea el valor de *global_local* a *false*. Esta variable indica si se esta en un óptimo local. A continuación comienza el segundo ciclo iterativo (línea 7 a la 11) en donde, se realizan los ciclos de búsqueda local *extra-rutas* e *intra-rutas* que son

exactamente los mismo utilizados en *HC* y se explicaron en la sección 3.4. Cuando las variables *extra_local* e *intra_local* adquieren sus valores de los procesos de búsquedas locales se setea el valor de la variable *global_local*. Al terminar este ciclo, se comprueba si el óptimo local posee una calidad superior que la mejor solución encontrada hasta el momento (línea 12 a la 15. Finalmente, en la línea 16 se encuentra la gran diferencia con respecto a *HC*, en vez de genera una nueva solución aleatoria, se realiza la perturbación de la solución actual. Esto permite, según el valor del parámetro de perturbación, escapar del óptimo local actual y poder en el próximo ciclo iterativo encontrar soluciones distintas a la actual.

Algorithm 5 *iterated local search*

```

1: procedure ITERATED_LOCAL_SEARCH(max_quality, max_time, f_cuotas, f_capacidad, fp)
2:   global_local, extra_local, intra_local
3:   best_solution  $\leftarrow$  random_feasible_solution()
4:   best_quality  $\leftarrow$  evaluate(solution)
5:   while best_solution < max_quality and current_time < max_time do
6:     global_local  $\leftarrow$  false
7:     while not global_local do
8:       extra_local  $\leftarrow$  extra_local_search()
9:       intra_local  $\leftarrow$  intra_local_search()
10:      global_local  $\leftarrow$  extra_local or intra_local
11:    end while
12:    if quality > best_quality then
13:      best_solution  $\leftarrow$  current_solution
14:      best_quality  $\leftarrow$  quality
15:    end if
16:    solution  $\leftarrow$  perturbation(solution, fp)
17:    quality  $\leftarrow$  evaluate(solution)
18:  end while
19:  return best_solution
20: end procedure

```

Al finalizar el algoritmo, ya sea, por que se llegó a la calidad máxima o al tiempo máximo

de ejecución, se retorna la mejor solución encontrada en todo el proceso (línea 19).

3.5.1. Perturbación

El proceso de perturbación en un ILS tiene como objetivo sacar a la solución del óptimo local en el que se encuentra atrapado. A diferencia de un *hill-climbing*, que utiliza una estrategia de reinicio, ILS lo hace a través de la aplicación de modificaciones aleatorias a una solución conocida como perturbaciones. En esta propuesta, dichas modificaciones se traducen en la aplicación de fp movimientos. Cada uno de estos movimientos consiste en la selección de un nodo aleatorio de cualquier ruta y su incorporación a cualquier otra posición, ya sea de la misma ruta o de una distinta. El factor fp indica el porcentaje de nodos que serán movidos aleatoriamente a otras posiciones y/o rutas durante el proceso de perturbación de las soluciones.

Algorithm 6 *perturbation*

```
1: procedure PERTURBATION(solution, fp)
2:    $deep\_disturbing \leftarrow farms\_length * fp$ 
3:   for  $deep\_disturbing$  as  $deep$  do
4:      $route1 \leftarrow$  random route from  $solution$ 
5:      $route2 \leftarrow$  random route from  $solution$ 
6:      $node1 \leftarrow$  random route from  $route1$ 
7:      $node2 \leftarrow$  random route from  $route2$ 
8:      $solution \leftarrow swap(solution, route1, node1, route2, node2)$ 
9:   end for
10:  return  $solution$ 
11: end procedure
```

La estructura general de la función de perturbación se muestra en el algoritmo 6. Primero se calcula la cantidad de nodos que serán intercambiados (línea 2). Luego se entra en un ciclo *For* tantas iteraciones como par de nodos haya que intercambiar. Dentro, se eligen aleatoriamente 2 rutas, y para cada una se seleccionan dos nodos al azar. Luego se hace el

intercambio mediante la función *swap* que recibe como parámetro el arreglo de solución, las rutas y los nodos de cada una. Al terminar el ciclo *For* (línea 9) se retorna la solución modificada (línea 10) .

Este movimiento es parecido al utilizado en el proceso de búsqueda *extra-rutas*. La diferencia radica en que en dicho proceso de búsqueda no se permite al nodo moverse dentro de la misma ruta, acá si es posible obtener este resultado. Además, en el proceso de búsqueda la aplicación del movimiento viene acompañada de una posterior revisión de la calidad de la solución resultante. En el proceso de perturbación del ILS, el movimiento se aplica sin importar la calidad resultante. Es exactamente esta cualidad la que permite la diversificación a espacios cercanos de la solución actual.

Cabe destacar la importancia del valor del factor de perturbación fp , si este es muy pequeño (cercano al 0), no habrán muchos cambios en la solución después de la perturbación, haciendo que en pocos movimientos el algoritmo pueda llevarla a su estado anterior y esta se quede estancada en el mismo máximo local. Por otro lado, si el parámetro de perturbación es muy grande (cercano a 1), la cantidad de nodos a cambiar será muy alta, haciendo que la solución resultante se aleje demasiado de la solución actual. Esto produciría resultados aún más desfavorables que al generar nuevas soluciones aleatorias factibles, ya que acá, se intenta mantener la factibilidad de las soluciones, en cambio, el proceso de perturbación del ILS realiza movimientos sin importar la factibilidad, haciendo que la solución, además de ser muy distinta, pueda pasar a ser tan infactible que los procesos de búsquedas locales no puedan devolverla a la zona factible.

3.6. Resumen

Para resolver el problema de recolección de leche se proponen dos algoritmos basados en búsqueda local: *hill-climbing* e *iterated local search*. Ambos parten de una solución generada de manera aleatoria, que luego mediante un búsqueda *extra-rutas* e *intra-rutas* buscan en el vecindario cercano encontrar una solución mejor que la actual. La función generadora de soluciones aleatorias, en su gran medida genera soluciones factibles, en donde se prioriza

agrupar productores que produzcan un mismo tipo de leche en los mismos camiones, y si es posible mantiene la capacidad de los camiones sin excesos. La búsqueda *extra-rutas* genera un vecindario compuesto al tomar cada uno de los nodos de todas las rutas e intentar moverlo en la mejor posición de cada una de las demás rutas. Por su parte, la búsqueda *intra-rutas* genera un vecindario compuesto al mover cada uno de los nodos de todas las rutas, dentro de la misma ruta. Además, durante todo el proceso se guarda la mejor solución encontrada hasta el momento. Esta será la variable que se retorne una vez terminada la ejecución del algoritmo. El cambio más importante entre *hill-climbing* e *iterated local search* radica en como se reinicia el proceso de búsqueda, en *hill-climbing* cuando no se encuentra una solución mejor que la actual en el vecindario de búsqueda, se genera una nueva solución aleatoria, mientras que en *iterated local search* se realiza una perturbación a la solución actual, esto es, cambiar un cierto porcentaje de los nodos en forma aleatoria sin tener en consideración costos, beneficios y restricciones.

La representación utilizada para guardar una solución es un vector de vectores de enteros, es decir, corresponde a una lista de rutas, donde en cada una, se guarda un listado de los nodos que son visitados en orden. Para evaluar soluciones se utiliza una función que suma el beneficio obtenido por cada litro de leche que fue entregado a la planta según sus requerimientos y resta el costo de las rutas realizadas por cada camión. Adicionalmente se realiza una resta por cada litro de leche que sobrepase la capacidad de los camiones o cada litro de leche que falte para cumplir las cuotas de la planta, ambos multiplicados por un factor.

Capítulo 4

Implementación

En este capítulo se presentan los experimentos realizados y los resultados obtenidos durante el proceso de evaluación de los algoritmos propuestos. Se especifican los objetivos del proceso de experimentación, se explican las instancias de prueba utilizadas, las características generales del proceso de experimentación y los equipos en que se ejecutaron los experimentos. Luego se exponen los resultados obtenidos y se comparan los dos enfoques propuestos: *hill-climbing* e *iterated local search*. Por último se presenta una comparación con los resultados de la literatura y se realiza un análisis de convergencia de las instancias más complicadas de resolver.

4.1. Objetivos

Los objetivos del proceso de experimentación de este trabajo de memoria se pueden resumir en los siguientes tres puntos:

- Evaluar la relevancia de los principales componentes de cada uno de los algoritmos propuestos. En el caso de *hill-climbing* se evalúa la relevancia de los valores de las penalizaciones y la importancia de la naturaleza de los procesos de búsqueda *intra-rutas* y *extra-rutas*. Para *iterated local search* se evalúa la importancia del valor del

parámetro de perturbación y la relevancia de la selección de la solución sometida a perturbación.

- Comparar los resultados obtenidos con los resultados del estado del arte.
- Analizar la convergencia de las instancias difíciles de resolver.

4.2. Instancias de Prueba

Para las pruebas de los algoritmos propuestos se consideran dos conjuntos de prueba. El primer conjunto de pruebas considera 37 instancias bien conocidas del problema de ruteo de vehículos. Todas estas instancias consideran tres camiones y tres tipos de leche. Cada tipo de leche tiene un valor asociado a su calidad, el cual, se utiliza para obtener el beneficio de la leche recolectada. Para la leche de tipo A este valor corresponde a 1,0 por litro, para la leche de tipo B es de 0,7 por litro y para la leche de tipo C es de 0,3 por litro. Este conjunto de instancias se denomina instancias pequeñas. El segundo conjunto de instancias fue generado a partir de la información de ubicación geográfica y niveles de producción de 500 granjas en el sur de Chile. Este conjunto de instancias se denominará instancias reales.

4.2.1. Instancias Pequeñas

El conjunto de instancias pequeñas está compuesto de 37 instancias usadas por Paredes-Belmar. [36]. La tabla 4.1 muestra las características de las instancias desde la instancia *a33* a la instancia *a80* que fueron obtenidas del trabajo de Augerat [2]. Los tamaños de estas instancias varían entre 33 y 80 nodos. La tabla 4.2 muestra las características de las instancias desde la instancia *tai75A* a la instancia *tai75D*, más las instancias *c50* y *c75*, las cuales fueron obtenidas del trabajo de Taillard [50]. Además se muestran las instancias *f45*, *f71* y *f72* que fueron obtenidas del trabajo de Fisher [19]. Los tamaños de las instancias de esta tabla varían entre 50 y 75 nodos. La tabla 4.3 muestra las características de las instancias desde la instancia *eil22* a la instancia *eil76*, y la *att48*, las cuales fueron obtenidas de la TSPLib de Reinelt [44]. Los tamaños de estas instancias varían entre 22 y 76 nodos. Para cada instancia

de estas tablas se indica su nombre, cantidad de nodos, demanda total y producción total para cada tipo de leche; y la capacidad de los tres camiones. En cada caso se considera una planta y el resto corresponde a productores. En [36] se les agregó tipos de leche a cada una de los productores, asignando para el primero, leche de tipo A, para el segundo, leche de tipo B, para el tercero, leche de tipo C, para el cuarto, leche de tipo A, y así sucesivamente hasta asignar un tipo de leche al último productor.

Para el conjunto de instancias de la tabla 4.1, en la mayoría de los casos, cualquier camión por si solo puede recolectar toda la leche de un mismo tipo, dada su amplia capacidad. En los pocos casos en que esto no ocurre, es posible asignar determinados camiones a recolectar ciertos tipos de leche. En muy pocos casos es estrictamente necesario mezclar leche de distintos tipos debido a que ningún camión por si solo puede recolectar toda la leche de un mismo tipo.

El conjunto de instancias de la tabla 4.2 presentan en su mayoría demandas superiores a la producción de un tipo de leche, esto hace necesario realizar mezcla para poder cumplir las cuotas de la planta. Esto hace que este conjunto de instancias sea más difícil de resolver. En cuanto a la capacidad de los camiones, en su mayoría sobra capacidad en relación a la producción, pero en algunos casos esta diferencia es algo baja.

El conjunto de instancias de la tabla 4.3 presenta demandas bajas en relación a la producción, mientras que la capacidad de los camiones es bastante amplia en relación a la producción. Estas dos observaciones hacen que este conjunto de instancias sea relativamente más fácil de resolver.

4.2.2. Instancias reales

Este conjunto de instancias fue construido en base a la información de 500 productores distribuidos en aproximadamente 9.600 km^2 en una región del sur de Chile. En la figura 4.1 se muestra la distribución de los productores y la planta procesadora. Los círculos, triángulos y cuadrados corresponden a los productores diferenciados por tipo de leche, mientras que el diamante negro ubicado en la zona superior izquierda corresponde a la planta.

Instancia	Nodos	Demanda			Producción			Capacidad		
a33	33	15	8	6	17,6	11,4	11,4	15	20	25
a34	34	10	12	14	15,9	16,2	16,2	20	20	25
a36	36	10	12	14	16,0	14,9	14,9	20	20	25
a37	37	10	8	6	13,5	12,7	12,7	20	15	10
a38	38	10	15	15	14,3	19,2	19,2	20	20	10
a39	39	10	12	14	14,6	17,3	17,3	20	20	20
a44	44	20	16	12	24,4	17,0	17,0	25	20	15
a45	45	20	18	18	24,3	18,0	18,0	25	20	20
a46	46	16	17	18	22,4	19,9	19,9	30	25	20
a48	48	20	20	20	18,8	17,9	17,9	30	25	20
a53	53	20	20	20	23,4	24,7	24,7	30	30	30
a54	54	5	5	5	12,6	11,6	11,6	15	15	15
a55	55	5	10	15	11,9	12,2	12,2	15	15	20
a60	60	8	12	16	10,9	12,6	12,6	20	10	20
a61	61	30	20	10	30,4	34,6	34,6	35	35	35
a62	62	10	11	12	12,6	10,8	10,8	15	15	15
a63	63	5	10	20	10,0	17,2	17,2	20	20	20
a64	64	5	10	20	11,7	16,0	16,0	20	20	20
a65	65	10	12	14	14,3	15,3	15,3	15	15	15
a69	69	10	15	15	11,7	15,7	15,7	20	20	20
a80	80	16	10	16	16,2	14,6	14,6	20	20	20

Tabla 4.1: Instancias de Augerat [2]

La producción diaria de leche por parte de los productores varía entre los 57 y los 25.000 litros, tal como se puede ver en el histograma de la figura 4.2. De los 500 productores, 313 producen leche de tipo A, 159 producen leche de calidad B y 28 producen leche de calidad C. Esto corresponde a una producción total de 1.430.715 litros de leche de tipo A, 268.564 de leche de tipo B y 74.475 de leche de tipo C. Las demandas de la planta son: para la leche de tipo A 1.250.000 litros, para la leche de tipo B 300.000 litros y para la leche de tipo C 100.000 litros. Cada tipo de leche tiene un valor asociado a su calidad, el cual, se utiliza para obtener el beneficio de la leche recolectada. Para la leche de tipo A este valor corresponde a 0,015 por cada litro, para la leche de tipo B es de 0,0105 por cada litro y para la leche de tipo C es de 0,0045 por cada litro. Esto significa que si se recolectan y entregan 1.000 litros de leche de tipo A en la planta, se obtiene 15 unidades monetarias un beneficio. A este valor

Instancia	Nodos	Demanda			Producción			Capacidad		
c50	51	25	20	15	33,1	34,5	34,5	35	35	35
c75	76	10	25	25	23,4	23,3	23,3	30	30	30
tai75A	76	10	15	20	25,1	11,8	11,8	25	30	35
tai75B	76	20	25	25	27,1	23,9	23,9	30	30	30
tai75C	76	5	10	15	14,1	8,9	8,9	15	20	25
tai75D	76	15	15	20	20,9	19,8	19,8	20	25	35
f45	45	5	10	5	9,85	17,9	17,9	20	15	10
f71	72	25	20	10	26,8	54,6	54,6	50	50	50
f72	72	20	30	40	26,8	54,6	54,6	60	60	60

Tabla 4.2: Instancias de Taillard [50] y Fisher [19]

Instancia	Nodos	Demanda			Producción			Capacidad		
eil22	22	6,0	5,0	4,0	9,8	7,2	7,2	10,0	15,0	20,0
eil23	23	1,0	1,5	2,0	6,1	2,0	2,0	6,0	7,0	8,0
eil30	30	2,2	2,4	2,6	2,4	6,8	6,8	5,0	5,5	6,0
eil33	33	8,0	7,0	6,0	11,2	11,6	11,6	20,0	20,0	15,0
eil51	51	22,0	23,0	24,0	22,4	29,6	29,6	25,0	30,0	35,0
eil76	76	45,0	43,0	40,0	46,8	46,7	46,7	45,0	48,0	51,0
att48	48	38,0	34,0	30,0	40,0	40,0	40,0	50,0	45,0	40,0

Tabla 4.3: Instancias de Reinelt [44]

se le restan los costos de transporte de las rutas y se obtiene la ganancia total de este proceso.

En la tabla 4.4 se muestran las características de las instancias generadas a partir de la información de la ubicación y producción de las 500 granjas. Todas estas instancias consideran tres tipos de leche. Para cada una de las instancias de la tabla se indica su nombre, número de nodos, cantidad de camiones y su capacidad total. En [36] Paredes-Belmar asignó, de manera arbitraria, 100 camiones para recolección a esta instancia. Esta instancia será denominada *real500-100*. En varias pruebas realizadas, se comprobó que las soluciones encontradas por el algoritmo no incluían a todos los camiones, es decir, dejaban camiones sin asignación de granjas. Esto se debe a que, como hay capacidad de sobra en los camiones, el algoritmo prefiere aprovechar los camiones en ruta para asignarles nuevos productores, en vez de utilizar nuevos camiones. Dado este comportamiento se decidió limitar la cantidad de camiones, manteniendo capacidades, cuotas y valores de cada leche.

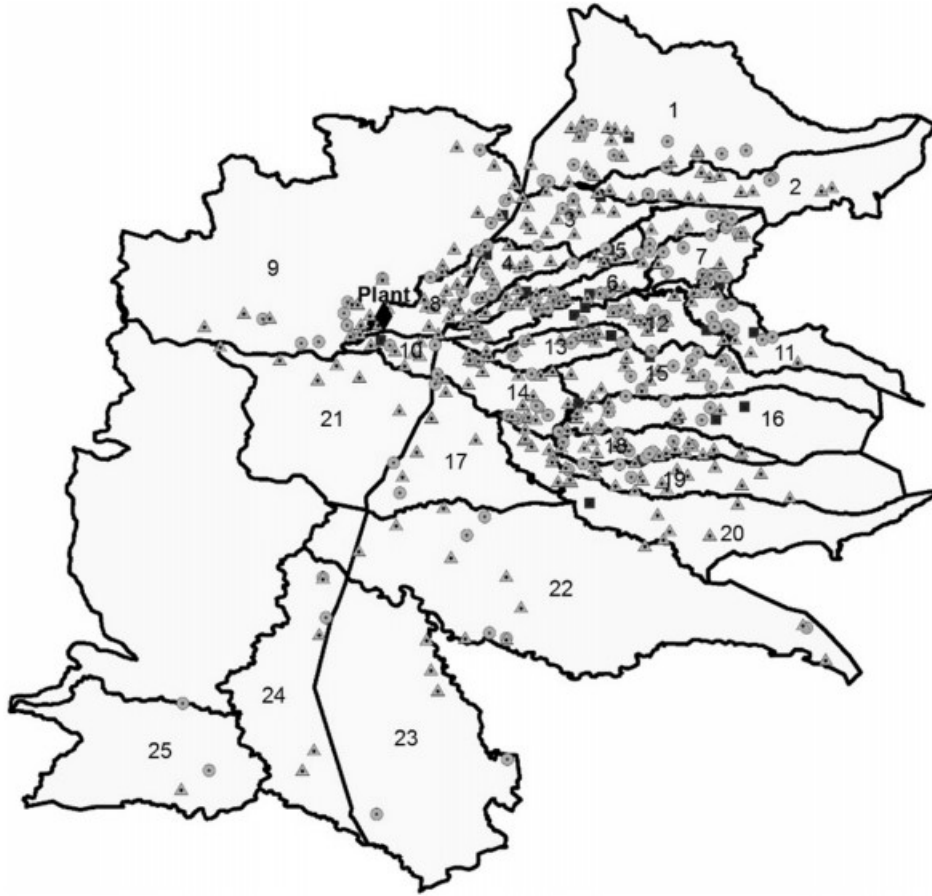


Figura 4.1: Región del sur de Chile con la distribución de todos los productores de la instancia real. Los círculos, triángulos y cuadrados corresponden a los productores diferenciados por tipo de leche, mientras que el diamante corresponde a la planta.

En [36] fue posible encontrar soluciones que utilizaban solo 72 camiones, por lo tanto se generaron instancias con 88, 77 y 72 camiones. Para generar las instancias *real500-88*, *real500-77* y *real500-72* se eliminaron de la instancia de Paredes-Belmar. [36] una cierta cantidad de camiones, comenzando por los de menor capacidad. Además al hacer el cambio en la cantidad de camiones, no se está variando el problema en sí, ya que el algoritmo también podría llegar a ocupar una cantidad inferior de camiones a partir de los 100 asignados inicialmente.

Cabe destacar que para esta instancia no se dan las coordenadas de cada productor, sino

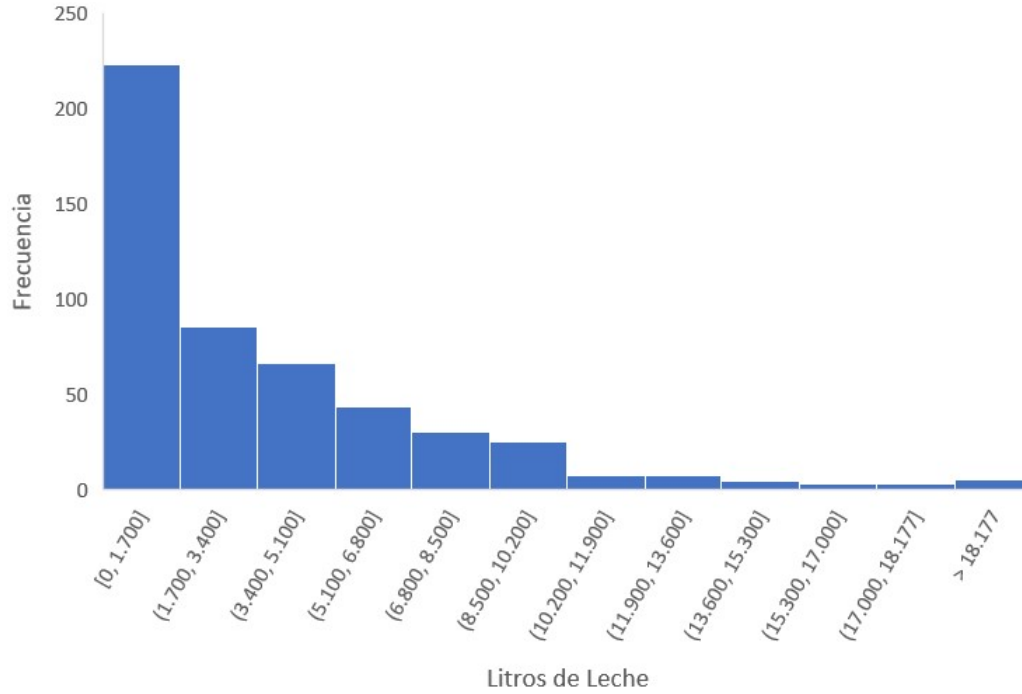


Figura 4.2: Histograma de producción de leche de los productores en litros.

que, se entrega directamente la matriz de distancias entre todos los nodos (planta y productores). Esta matriz es asimétrica, por lo que, en general, la distancia de ir de un nodo cualquiera a otro no es la misma que realizar el mismo recorrido de vuelta. Esto se debe a que la información proviene de un caso real, por lo tanto, considera viajes en un terreno que tiene caminos que pasan por autopistas, caminos de un solo sentido, pendientes en un sentido o peajes que hacen variar los costos en las rutas entre pares de nodos.

Instancia	Nodos	Capacidad	Camiones
real500-100	500	2.500.000	100
real500-88	500	2.320.000	88
real500-77	500	2.115.000	77
real500-72	500	2.015.000	72

Tabla 4.4: Instancia de Paredes-Belmar. [36] con 100 camiones, más variantes que consideran 88, 77 y 72 camiones.

4.2.3. Entorno de Experimentación

Considerando la naturaleza estocástica de los algoritmos propuestos, todas las pruebas presentadas en este trabajo de memoria consideran la ejecución de los respectivos algoritmos utilizando un conjunto de 30 semillas aleatorias diferentes. Al ejecutar cada versión del algoritmo con estas 30 semillas aleatorias distintas se elimina cualquier sesgo en los resultados producto de las características de una semilla en particular.

Las pruebas se realizaron en un equipo con procesador Intel Core i5-4690K 3.50GHz, con 16GB de memoria RAM a 2.400GHz y un disco de estado sólido. El equipo cuenta con Windows 10 Education, pero las pruebas fueron realizadas en una máquina virtual 2 núcleos de procesador asignados, 4GB de RAM y sistema operativo Ubuntu 18.04.3 LTS.

4.3. Resultados

A continuación se presentan los resultados obtenidos en el proceso de evaluación de los algoritmos. Primero se presentan los resultados orientados a analizar la relevancia de los componentes de los algoritmos implementados. Para el algoritmo *hill-climbing* se analiza el efecto de los pesos de las penalizaciones y la relevancia de la naturaleza de los procesos de búsqueda *intra-rutas* y *extra-rutas*. Para *iterated local search* se evalúa el valor del parámetro de perturbación y la selección de la solución sometida a perturbación en cada paso. Luego se realiza un análisis de convergencia de las instancias con problemas y en las cuales no fue posible obtener el óptimo global publicado por Paredes-Belmar. Finalmente, se presentan los resultados obtenidos al ejecutar los algoritmos con las instancias reales.

4.3.1. Hill Climbing

Parámetros de penalización

El algoritmo implementado trabaja sobre el espacio de soluciones infactibles, es decir, es posible que la solución inicial generada en forma aleatoria genere soluciones que consideren camiones con exceso de capacidad o cuotas de leche no satisfechas en la planta.

La función de evaluación considera la penalización de cada una de estas insatisfacciones. El incumplimiento de cada restricción fue penalizada en una cantidad proporcional a los litros de leche (exceso o faltantes según corresponda) multiplicados por un factor de penalización distinto como se muestra en la ecuación 4.1.

$$Quality = milk_profit - route_cost - litros_cuota * f_cuotas - litros_exceso * f_capacidad \quad (4.1)$$

Las tablas 4.5, 4.6 y 4.7, muestran los resultados obtenidos al ejecutar el algoritmo *hill-climbing* propuesto en el conjunto de instancias pequeñas, considerando cuatro configuraciones diferentes de valores de pesos en la función de evaluación. Para las pruebas se consideraron los valores 10 y 100 de manera tal de asignar un peso relevante a las penalizaciones comparadas con la ganancia de la leche y los costos de las rutas. Además se considera importante el asignar un orden de magnitud de diferencia entre los parámetros de penalizaciones de forma tal de analizar si alguna de las restricciones es mucho más importante de satisfacer que la otra. En la parte izquierda de cada tabla, la primera columna (*I*) corresponde al nombre de la instancia, luego se muestra la calidad del óptimo global obtenido por Paredes-Belmar y el tiempo en segundos en que este fue encontrado para dicha instancia. En la parte derecha de la tabla se encuentran las versiones del algoritmo que se ejecutaron: *10 - 10*, *10 - 100*, *100 - 10* y *100 - 100*, donde el primer número corresponde a valor del parámetro de penalización por incumplir capacidad de los camiones, mientras que el segundo número corresponde al parámetro de penalización por incumplimiento de cuotas de la planta. Cada versión fue ejecutada 30 veces, cada una con una semilla aleatoria distinta. En cada ejecución se puede ver el GAP y tiempo (en segundos) obtenido para una instancia en particular. El GAP corresponde al error relativo entre la calidad del óptimo global y la encontrada en una prueba determinada. Cuando este es positivo, significa que ha logrado superar el valor del óptimo

global publicado por Paredes-Belmar, cuando es cero, significa que se ha obtenido el óptimo global, y cuando es negativo, significa que se ha superado el valor del óptimo global publicado por G. Paredes-Belmar. Este último caso, solo se da para la instancia *tai75C*, en la que, se obtiene un punto de calidad mayor que el óptimo global publicado por Paredes-Belmar. Este valor se encuentra en todas las pruebas realizadas posteriormente, y se asocia más a un error de aproximación que a una diferencia en el óptimo.

En la tabla 4.5, se puede apreciar que el GAP es cero (denotado por " – " en las tablas) en la mayoría de las instancias. Los tiempos denotados por " – ", indican que el algoritmo encontró el óptimo global en menos de 1 segundo de ejecución. En la instancia *a38* y *a60* no se encontró el óptimo global en los tiempos utilizados por Paredes-Belmar, pero se ha llegado a un valor cercano. Esto se concluye con el GAP tan pequeño obtenido en ambos casos, sobre todo en la instancia *a38*. La configuración de parámetros que dio mejores resultados para este conjunto de instancias fue *100 - 10*. La instancia que determino esto fue la *a64*, ya que esta fue la única que presentó resultados distintos en tiempos de ejecución. En cuanto a los GAPs todas las instancias obtuvieron los mismos resultados.

En la tabla 4.6, se encuentran la mayoría de las instancias con problemas. Es por esto, que se ocupa todo el tiempo de ejecución y se obtienen valores muy inferiores a los óptimos publicados por Paredes-Belmar. Además, los GAPs también tienen valores altos. En las otras instancias, se aprecia una leve mejora en los tiempos cuando el segundo parámetro tiene como valor 10, quedándose como mejor configuración la *100-10*. En cuanto a los GAPs, solo se aprecian diferencias en la instancia *f71*, donde se aprecia una diferencia de una centésima entre las pruebas *10-X* y *100-X*.

En la tabla 4.7, se encuentra el óptimo global de todas las instancias, en la mayoría en tiempos inferiores a 1 segundo, excepto en la instancia *eil76*, cuyos tiempos de ejecución es de 1 segundo para las pruebas *X-10* y de 2 segundos para las pruebas *X-100*.

Al comparar los resultados de todas las tablas en conjunto, se aprecian pocas diferencias en los tiempos de ejecución y en los GAPs. En las instancias *a64*, *f71* y *f72* se aprecian diferencias en los tiempos mas considerables. Respecto a los GAPs, no se aprecian diferencias, ni siquiera en las instancias con problemas. Esto quiere decir que en todas las pruebas se

I	Paredes-Belmar		10 - 10		10 - 100		100 - 10		100 - 100	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
a33	29.417	62	-	-	-	-	-	-	-	-
a34	30.496	40	-	-	-	-	-	-	-	-
a36	29.233	110	-	-	-	-	-	-	-	-
a37	24.837	45	-	-	-	-	-	-	-	-
a38	28.596	570	0,045	570	0,045	570	0,045	570	0,045	570
a39	30.808	110	-	-	-	-	-	-	-	-
a44	38.771	101	-	-	-	-	-	-	-	-
a45	40.282	136	-	-	-	-	-	-	-	-
a46	40.696	66	-	-	-	-	-	-	-	-
a48	39.800	230	-	-	-	-	-	-	-	-
a53	46.662	183	-	-	-	-	-	-	-	-
a54	22.414	304	-	-	-	-	-	-	-	-
a55	24.694	270	-	-	-	-	-	-	-	-
a60	25.041	3.565	1,813	3.565	1,813	3.565	1,813	3.565	1,813	3.565
a61	60.644	561	-	-	-	-	-	-	-	-
a62	22.917	1.022	-	-	-	-	-	-	-	-
a63	24.447	2.930	-	1	-	1	-	1	-	1
a64	24.100	5.395	-	186	-	195	-	172	-	222
a65	28.046	478	-	-	-	-	-	-	-	-
a69	25.822	1.552	-	1	-	1	-	1	-	1
a80	29.977	5.626	-	-	-	-	-	-	-	-

Tabla 4.5: Evaluación de parámetros de penalización con *hill-climbing*.

llega a los mismos resultados, lo que cambia es el tiempo en que se estos se obtienen. En la tabla 4.8 se muestra la sumatoria de los tiempos de ejecución y de los GAPs para este escenario de pruebas. Se aprecia que la diferencia en los GAPs es mínima, pero la diferencia en los tiempos presenta diferencias considerables, sobre todo entre la prueba *10 - 10* que fue la que otorgó menores tiempos de ejecución y la prueba *100 - 100* que presentó los mayores tiempos de ejecución. En resumen, la selección de parámetros de penalización que otorga mejores resultados es la que asigna para ambos parámetros el valor de 10, presentando un tiempo total de ejecución de 29.749 segundos, con un GAP acumulado de 19,984.

Relevancia de la naturaleza de los procesos de búsqueda

La estructura del algoritmo *hill-climbing* propuesto considera el uso de dos procesos de búsqueda local: los procesos de búsqueda *intra-rutas* y *extra-rutas*. El proceso de búsqueda *intra-rutas* está enfocado en mejorar la calidad de las rutas en términos de costos sin cambiar

I	Paredes-Belmar		10 - 10		10 - 100		100 - 10		100 - 100	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
c50	66.160	84	1,032	84	1,032	84	1,032	84	1,032	84
c75	46.180	13.760	8,155	13.760	8,155	13.760	8,155	13.760	8,155	13.760
tai75A	42.992	4.806	5,785	4.806	5,785	4.806	5,785	4.806	5,785	4.806
tai75B	48.238	15.056	-	1.239	-	1.374	-	1.149	-	1.787
tai75C	25.906	4.537	-0,004	13	-0,004	16	-0,004	13	-0,004	16
tai75D	43.869	2.645	3,157	2.645	3,157	2.645	3,157	2.645	3,157	2.645
f45	23.705	80	-	-	-	-	-	-	-	-
f71	72.864	3.483	-	2.665	-	2.990	0,001	3.483	0,001	3.483
f72	72.072	3.659	-	213	-	249	-	190	-	270

Tabla 4.6: Evaluación de parámetros de penalización con *hill-climbing*.

I	G. Paredes-Belmar		10 - 10		10 - 100		100 - 10		100 - 100	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
eil22	15.947	12	-	-	-	-	-	-	-	-
eil23	7.207	6	-	-	-	-	-	-	-	-
eil30	7.117	99	-	-	-	-	-	-	-	-
eil33	20.409	58	-	-	-	-	-	-	-	-
eil51	50.128	154	-	-	-	-	-	-	-	-
eil76	91.461	1.700	-	1	-	2	-	1	-	2
att48	17.452	284	-	-	-	-	-	-	-	-

Tabla 4.7: Evaluación de parámetros de penalización con *hill-climbing*.

la cantidad ni tipo de leche transportada por la ruta. El proceso de búsqueda *extra-rutas* está enfocado en intercambiar nodos entre rutas diferentes por lo cual puede cambiar la calidad de las rutas en términos de costos, cantidad y tipo de leche transportado por las rutas involucradas. Ambos procesos generan un vecindario de búsqueda que recorren para encontrar soluciones de mejor calidad. En este proceso existen dos enfoques: *mejor-mejora* que busca en todo el vecindario y elige en él la solución de mejor calidad y *primera-mejora* que busca en el vecindario, quedándose con la primera solución cuya calidad sea superior a la

Prueba	Tiempo	GAP
10 - 10	29.749	19,984
10 - 100	30.258	19,984
100 - 10	30.440	19,985
100 - 100	31.212	19,985

Tabla 4.8: Resumen con un compilado de los resultados de las pruebas realizadas con *hill-climbing* para determinar el mejor valor de los parámetros de penalización.

de la solución actual. Para realizar estas pruebas estableció el valor de 10 en cada parámetro de penalización, debido a que esta configuración fue el que presentó mejores resultados en el escenario de pruebas anterior.

Las tablas 4.9, 4.10 y 4.11 muestran los resultados obtenidos al ejecutar el algoritmo *hill-climbing* propuesto en el conjunto de instancias pequeñas, considerando cuatro configuraciones diferentes para los enfoques dentro del vecindario de búsqueda. Para las pruebas se consideran los enfoques de *primera-mejora* y *mejor-mejora* de tal manera de ver como la combinación de estos enfoques en los procesos de búsqueda *intra-rutas* y *extra-rutas* afecta a la calidad y convergencia de las soluciones obtenidas. En la parte izquierda de cada tabla, la primera columna (*I*) corresponde al nombre de la instancia, luego se muestra la calidad del óptimo global obtenido por Paredes-Belmar y el tiempo en que este fue encontrado para dicha instancia. En la parte derecha de la tabla se encuentran las versiones del algoritmo que se ejecutaron: *PM - PM*, *PM - MM*, *MM - PM* y *MM - MM*, donde *PM* significa *primera-mejora* y *MM* significa *mejor-mejora*. La primera parte de la nomenclatura corresponde al enfoque utilizado en el proceso de búsqueda *extra-rutas*, mientras que la segunda parte de la nomenclatura corresponde al enfoque utilizado en el proceso de búsqueda *intra-rutas*. Cada versión fue ejecutada 30 veces, cada una con una semilla aleatoria distinta. En cada ejecución se puede ver el GAP y tiempo (en segundos) obtenido para una instancia en particular.

En la tabla 4.9 se aprecia que en la mayoría de las instancias se encuentra el óptimo global en tiempos inferiores a 1 segundo. Las excepciones nuevamente son la instancia *a38* con un GAP muy bajo que va de 0,045 a 0,049 y la instancia *a60* con un GAP un poco más alto de 1,813 en todas las pruebas. Los tiempos en que se encontró el óptimo global en la mayoría de los casos fue inferior a 1 segundo. En la instancia *a64* se presentan tiempos mayores que van de los 186 a los 1.022 segundos.

En la tabla 4.10 se encuentran la mayoría de las instancias con problemas, las cuales se distinguen por que utilizan todo el tiempo de ejecución disponible y presentan GAPs muy altos, llegando hasta 8,115 en el caso de la instancia *c75*. Los tiempos de ejecución permanecen muy cercanos para las distintas pruebas, excepto en la instancia *f71*, donde se obtienen tiempos de ejecución muy distintos, específicamente en la prueba *MM - PM* se obtienen un tiempo de 406 segundos, mientras que en las demás se obtienen tiempos cercanos

I	G. Paredes-Belmar		PM - PM		PM - MM		MM - PM		MM - MM	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
a33	29.417	62	-	-	-	-	-	-	-	-
a34	30.496	40	-	-	-	-	-	-	-	-
a36	29.233	110	-	-	-	-	-	-	-	-
a37	24.837	45	-	-	-	-	-	-	-	-
a38	28.596	570	0,045	570	0,045	570	0,045	570	0,049	570
a39	30.808	110	-	-	-	-	-	-	-	-
a44	38.771	101	-	-	-	-	-	9	-	6
a45	40.282	136	-	-	-	-	-	-	-	-
a46	40.696	66	-	-	-	-	-	-	-	-
a48	39.800	230	-	-	-	-	-	-	-	-
a53	46.662	183	-	-	-	-	-	-	-	-
a54	22.414	304	-	-	-	-	-	-	-	-
a55	24.694	270	-	-	-	-	-	-	-	-
a60	25.041	3.565	1,813	3.565	1,813	3.565	1,813	3.565	1,813	3.565
a61	60.644	561	-	-	-	-	-	-	-	-
a62	22.917	1.022	-	-	-	-	-	-	-	-
a63	24.447	2.930	-	1	-	1	-	-	-	-
a64	24.100	5.395	-	186	-	1.022	-	408	-	405
a65	28.046	478	-	-	-	-	-	-	-	-
a69	25.822	1.552	-	1	-	3	-	-	-	3
a80	29.977	5.626	-	-	-	1	-	-	-	1

Tabla 4.9: Evaluación de naturaleza de los procesos de búsqueda con *hill-climbing*.

a los 2.000 y 3.000 segundos.

En la tabla 4.11 se aprecia un rendimiento excelente del algoritmo, encontrando el óptimo global de todas las instancias en tiempos inferiores a cero en la mayoría de las instancias. En la instancia *eil76* se requiere tiempos de 1 segundos para encontrar el óptimo y en la prueba *PM - MM* se requirieron 14 segundos para ello.

Al comparar los resultados de todas las tablas en conjunto, se aprecian pocas diferencias en los tiempos de ejecución y en los GAPs. En las instancias *a64*, *f71* y *f72* se aprecian diferencias en los tiempos mas considerables. Respecto a los GAPs, no se aprecian diferencias, ni siquiera en las instancias con problemas. Esto quiere decir que en todas las pruebas se llega a los mismos resultados, lo que cambia es el tiempo en que se estos se obtienen. En la tabla 4.12 se muestra la sumatoria de los tiempos de ejecución y de los GAPs para este escenario de pruebas. Se aprecia que la diferencia en los GAPs es baja, notando una leve mejora en las pruebas *PM - MM* y *MM - PM* con un valor de 19,971 en comparación con

I	G. Paredes-Belmar		PM - PM		PM - MM		MM - PM		MM - MM	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
c50	66.160	84	1,032	84	1,032	84	1,032	84	1,032	84
c75	46.180	13.760	8,155	13.760	8,142	13.760	8,142	13.760	8,144	13.760
tai75A	42.992	4.806	5,785	4.806	5,785	4.806	5,785	4.806	5,785	4.806
tai75B	48.238	15.056	-	1.239	-	903	-	591	-	3.714
tai75C	25.906	4.537	-0,004	13	-0,004	9	-0,004	11	-0,004	-
tai75D	43.869	2.645	3,157	2.645	3,157	2.645	3,157	2.645	3,157	2.645
f45	23.705	80	-	-	-	-	-	-	-	-
f71	72.864	3.483	-	2.665	-	1.968	-	406	0,012	3.483
f72	72.072	3.659	-	213	-	144	-	93	-	671

Tabla 4.10: Evaluación de naturaleza de los procesos de búsqueda con *hill-climbing*.

I	G. Paredes-Belmar		PM - PM		PM - MM		MM - PM		MM - MM	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
eil22	15.947	12	-	-	-	-	-	-	-	-
eil23	7.207	6	-	-	-	-	-	-	-	-
eil30	7.117	99	-	-	-	-	-	-	-	-
eil33	20.409	58	-	-	-	-	-	-	-	-
eil51	50.128	154	-	-	-	-	-	-	-	-
eil76	91.461	1.700	-	1	-	1	-	1	-	14
att48	17.452	284	-	-	-	-	-	-	-	-

Tabla 4.11: Evaluación de naturaleza de los procesos de búsqueda con *hill-climbing*.

las otras cuyo GAP es mayor con valores de 19,984 y 19,989. En los tiempos de ejecución totales se aprecian diferencias más importantes. La prueba con menor tiempos de ejecución total (26.949 segundos) fue la *MM - PM* que considera un enfoque de mejor mejora para la búsqueda *extra-rutas* y un enfoque primera mejora para la búsqueda *intra-rutas*. Esta configuración además presenta los GAPs más bajos. La prueba que presenta peores resultados es la que considera el enfoque de *Mejor Mejora* en ambos procesos de búsqueda. Esto puede deberse a que el proceso de *Mejor Mejora* requiere mucho mas tiempo que el proceso de *Primera Mejora*, debido a que el primero requiere revisar todo el vecindario de búsqueda, mientras que el segundo detiene la búsqueda al encontrar una solución mejor que la actual y permite aprovechar el tiempo moviéndose por el espacio de búsqueda.

Prueba	Tiempo	GAP
PM - PM	29.749	19,984
PM - MM	29.482	19,971
MM - PM	26.949	19,971
MM - MM	33.727	19,989

Tabla 4.12: Resumen con un compilado de los resultados de las pruebas realizadas con *hill-climbing* para determinar la naturaleza de los procesos de búsqueda que otorga los mejores resultados.

Resumen

Al ver los resultados de las tablas 4.8 y 4.12 se pueden ver que en todas las pruebas realizadas con *hill-climbing* se aprecian comportamientos similares. Los GAPs se mantienen prácticamente invariantes, solo presentan pequeñas variaciones del orden de las décimas. Esto significa que el cambio en los parámetros de penalización y elección de la naturaleza de los procesos de búsqueda no afectan en gran medida en la calidad de las soluciones encontradas. Donde si se aprecian diferencias destacables es en los tiempos totales en que se encuentran los óptimos globales de todas las instancias que van de los 33.727 a los 26.949 segundos. La prueba que otorga los mejores resultados es *MM - PM* que además presenta el GAP más bajo.

Cabe destacar que independiente del esquema utilizado, el tiempo que se ahorra respecto al utilizado en el trabajo de G. Paredes-Belmar es importante. El tiempo que requieren para resolver todas las instancias es de 73.779 segundos, en comparación con los 26.949 de la prueba que otorgó los mejores resultados (*MM-PM*). Esto corresponde a tan solo un 36 % del tiempo utilizado por G. Paredes-Belmar.

4.3.2. Iterated Local Search

En esta sección se presentan los resultados obtenidos al evaluar la relevancia del parámetro de perturbación y la selección de la solución sometida a perturbación.

Parámetro de perturbación

La versión de *iterated local search* implementada, requiere fijar el valor de un parámetro de perturbación, el cual indica, el porcentaje de nodos que serán alterados durante la perturbación de las soluciones. Este parámetro puede tomar valores de 0 a 1, y no cambia durante todo el proceso de ejecución del algoritmo. El proceso de perturbación elige al azar dos nodos (los cuales pueden ser de la misma ruta o de distinta) y los intercambia, haciendo que si eran nodos de rutas diferentes, se asignen rutas distintas a dos productores, y si son de la misma ruta, cambie el orden de visita de dichos productores en esa ruta. Este proceso se repite tantas veces como dicte el parámetro de perturbación.

Las tablas 4.13, 4.14 y 4.15 muestran los resultados obtenidos al ejecutar el algoritmo *iterated local search* propuesto sobre las instancias pequeñas, considerando cuatro configuraciones diferentes para el parámetro de perturbación. Para las pruebas se consideraron los valores de 0,05; 0,10; 0,15 y 0,20 del parámetro de perturbación, los cuales corresponden a cambiar el 5 %, 10 %, 15 % y 20 % respectivamente de los nodos de una solución. En la parte izquierda de cada tabla, la primera columna (*I*) corresponde al nombre de la instancia, luego se muestra la calidad del óptimo global obtenido por Paredes-Belmar y el tiempo en que este fue encontrado para dicha instancia. En la parte derecha de la tabla se encuentran las versiones del algoritmo que se ejecutaron: 0,05, 0,10, 0,15 y 0,20, que representan los distintos valores del parámetro de perturbación utilizado. Cada versión fue ejecutada 30 veces, cada una con una semilla aleatoria distinta. En cada ejecución se puede ver el GAP y tiempo (en segundos) obtenido para cada instancia en particular.

Revisando las tablas con los resultados, se puede ver que en la mayoría de las instancias se obtiene el óptimo global en tiempos inferiores a un segundo. En este caso, se logró encontrar el óptimo global para la instancia *a38* que en *hill-climbing* no se pudo encontrar. En ILS siempre se trabaja sobre la misma solución, la cual evoluciona, mediante los procesos de búsqueda local y posterior perturbación para explorar nuevos espacios cercanos. Esta cualidad permitió encontrar el óptimo de la instancia *a38*. En HC una vez ejecutados los procesos de búsqueda se genera una nueva solución aleatoria, perdiendo el avance obtenido en la solución en cada paso.

I	Paredes-Belmar		0,05		0,10		0,15		0,20	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
a33	29.417	62	-	-	-	-	-	-	-	-
a34	30.496	40	-	-	-	-	-	-	-	-
a36	29.233	110	-	-	-	-	-	-	-	-
a37	24.837	45	-	-	-	-	-	-	-	-
a38	28.596	570	-	1	-	1	-	1	-	15
a39	30.808	110	-	-	-	-	-	-	-	-
a44	38.771	101	-	-	-	-	-	-	-	2
a45	40.282	136	-	-	-	-	-	-	-	-
a46	40.696	66	-	-	-	-	-	-	-	-
a48	39.800	230	-	-	-	-	-	-	-	-
a53	46.662	183	-	-	-	-	-	-	-	-
a54	22.414	304	-	-	-	-	-	-	-	-
a55	24.694	270	-	-	-	-	-	-	-	3
a60	25.041	3.565	1,813	3.565	1,813	3.565	1,813	3.565	1,813	3.565
a61	60.644	561	-	-	-	-	-	-	-	-
a62	22.917	1.022	-	-	-	-	-	-	-	1
a63	24.447	2.930	-	-	-	1	-	2	-	1
a64	24.100	5.395	-	-	-	13	-	69	-	141
a65	28.046	478	-	-	-	-	-	1	-	3
a69	25.822	1.552	-	-	-	1	-	1	-	15
a80	29.977	5.626	-	-	-	-	-	2	-	21

Tabla 4.13: Evaluación del valor del parámetro de perturbación con *iterated local search*.

En las tablas hay un conjunto de instancias en las que no fue posible encontrar el óptimo. Estas son las instancias con problemas de consistencias que serán analizadas en detalle más adelante. Se aprecian diferencias importantes en tiempo para algunas instancias como la *a64* con una diferencia de 14 segundos entre pruebas, la *a64* con una diferencia de 128 segundos entre pruebas, la *a80* y *f72* con una diferencia de 19 entre pruebas, la *tai75B* con una diferencia de 199 entre pruebas, la *f71* con una diferencia de 975 entre pruebas y la *eil76* con una diferencia de 30 entre pruebas. Estas diferencias en cuanto al tiempo de ejecución son determinantes para establecer el mejor valor del parámetro de penalización. En la tabla 4.16 se muestra la sumatoria de los tiempos de ejecución y de los GAPs para este escenario de pruebas. Se aprecia que no hay diferencia en los GAPs totales, lo que significa que absolutamente todas las pruebas obtuvieron las mismas soluciones. En los tiempos de ejecución totales se aprecian significativas de aproximadamente de 1.500 segundos. La prueba con menor tiempo de ejecución total (24.888 segundos) fue la *0,05* en la que se utiliza como parámetro de penalización el valor 0,05. Esto quiere decir que en el proceso de perturbación se ven afectados

I	Paredes-Belmar		0,05		0,10		0,15		0,20	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
c50	84	66.160	1,032	84	1,032	84	1,032	84	1,032	84
c75	13.760	46.180	8,136	13.760	8,136	13.760	8,136	13.760	8,136	13.760
tai75A	4.806	42.992	5,785	4.806	5,785	4.806	5,785	4.806	5,785	4.806
tai75B	15.056	48.238	-	4	-	21	-	123	-	203
tai75C	4.537	25.906	-0,004	1	-0,004	4	-0,004	3	-0,004	8
tai75D	2.645	43.869	3,157	2.645	3,157	2.645	3,157	2.645	3,157	2.645
f45	80	23.705	-	-	-	-	-	-	-	-
f71	3.483	72.864	-	19	-	348	-	418	-	994
f72	3.659	72.072	-	2	-	1	-	9	-	21

Tabla 4.14: Evaluación del valor del parámetro de perturbación con *iterated local search*.

I	Paredes-Belmar		0,05		0,10		0,15		0,20	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo	GAP	Tiempo
eil22	12	15.947	-	-	-	-	-	-	-	-
eil23	6	7.207	-	-	-	-	-	-	-	-
eil30	99	7.117	-	-	-	-	-	-	-	-
eil33	58	20.409	-	-	-	-	-	-	-	-
eil51	154	50.128	-	-	-	-	-	-	-	-
eil76	1.700	91.461	-	1	-	6	-	20	-	31
att48	284	17.452	-	-	-	-	-	-	-	-

Tabla 4.15: Evaluación del valor del parámetro de perturbación con *iterated local search*.

el 5 % de los nodos de la solución. Por otro lado, el enfoque que presenta peores resultados en cuanto a tiempo (26.319 segundos) es 0,20 que se da cuando se intercambia el 20 % de los nodos de la solución. De hecho se puede concluir que a medida que se cambia una mayor porcentaje de los nodos de la solución, mayor son los tiempos de ejecución totales para el conjunto de pruebas actuales.

Prueba	Tiempo	GAP
0,05	24.888	19,919
0,10	25.256	19,919
0,15	25.509	19,919
0,20	26.319	19,919

Tabla 4.16: Resumen con un compilado de los resultados de las pruebas realizadas con *iterated local search* para la determinar valor óptimo del parámetro de perturbación.

Selección de solución sometida a perturbación

La versión de *iterated local search* implementada, realiza en cada ciclo de búsqueda una perturbación de la solución, en reemplazo de la generación de una solución aleatoria como lo hace *hill-climbing*. La perturbación de la solución puede realizarse sobre la solución actual (solución que ha pasado por los procesos de búsqueda *extra-rutas* e *intra-rutas* en el ciclo actual) o sobre la mejor solución encontrada hasta el momento a lo largo de todo el proceso de búsqueda. En el primer esquema, siempre se está trabajando sobre la misma solución, la cual va evolucionando en cada momento, mientras que en el segundo esquema, si la perturbación actual y su posterior proceso de mejoramiento no dieron buenos resultados, se descarta esta solución y se vuelve a realizar la perturbación sobre la mejor solución. Esto permite una mayor intensificación del proceso de búsqueda de soluciones, el cual, se centra en la mejor solución y hallazgo de espacios de búsqueda cercanos a esta.

Las tablas 4.17, 4.18 y 4.19 muestran los resultados obtenidos al ejecutar el algoritmo *iterated local search* propuesto en las instancias pequeñas considerando los dos esquemas descritos respecto a la selección de solución sometida a perturbación. En la parte izquierda de cada tabla, la primera columna (*I*) corresponde al nombre de la instancia evaluada, la calidad del óptimo global obtenido por Paredes-Belmar y el tiempo en que fue encontrado. En la parte derecha de la tabla se encuentran las versiones del algoritmo que se ejecutaron: *Actual* que corresponde a la versión que realiza la perturbación sobre la solución actual y *Mejor* que corresponde a la versión que realiza la perturbación sobre la mejor solución. Cada versión fue ejecutada 30 veces, cada una con una semilla aleatoria distinta. Se utilizó 0,05 como valor del parámetro de perturbación. En cada ejecución se puede ver el GAP y tiempo (en segundos) obtenido para una instancia en particular.

Al comparar los resultados de todas las tablas en conjunto, se aprecia a simple vista la mejora que conlleva el realizar el proceso de perturbación sobre la mejor solución respecto a la solución actual. Tanto los GAPs como los tiempos de ejecución varían enormemente. En la prueba *Actual*, no se encuentra el óptimo global casi en ninguna instancia, pero se obtienen GAPs muy pequeños. Esto significa que en la mayoría de las instancias se obtuvieron valores muy cercanos al óptimo. En la tabla 4.20 se muestra la sumatoria de los tiempos de ejecución

I	Paredes-Belmar		Actual		Mejor	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo
a33	29.417	62	0,003	62	-	-
a34	30.496	40	-	-	-	-
a36	29.233	110	-	-	-	-
a37	24.837	45	0,072	45	-	-
a38	28.596	570	0,518	570	-	1
a39	30.808	110	-	-	-	-
a44	38.771	101	0,286	101	-	-
a45	40.282	136	0,050	136	-	-
a46	40.696	66	-	-	-	-
a48	39.800	230	0,003	230	-	-
a53	46.662	183	0,004	183	-	-
a54	22.414	304	-	1	-	-
a55	24.694	270	0,012	270	-	-
a60	25.041	3.565	1,989	3.565	1,813	3.565
a61	60.644	561	0,005	561	-	-
a62	22.917	1.022	0,026	1.022	-	-
a63	24.447	2.930	0,078	2.930	-	-
a64	24.100	5.395	0,116	5.395	-	-
a65	28.046	478	0,139	478	-	-
a69	25.822	1.552	0,023	1.552	-	-
a80	29.977	5.626	0,053	5.626	-	-

Tabla 4.17: Evaluación de enfoques de perturbación con *iterated local search*.

y de los GAPs para este escenario de pruebas. Se aprecia una diferencia de aproximadamente 1,2 en los GAPs. Esta diferencia no es tan grande, en comparación a la diferencia que existe en los tiempos, la cual presenta diferencias de 47.902 segundos. Esta diferencia se debe a que en la prueba *Actual* se ocupó todo el tiempo de ejecución en las instancias en las cuales no fue posible hallar el óptimo global.

Resumen

Al ver los resultados de las tablas 4.16 y 4.20 se puede ver que la modificación de los parámetros de perturbación no afecta en gran medida a los GAPs acumulados, pero si afecta en parte a los tiempos de ejecución. La modificación de la solución a la cual se le aplica perturbación tiene un efecto muy grande en el tiempo de ejecución y un efecto considerable

I	Paredes-Belmar		Actual		Mejor	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo
c50	66.160	84	1,035	84	1,032	84
c75	46.180	13.760	8,244	13.760	8,136	13.760
tai75A	42.992	4.806	5,955	4.806	5,785	4.806
tai75B	48.238	15.056	0,060	15.056	-	4
tai75C	25.906	4.537	0,143	4.537	-0,004	1
tai75D	43.869	2.645	3,184	2.645	3,157	2.645
f45	23.705	80	0,030	80	-	-
f71	72.864	3.483	0,054	3.483	-	19
f72	72.072	3.659	0,010	3.659	-	2

Tabla 4.18: Evaluación de enfoques de perturbación con *iterated local search*.

I	Paredes-Belmar		Actual		Mejor	
	Calidad	Tiempo	GAP	Tiempo	GAP	Tiempo
eil22	15.947	12	-	-	-	-
eil23	7.207	6	-	-	-	-
eil30	7.117	99	0,042	99	-	-
eil33	20.409	58	-	-	-	-
eil51	50.128	154	0,014	154	-	-
eil76	91.461	1.700	0,031	1.700	-	1
att48	17.452	284	-	-	-	-

Tabla 4.19: Evaluación de enfoques de perturbación con *iterated local search*.

en los GAPs. Dado esto se puede concluir que la configuración de parámetros que obtiene mejores resultados es la de utilizar *iterated local search* con una perturbación sobre la mejor solución del 5 %, es decir, utilizando un parámetro de perturbación de 0,05. Esta configuración corresponde a la prueba *Mejor* o 0,05 las cuales como se puede ver en las tablas son la misma.

Prueba	Tiempo	GAP
Actual	72.790	22,178
Mejor	24.888	19,919

Tabla 4.20: Resumen con un compilado de los resultados de las pruebas realizadas con *iterated local search* para la determinar la solución a la cual aplicarle la perturbación.

El tiempo total de la prueba *Actual* fue de 72.790 segundos el cual es muy similar al obtenido por G. Paredes-Belmar que fue de 73.779 segundos. Al pasar a utilizar la mejor solución en el proceso de perturbación, el tiempo disminuyó a los 24.888 segundos. Este cambio en ILS es muy significativo y marco un antes y un después del desarrollo e implementación del algoritmo. Los 24.888 segundos de tiempo de la prueba *Mejor* corresponden al 33 % del tiempo utilizado en G. Paredes-Belmar para encontrar el óptimo de las 37 instancias.

Al comparar los resultados del algoritmo *iterated local search* con *hill-climbing* se aprecia una mejora en los tiempos de unos 2.000 segundos aproximadamente a favor de *iterated local search*, comparando las pruebas con mejores resultados en ambos métodos. Los GAPs se mantienen muy parecidos en ambos métodos, presentando leves mejoras en *iterated local search*, esto es, un menor valor del GAP total. Con esto se puede concluir que el algoritmo *iterated local search* es el método que entregó los mejores resultados para resolver el conjunto de las 37 instancias de pequeño tamaño utilizadas en este trabajo. El algoritmo *hill-climbing* no se queda lejos en cuanto a resultados, por lo que puede ser una alternativa factible para resolver instancias con similares características.

4.3.3. Análisis de Convergencia

En esta sección se analiza en detalle el funcionamiento del proceso de búsqueda en las instancias de prueba en las que no fue posible obtener el óptimo global. Este análisis se realizará utilizando la versión del algoritmo que dio mejores resultados: *iterated local search* con perturbación sobre mejor solución y con parámetro de perturbación de 0,05.

a60

Esta instancia no presenta problemas de consistencia, pero se considera como caso de estudio pues no fue posible encontrar el óptimo global publicado por Paredes-Belmar con ninguno de los métodos propuestos. El autor obtuvo como óptimo global una solución con calidad de 25.041 en 3.565 segundos. En la figura 4.3 se muestra el gráfico de convergencia para 40 ejecuciones con distintas semillas aleatorias. Los puntos azules corresponden a

las calidades obtenidas a lo largo del tiempo de ejecución y la línea verde al óptimo global publicado. El algoritmo fue dejado correr durante 3.565 segundos, pero en menos de 600 segundos las 40 ejecuciones se quedaron estancadas en una calidad de 24.587. Después de este tiempo no se encontró una solución con calidad superior a los 24.587. Este comportamiento se da en muchas de las instancias cuando se encuentra el óptimo global, por lo que, se sospecha que 24.587 sea el correcto óptimo global de la instancia *a60*.

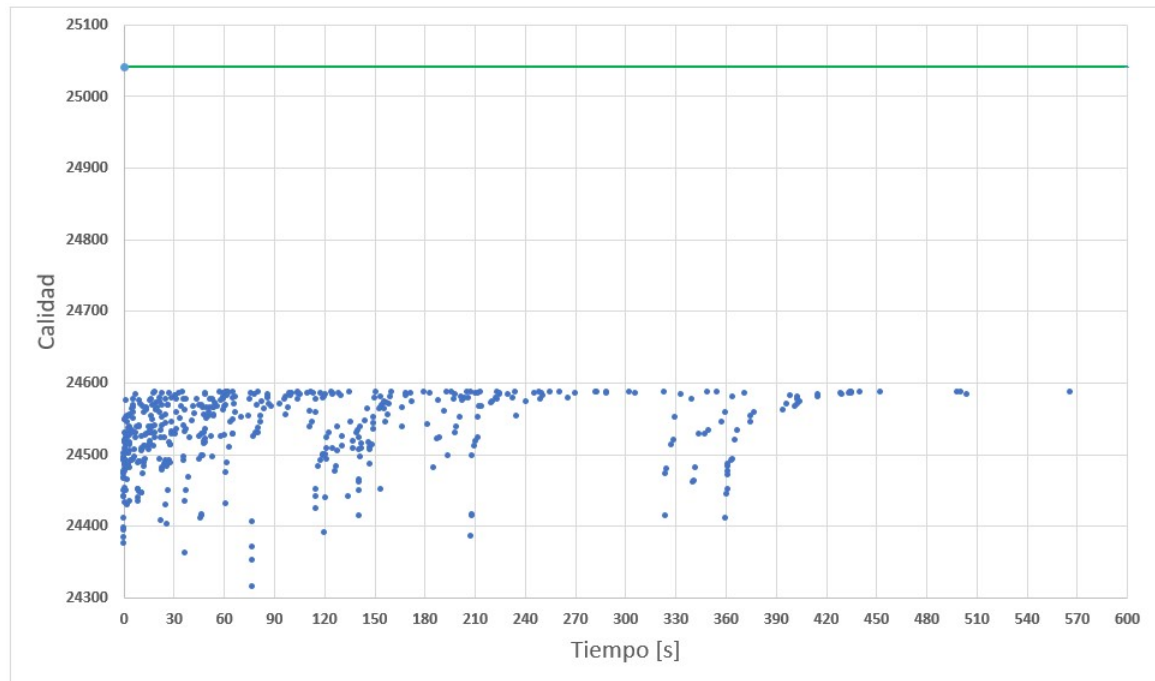


Figura 4.3: Convergencia para la instancia *a60*.

c50

Esta instancia presenta problemas de consistencia, ya que, de acuerdo a la información de la instancias, se tiene una producción de leche mayor a lo que pueden llevar todos los camiones. La leche total que hay en las granjas es de 97.300 litros. La capacidad total de los camiones es la siguiente: $35.000 + 30.000 + 30.000 = 95.000$ litros, la cual, es inferior al total de leche que hay disponible. Esto es un problema, ya que una de las restricciones especifica que se debe recoger toda la leche y visitar a todas las granjas. Para resolver este problema se modificó el archivo de instancia, aumentando la capacidad de los camiones de

30.000 a 35.000 litros. Con esto quedan 3 camiones de 35.000 litros de capacidad para ir a buscar un total de 97.300 litros a 50 granjas. Con este cambio, el óptimo global cambia respecto al publicado por Paredes-Belmar. Como no se tiene conocimiento de su valor, se deja ejecutando el algoritmo por un largo período de tiempo. La figura 4.4 muestra la convergencia del proceso de búsqueda para 40 ejecuciones con distintas semillas aleatorias. Los puntos azules corresponden a las calidades obtenidas a lo largo del tiempo de ejecución y la línea verde al óptimo global publicado de 66.160. Al cabo de tan solo 10 segundos, las 40 ejecuciones llegan al valor de 65.477. Después de varias horas ninguna obtuvo un valor superior a este. Dado esto, es muy probable de que se este ante el óptimo global para la instancia *c50* modificada.

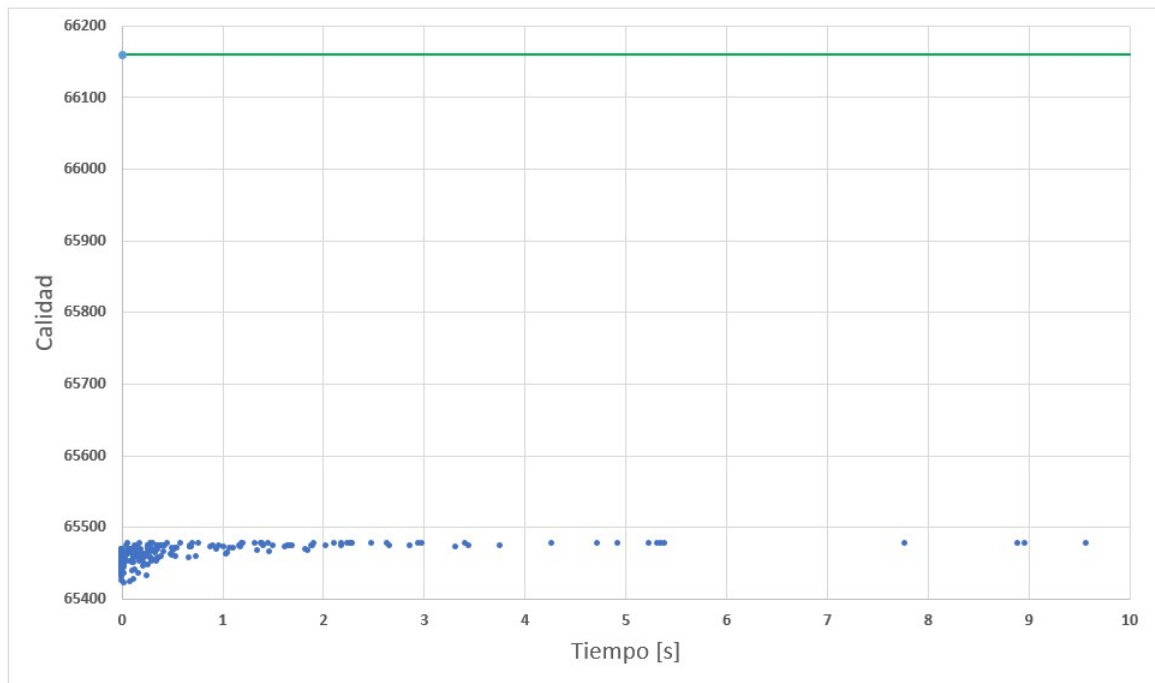


Figura 4.4: Convergencia para la instancia *c50*.

c75

Esta instancia presenta problemas de consistencia, ya que, los requerimientos de leche de la planta superan a la leche total producida por todas las granjas. En total hay 68.200 litros de leche, con 23.400 de tipo A, 23.350 de tipo B y 21.450 de tipo C. Los requerimientos

de leche de la planta son: 40.000 de A, 45.000 de B y 50.000 de C, cuya suma alcanza los 135.000 litros de leche. Esto es mucha mas leche de la que hay en todas las granjas juntas. Para utilizar esta instancia se realizó un ajuste en los requerimientos de la planta como sigue: 10.000 de tipo A, 25.000 de tipo B y 25.000 de tipo C. Con estos cambios, los requerimientos de leche de cada tipo pueden ser satisfechos mezclando leche de tipo A, con B y C, para que estos últimos puedan alcanzar la cuota de los 25.000 litros. Como no se tiene conocimiento del óptimo global de esta instancia, el algoritmo se deja ejecutando durante 13.760 segundos. La figura 4.5 muestra el proceso de convergencia para 40 ejecuciones con distintas semillas aleatorias. Los puntos azules corresponden a las calidades obtenidas a lo largo del tiempo de ejecución y la línea verde al óptimo global publicado cuyo valor alcanza los 46.180.

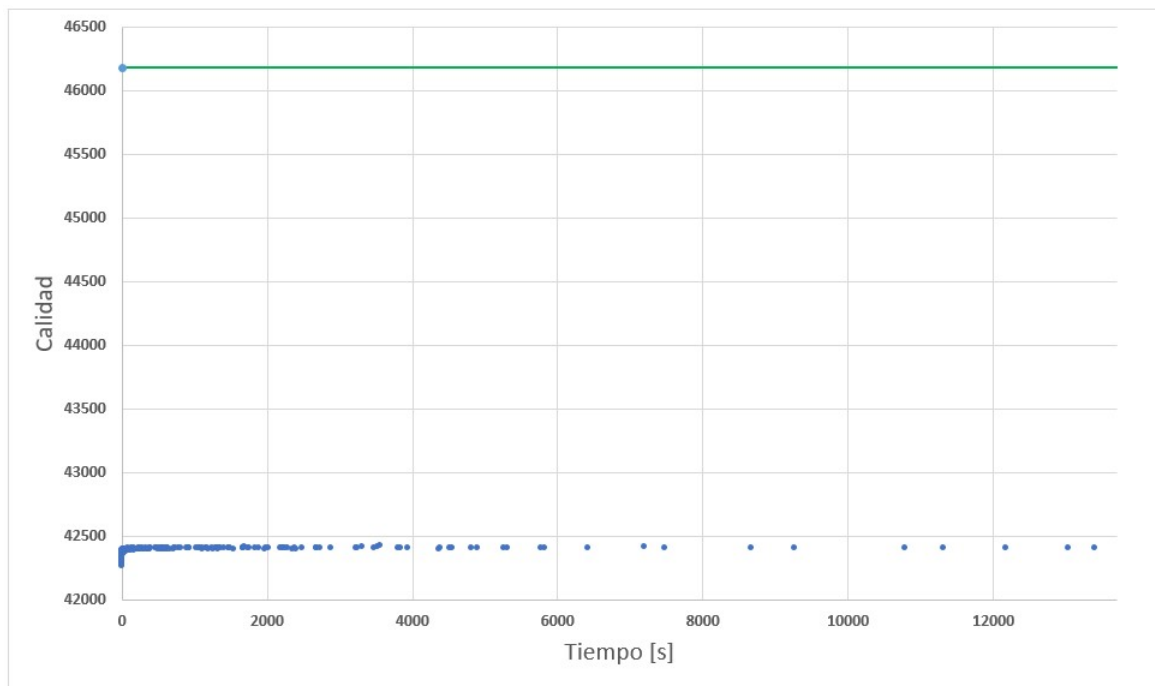


Figura 4.5: Convergencia para la instancia *c75*.

A diferencia de los casos anteriores, acá no se obtuvo una convergencia absoluta para todas las ejecuciones y se encontraron mejores soluciones a lo largo de todo el tiempo de ejecución. Para analizar los resultados de mejora manera se construye un boxplot (figura 4.6) con 40 ejecuciones de *hill-climbing* y otras 40 para *iterated local search* durante 13.760 segundos. Un boxplot, también llamado diagrama de caja y bigote, es un método estandarizado para

representar una serie de datos numéricos a través de sus cuartiles. Con esto se puede ver a simple vista la mediana y los valores atípicos del conjunto de datos.

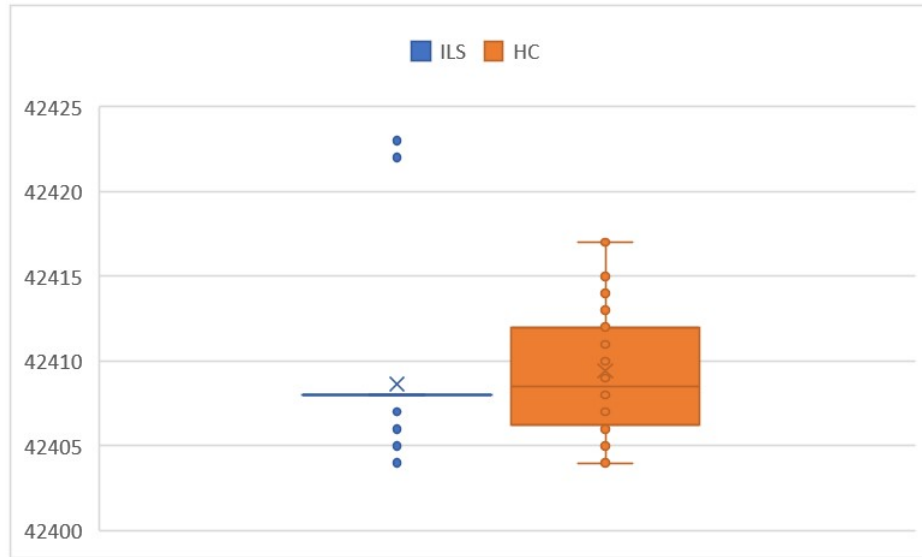


Figura 4.6: Boxplot con los resultados de la instancia *c75*.

Utilizando *iterated local search* se obtiene una menor dispersión en cuanto a los resultados, concentrándose la mayoría en torno a los 42.408 de calidad. Excepcionalmente, se aprecian 2 outliers que corresponden a calidades de 42.422 y 42.423. Por otro lado, en HC la mayoría de las soluciones se encuentran a menos de una desviación de la media que es 42.409. Con estos resultados y dado el extenso tiempo de ejecución, no se puede asegurar cual es el máximo global, pero si se puede decir que esta cercano a los valores encontrados.

tai75A

Esta instancia presenta problemas de consistencia, ya que, el beneficio óptimo obtenido en [36] es de 65.477, pero según el archivo de instancia con la que se cuenta y sin considerar mezcla ni costos, se obtiene un beneficio máximo de 42.992, que es inferior al publicado. Revisando el archivo de instancia se podrían obtener soluciones de calidad como máximo de 42.992 si es que no se incluye una disminución asociada a los costos en ruta. Entonces esto nos deja con una instancia sin óptimo global, solo se tiene una cota máxima para la calidad

de las soluciones. Dado esto, se ejecutó el algoritmo durante el mismo tiempo que en [36]. La figura 4.7 muestra la convergencia de 40 ejecuciones.

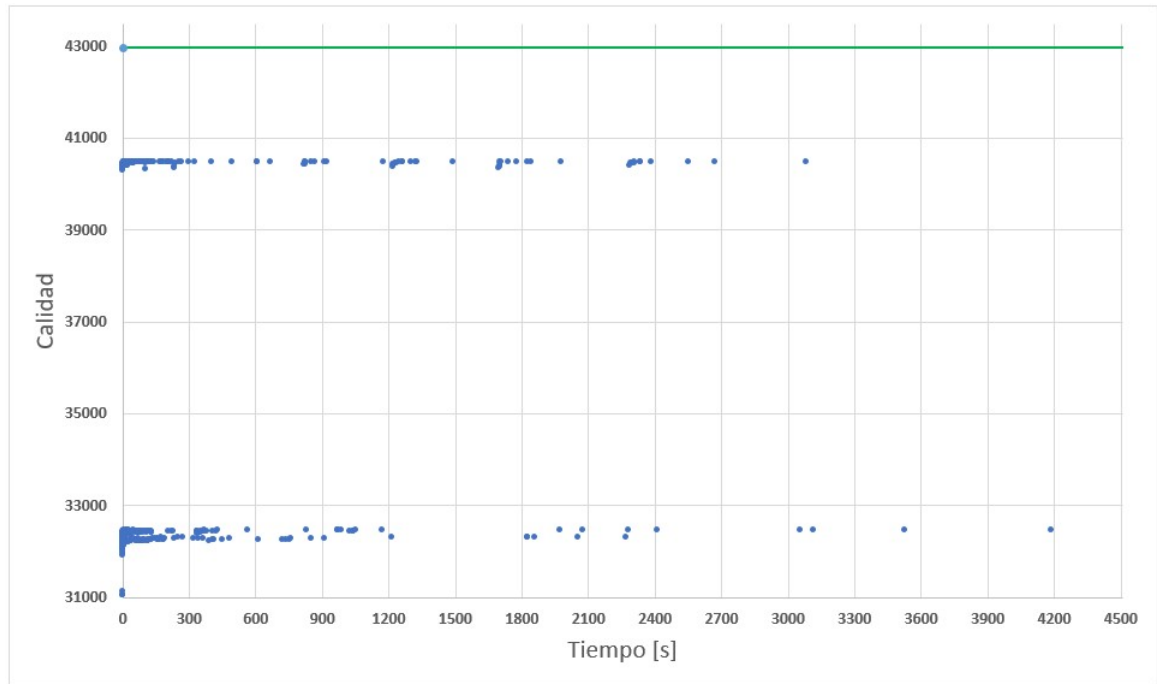


Figura 4.7: Convergencia para la instancia *tai75A*.

Se puede apreciar en la gráfica que hay ejecuciones que se quedaron estancadas por mucho tiempo en valores cercanos a los 33.000, mientras que otras de inmediato llegaron a valores cercanos a los 40.000. Para un mejor entendimiento de la convergencia y la distribución de los puntos se construye un boxplot (figura 4.8) con 40 ejecuciones de *hill-climbing* y otras 40 para *iterated local search* durante 4.806 segundos. Se puede ver como la dispersión en cuanto a los resultados de *iterated local search* es muy grande, mientras que en *hill-climbing* todas las pruebas convergen al mismo valor. Lo destacable de estas pruebas es que ambos métodos llegan al mismo valor máximo: 40.505. Este podría ser el óptimo global para esta instancia modificada.

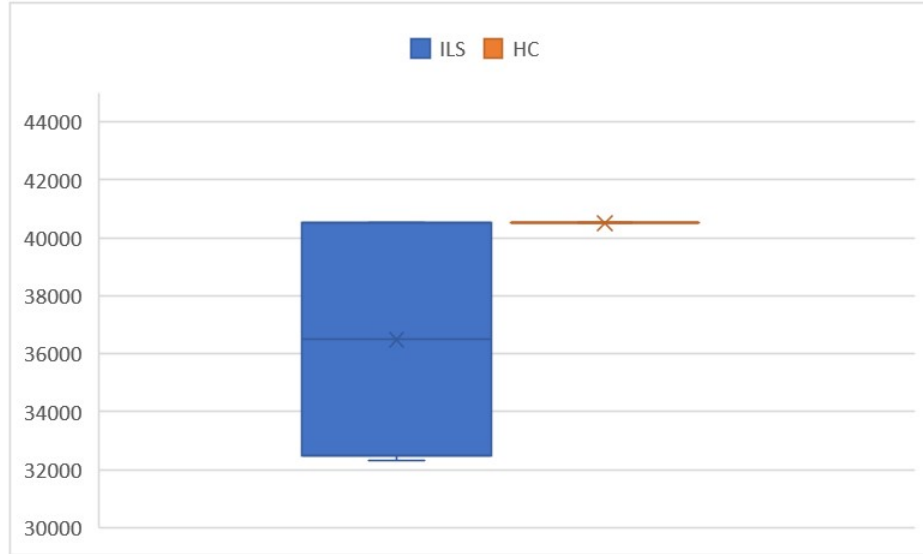


Figura 4.8: Boxplot con los resultados de la instancia *tai75A*.

tai75D

En esta instancia pasa lo mismo que en la *tai75A*. El beneficio óptimo obtenido en [36] es de 65.477, pero calculando el beneficio total sin considerar mezcla ni costos, se obtiene 43.869. Este valor es la cota máxima de calidad que se puede obtener para esta instancia. Dado esto, se ejecutó el algoritmo el mismo tiempo utilizado en el artículo. El gráfico de convergencia se puede ver en la figura 4.9. Se puede ver que hay pocas ejecuciones que llegaron a un valor cercano a los 42.500 y fue casi al comienzo de la prueba. La mayoría se quedó en valores más bajos y no lograron aumentar su valor. Para ver el análisis más en detalle, se construye un boxplot (figura 4.10) con 40 ejecuciones de *hill-climbing* y otras 40 para *iterated local search* durante 43.869 segundos. En él, se puede ver como con *iterated local search* se aprecia una dispersión bastante grande en cuanto a los valores de las soluciones obtenidos después de la prueba completa. En cambio, con *hill-climbing* todas las ejecuciones convergieron al mismo valor: 42.484. Este es buen candidato para ser el óptimo global de esta instancia modificada considerando el desempeño mostrado por *iterated local search* en casos anteriores.

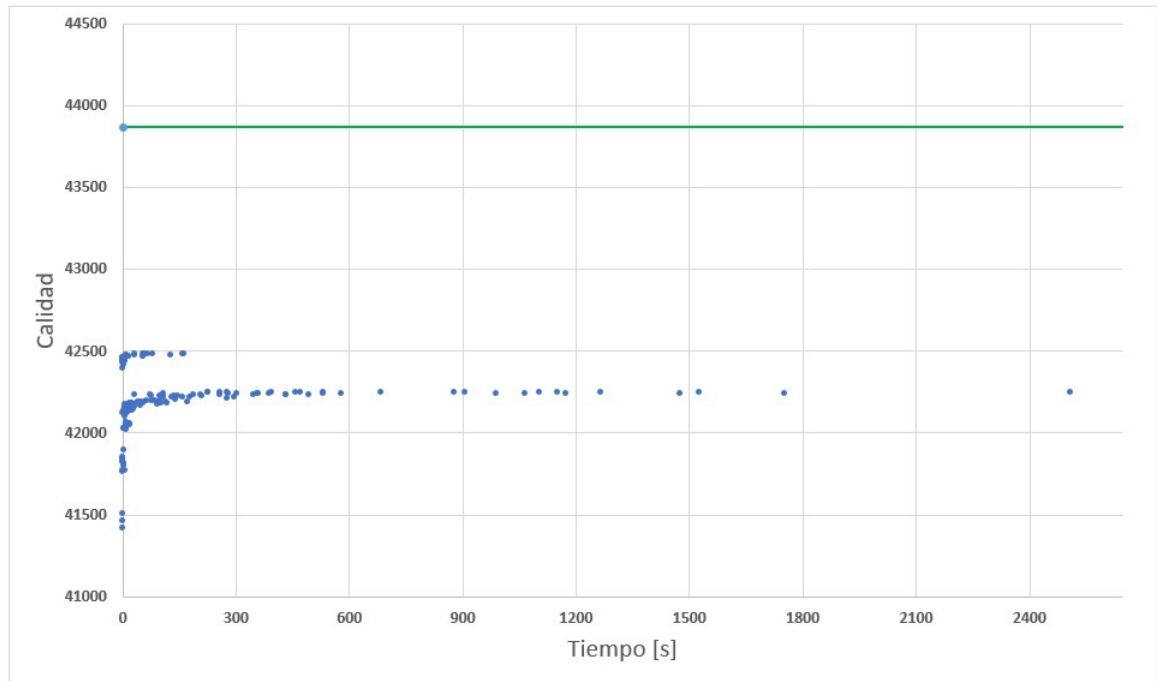


Figura 4.9: Gráfico de convergencia para la instancia *tai75D*.

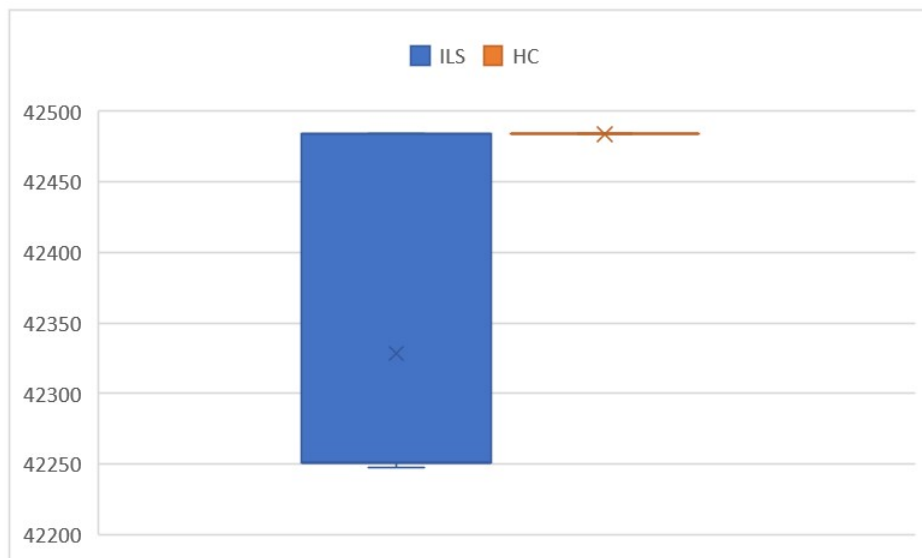


Figura 4.10: Boxplot con los resultados de la instancia *tai75D*.

4.3.4. Instancias Reales

En este apartado se realiza un análisis profundo de las instancias reales, para las cuales no se cuenta información acerca del óptimo global. Las instancias reales fueron construidas a partir de la información de 500 granjas distribuidas en el sur de Chile, variando solo la cantidad de camiones asignados para realizar las rutas.

Comparación de métodos

Primero se comparan los dos algoritmos implementados para resolver el problema *hill-climbing* e *iterated local search*. Se ejecutaron 40 pruebas de cada algoritmo, con semillas aleatorias distintas, por 5.959 segundos, que fue el tiempo utilizado por G. Paredes-Belmar para encontrar un máximo local sin utilizar particiones de terreno.

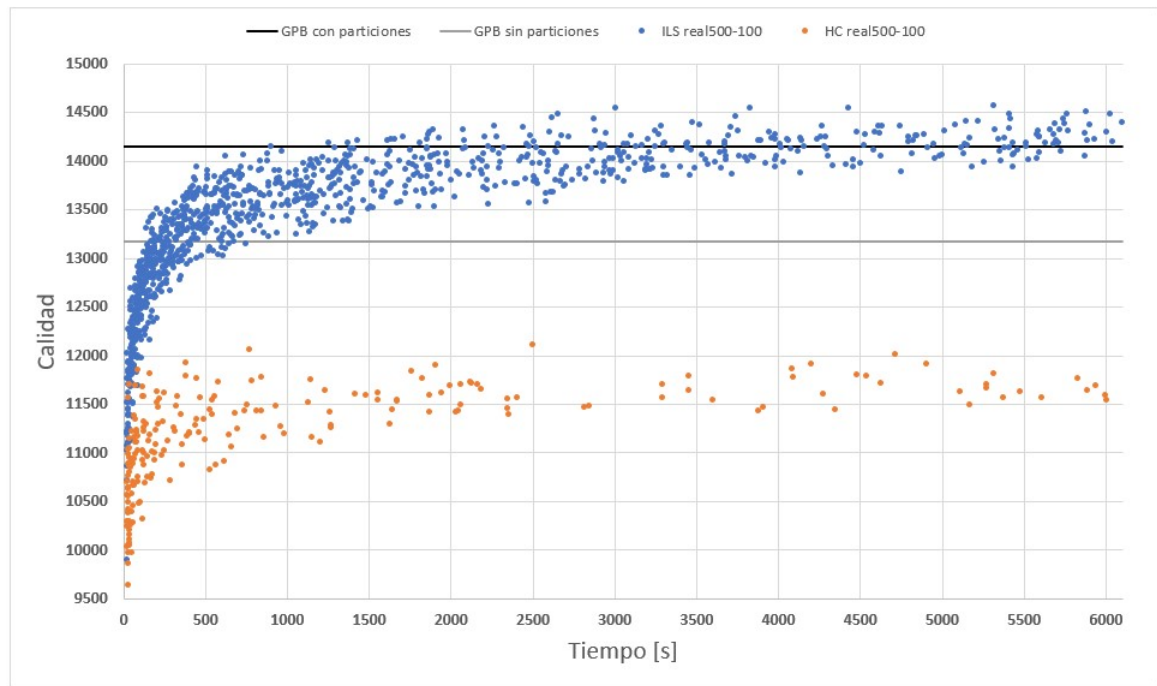


Figura 4.11: Convergencia de la instancia *real500-100* utilizando *hill-climbing* e *iterated local search*.

Los puntos de color naranja corresponden a las 40 ejecuciones con *hill-climbing*, mientras

que los puntos de color azul corresponden a las 40 ejecuciones con *iterated local search*. La recta horizontal negra de la parte superior, corresponde al máximo valor en cuanto a calidad encontrado por G. Paredes-Belmar para esta instancia aplicando particiones sobre el terreno. La recta horizontal gris, ubicada un poco más abajo de la negra, corresponde al máximo valor en cuanto a calidad encontrado por G. Paredes-Belmar para esta instancia sin aplicar particiones sobre el terreno. Se puede observar a simple vista que *iterated local search* obtiene mejores soluciones que *hill-climbing*. En los primeros minutos de ejecución alcanza valores muy superiores a los que *hill-climbing* obtiene durante todo el tiempo de ejecución. Por otro lado, *hill-climbing* siquiera es capaz de obtener valores superiores al máximo de G. Paredes-Belmar sin particiones. La curva del conjunto de soluciones en *iterated local search* aumenta en forma exponencial en los primeros minutos, luego se aprecia un aumento paulatino y continuo a lo largo de todo el tiempo de ejecución. En cambio, en *hill-climbing* se no se aprecia crecimiento aparente a lo largo de todo el tiempo de ejecución. Se ve un crecimiento acelerado al principio, pero luego no se aprecia mejoramiento de las soluciones.

Comparación de instancias

En general, durante el armado de rutas, el algoritmo no utiliza todos los camiones para la recolección de leche. Los camiones que suelen no utilizarse son los de menor capacidad. Con esto en mente, se creo un conjunto de instancias adicionales (descrito en la sección 4.2.2). En estos casos se obliga al algoritmo a utilizar la cantidad de camiones con la que se cuenta. Concretamente se probó disminuyendo los camiones a 88, 77 y 72. En la figura 4.12 se muestra una gráfica con la convergencia de las instancias reales utilizando 72 camiones (puntos de color amarillo), 77 camiones (puntos de color verde), 88 camiones (puntos de color violeta) y 100 camiones (puntos de color azul). Estas instancias fueron ejecutadas utilizando el algoritmo *iterated local search* con un tiempo de ejecución de 5.956 segundos. En la gráfica se puede apreciar como, a menor cantidad de camiones, hay una mayor es la rapidez de convergencia a buenas soluciones.

Para una mejor comprensión se han separado los gráficos de las 4 instancias. Estos se presentan en la figura 4.13. Se puede apreciar como en la instancia *real500-72* se logra una convergencia más rápida, y en la instancia *real500-100* se logra una convergencia un tanto

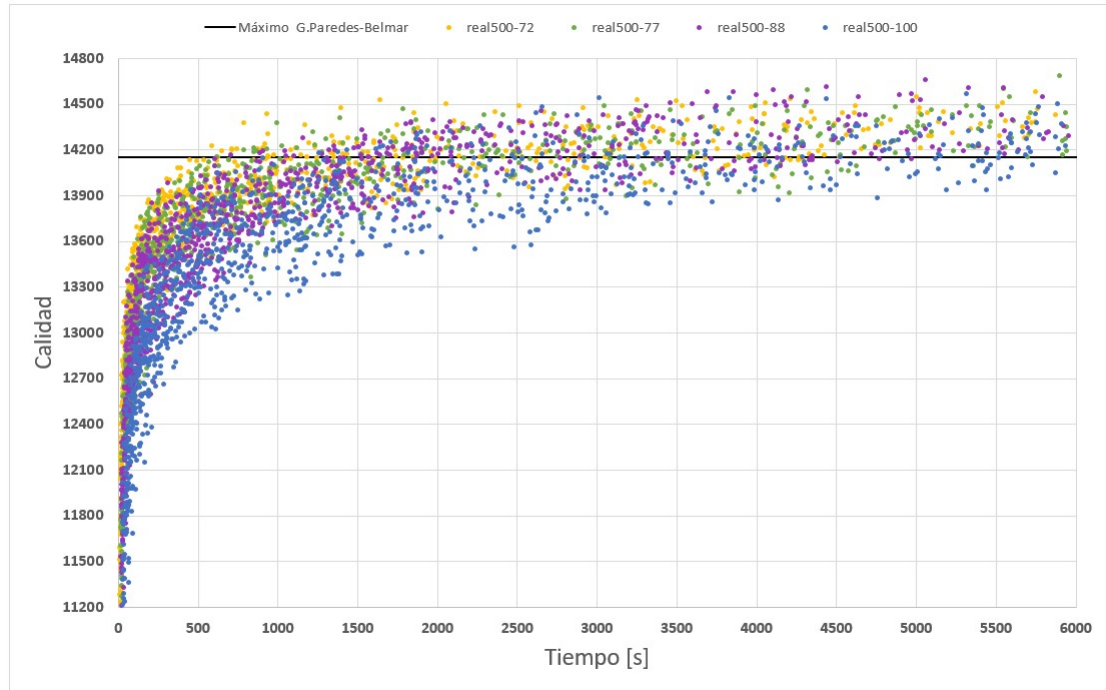


Figura 4.12: Convergencia de las instancias *reales* utilizando *iterated local search*.

más lenta que en las demás. Las 4 instancias llegan a resultados similares, lo que cambia es el tiempo de convergencia a estos valores.

Convergencia de instancias reales

Debido a que no se sabe el valor del óptimo global de las instancias reales, se realizó una ejecución extensa por 26.500 segundos. Con esto, se espera obtener un valor más alto que el encontrado por G. Paredes-Belmar en [36]. En la figura 4.14 se muestra una gráfica con la convergencia de las instancias reales *real500-72* y *real500-100*, utilizando *iterated local search* con un tiempo de ejecución de 24.500 segundos. La línea negra horizontal corresponde al valor obtenido por Paredes-Belmar, cuyo valor fue de 14.155. El máximo valor encontrado para la instancia real con 72 camiones (*ILS 72*) fue de 14.844 a los 23.361 segundos y para la instancia real con 100 camiones (*ILS 100*) fue de 14.846 a los 26.432 segundos. Si bien, el tiempo requerido para encontrar estos valores fue alto, es importante considerar que en [36] este valor se obtiene a los 57.962 segundos. Valor superado en el caso *ILS 72* a

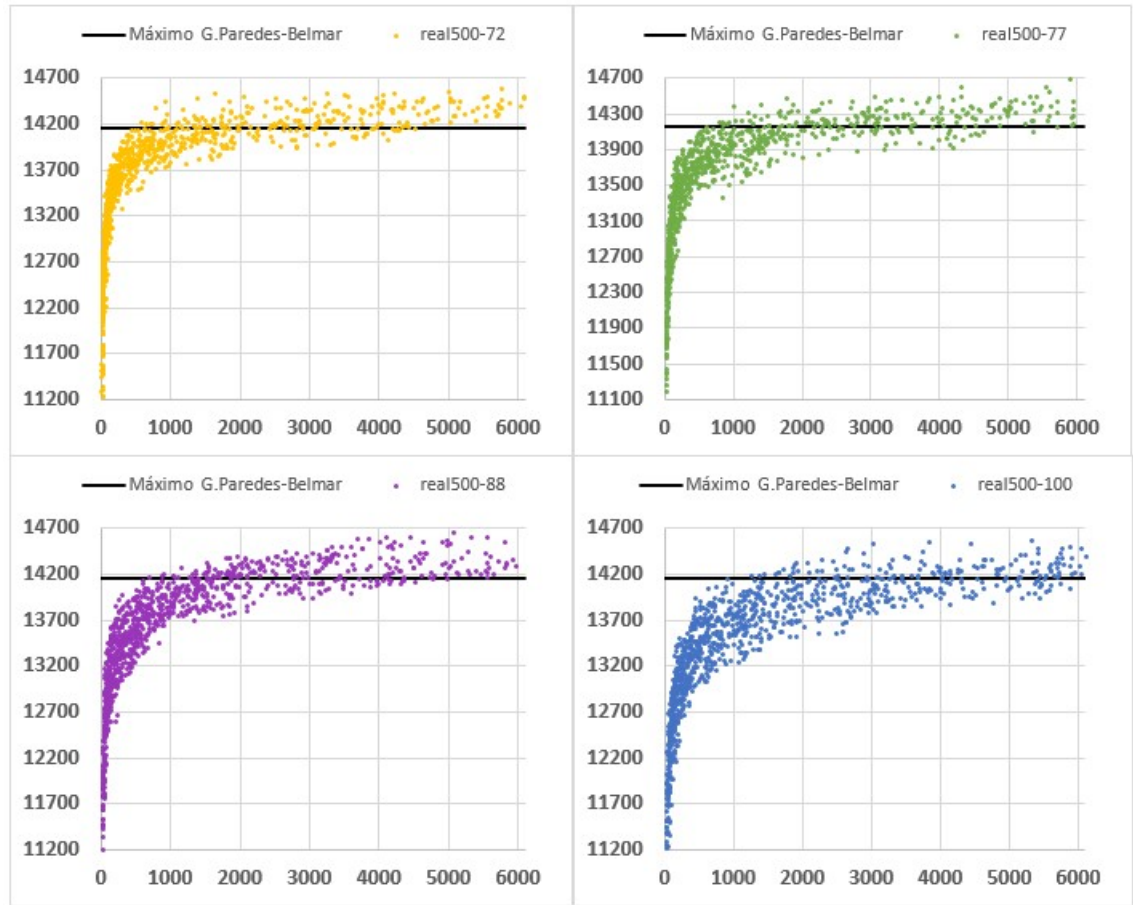


Figura 4.13: Convergencia de las instancias *reales* utilizando *iterated local search*.

los 581 segundos y en el caso *ILS 100* a los 899 segundos. En la figura 4.14 se puede apreciar el mismo comportamiento visto anteriormente. A menor cantidad de camiones, mayor es la convergencia, pero al largo plazo ambas instancias consiguen los mismos resultados.

4.4. Conclusiones

En esta sección se compararon los dos algoritmos de búsqueda local implementados: *hill-climbing* e *iterated local search*. El objetivo fue determinar la versión de cada uno que otorgue los mejores resultados, variando entre diferentes enfoques y combinación de parámetros. En general ambos algoritmos presentaron resultados muy parecidos, tanto en tiempos de ejecución como en calidad de las soluciones encontradas. Algunos casos particulares, como el

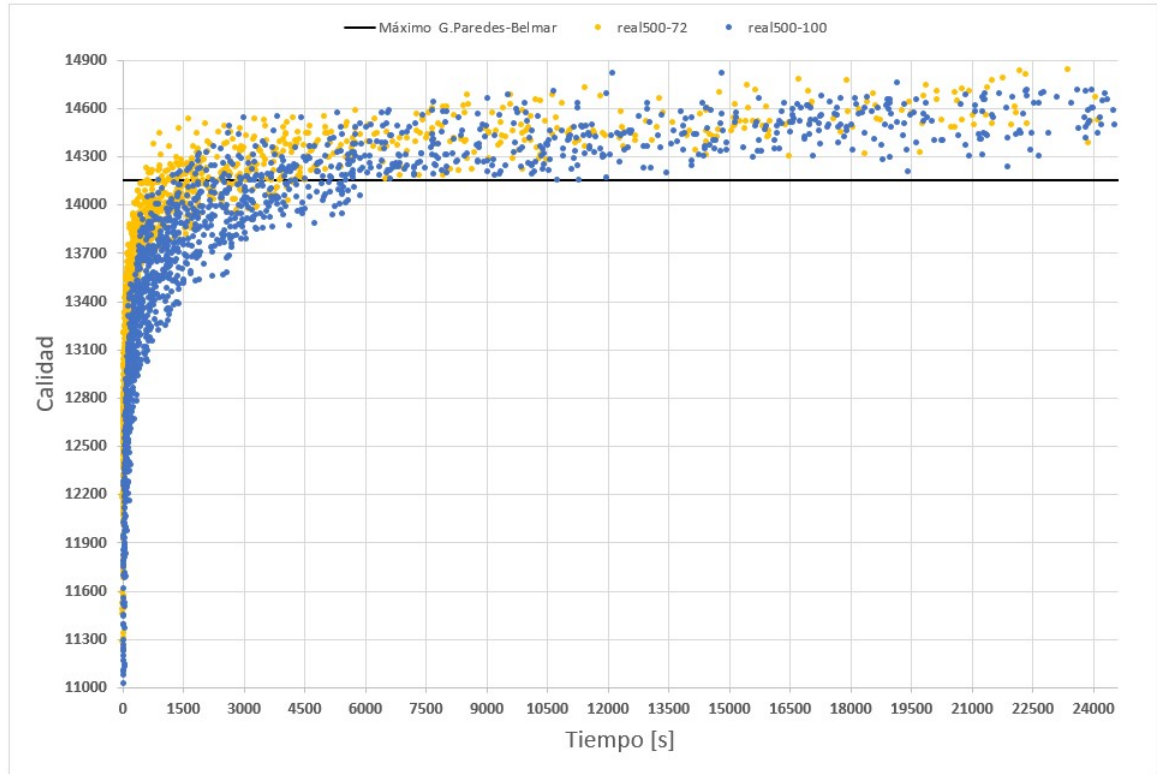


Figura 4.14: Convergencia de la instancias *real500-72* y *real500-100* utilizando *iterated local search* por un tiempo de 26.500 segundos.

cambio en la solución a la cual se le aplica la perturbación, marcaron un antes y un después en el desarrollo del algoritmo *iterated local search*, ya que el cambio de utilizar la solución actual a la mejor, presentó mejoras significativa en cuanto a tiempos de ejecución. Las pruebas con distintos valores del parámetro de perturbación permitió obtener conclusiones claras respecto a los resultados obtenidos con *iterated local search*. Específicamente, se encontró que cuando más grande es el parámetro de perturbación mayores son los tiempos de ejecución del algoritmo. En las pruebas con *hill-climbing* se encontró una combinación especialmente buena de enfoques y parámetros que permite obtener resultados muy cercanos a los de *iterated local search*, estos son: establecer cada parámetro de penalización con el valor de 10 y utilizar un enfoque de *mejor mejora* en la búsqueda local *extra-rutas* y un enfoque de *primera mejora* en la búsqueda local *intra-rutas*.

Teniendo las versiones del algoritmo que otorgaron los mejores resultados sobre las instancias pequeñas, se probó cada una sobre el conjunto de instancias reales. El algoritmo *iterated*

local search fue un claro ganador. A partir de acá, se realizó un análisis de convergencia para determinar cual instancia obtiene los mejores resultados con mayor rapidez. La instancia con 72 camiones presentó los mejores resultados de convergencia. Con esto se concluyo que, si bien, a menor cantidad de camiones, mayor es la rapidez de convergencia, todas las instancias obtienen resultados similares a largo plazo, la diferencia esta en la rapidez en que los obtienen.

Capítulo 5

Conclusiones

Los VRP son problemas que llevan tiempo estudiándose en la literatura ya que la distribución de productos desde depósitos a usuarios finales juega un papel central en la gestión de la mayoría de los sistemas logísticos. La necesidad de transportar productos de la industria alimentaria agrega naturalmente al VRP componentes adicionales como la inclusión de compartimentos, restricciones en cuanto al tiempo de entrega, periodicidad de visita, gestión de diversos tipos de producto, entre otros. Por su parte, el problema de enrutamiento de vehículos para la recolección de leche, busca encontrar rutas para un conjunto de camiones que, partiendo desde una planta, recogen leche de una cierta calidad desde un conjunto de productores y la llevan de regreso a la planta para su posterior procesamiento, con el agregado adicional de que al existir diferentes tipos de leche, se permite la mezcla de esta tanto en la ruta, como en la planta. La mezcla de leche en los camiones, puede producir una disminución de los beneficios, pero permite obtener rutas más óptimas, cuya disminución de costos es mayor a la pérdida del beneficio.

El modelamiento del problema de recolección de leche con mezcla como extensión del VRP permitió resolver instancias grandes y medianas en tiempos bastante razonables, pero no fueron aplicables a instancias de gran tamaño. En este trabajo se presentan dos propuestas basadas en búsqueda local para el problema de recolección de leche con mezclas.

Los algoritmos implementados para resolver el problema de enrutamiento de vehículos

considerando mezcla fueron *hill-climbing* e *iterated local search*, los cuales obtienen soluciones de buena calidad en tiempos bastante acotados respecto a los de la literatura. Esto valida la existencia de técnicas de búsqueda local en comparación a las técnicas completas, cuya implementación resulta más compleja y tiempos de ejecución son muy extensos. En el otro extremo, también se valida la incorporación de estas técnicas en reemplazo de la asignación de rutas en forma manual, la cual, además de requerir mucho tiempo, representa rutas ineficientes que no aprovechan las ventajas de la mezcla, ni la optimización de recursos.

El proceso de implementación de estos algoritmos requirió evaluar diferentes enfoques y combinaciones de parámetros en las instancias de prueba. Entre los que se destacan, la asignación de parámetros de penalización, selección de enfoque primera mejora o mejor mejora en procesos de búsqueda local, asignación de parámetros de perturbación en *iterated local search* y selección de la solución a la cual se le aplican las perturbaciones. Las distintas pruebas, con su posterior análisis y comparación, permitieron obtener versiones de los algoritmos del nivel lo suficiente alto como para ser probadas con las instancias reales, las cuales, aumenta considerablemente en cantidad de granjas y número de camiones.

En cuanto a los resultados obtenidos, tanto *hill-climbing* como *iterated local search* obtienen el óptimo global de la mayoría de las 37 instancias de prueba utilizadas. El algoritmo *iterated local search* obtuvo el óptimo en una mayor cantidad de instancias, en comparación a *hill-climbing*. Además, obtuvo tiempos de ejecución en general fueron menores. Del conjunto de instancias de prueba, hay un grupo en las que no fue posible obtener el óptimo global. A estas se les realizó un análisis de convergencia para determinar la causa del problema. En muchas de ellas se encontró un posible óptimo global distinto al publicado en el estado del arte. En otros casos, simplemente las instancias requieren tiempos de ejecución bastante extensos para su resolución.

En la instancias reales *iterated local search* logra obtener mejores resultados que el estado del arte, tanto en calidad de las soluciones como en tiempos de ejecución. Una conclusión interesante en este apartado fue que a menor cantidad de camiones, mayor es la rapidez de convergencia de las soluciones, esto aplicado al rango de camiones con los que se probó el algoritmo. Si bien, la convergencia es mayor a menor cantidad de camiones, todas las instancias lograron obtener resultados similares, el cambio esta en el tiempo en que obtienen

dichos resultados. Por su parte *hill-climbing*, se queda atrás en este conjunto de instancias, presentando soluciones que no se acercan a las del estado del arte en cuanto a calidad de las soluciones.

Para el futuro, algunos ponentes de la literatura proponen la incorporación de camiones con distintos tamaños de compartimentos. Además de la inclusión de centros de recogida que proponen la incorporación de múltiples plantas, así como también incluirlo como un proceso previo a la búsqueda de centros de recogida. En cuanto a los métodos de resolución, se quiere llegar a un modelo que permite resolver el problema directamente sin la necesidad de utilizar subrutinas que resuelvan parte del problema por separado. La incorporación de algoritmos genéticos para resolver el problema de recolección de leche con mezclas podrían ayudar a encontrar mejores soluciones, ya que, debido a que trabajan con una población de soluciones a la que se aplican sucesivos cruzamientos y mutaciones, permiten explorar en forma mucho más extensa el espacio de búsqueda. Por otro lado, el proceso de codificación y decodificación en estos algoritmos, permite trabajar mucho mejor con instancias grandes, como las utilizadas en el conjunto de instancias reales.

Bibliografía

- [1] Claudia Archetti and Maria Grazia Speranza. The split delivery vehicle routing problem: a survey. *The vehicle routing problem: Latest advances and new challenges*, pages 103–122, 2008.
- [2] Ph Augerat, Jose Manuel Belenguer, Enrique Benavent, A Corberán, D Naddef, and G Rinaldi. *Computational results with a branch and cut code for the capacitated vehicle routing problem*. IMAG, 1995.
- [3] Chuda Basnet, Les R Foulds, and John M Wilson. An exact algorithm for a milk tanker scheduling and sequencing problem. *Annals of Operations Research*, 86:559–568, 1999.
- [4] Gerald G Brown and Glenn W Graves. Real-time dispatch of petroleum tank trucks. *Management science*, 27(1):19–32, 1981.
- [5] Martin Butler, H Paul Williams, and Leslie-Ann Yarrow. The two-period travelling salesman problem applied to milk collection in ireland. *Computational Optimization and Applications*, 7(3):291–306, 1997.
- [6] Ann Melissa Campbell and Jill Hardin Wilson. Forty years of periodic vehicle routing. *Networks*, 63(1):2–15, 2014.
- [7] Massimiliano Caramia and Francesca Guerriero. A milk collection problem with incompatibility constraints. *Interfaces*, 40(2):130–143, 2010.
- [8] Jiumei Chen and Jing Shi. A multi-compartment vehicle routing problem with time windows for urban distribution—a comparison study on particle swarm optimization algorithms. *Computers & Industrial Engineering*, 133:95–106, 2019.
- [9] Lu Chen, Yang Liu, and André Langevin. A multi-compartment vehicle routing problem in cold-chain distribution. *Computers & Operations Research*, 2019.
- [10] GDH Claassen and Th HB Hendriks. An application of special ordered sets to a periodic milk collection problem. *European Journal of Operational Research*, 180(2):754–769, 2007.

- [11] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.
- [12] Benoit Crevier, Jean-François Cordeau, and Gilbert Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756–773, 2007.
- [13] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.
- [14] Iman Dayarian, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. A column generation approach for a multi-attribute vehicle routing problem. *European Journal of Operational Research*, 241(3):888–906, 2015.
- [15] AE Dooley, WJ Parker, and HT Blair. Modelling of transport costs and logistics for on-farm milk segregation in new zealand dairying. *Computers and electronics in agriculture*, 48(2):75–91, 2005.
- [16] Moshe Dror, Gilbert Laporte, and Pierre Trudeau. Vehicle routing with split deliveries. *Discrete Applied Mathematics*, 50(3):239–254, 1994.
- [17] Miguel A Figliozzi. An iterative route construction and improvement algorithm for the vehicle routing problem with soft and hard time windows. 2008.
- [18] Miguel Andres Figliozzi. An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows. *Transportation Research Part C: Emerging Technologies*, 18(5):668–679, 2010.
- [19] Marshall L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations research*, 42(4):626–642, 1994.
- [20] Bernhard Fleischmann. The vehicle routing problem with multiple use of vehicles. *Fachbereich Wirtschaftswissenschaften, Universität Hamburg*, 1990.
- [21] Michel Gendreau, Gilbert Laporte, Christophe Musaraganyi, and Éric D Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 26(12):1153–1173, 1999.
- [22] A Hoff and A Løkketangen. A tabu search approach for milk collection in western norway. In *6th Triennial Symposium on Transportation Analysis, TRISTAN, Pukkett, Thailand*, 2007.
- [23] Islem Kaabachi, Hiba Yahyaoui, Saoussen Krichen, and Abdelkader Dekdouk. Measuring and evaluating hybrid metaheuristics for solving the multi-compartment vehicle routing problem. *Measurement*, 141:407–419, 2019.

- [24] Nadia Lahrichi, Teodor Gabriel Crainic, Michel Gendreau, Walter Rei, and Louis-Martin Rousseau. Strategic analysis of the dairy transportation problem. *Journal of the Operational Research Society*, 66(1):44–56, 2015.
- [25] Emile Aarts Jan K. Lenstra. Local search in combinatorial optimization. In *Local Search in Combinatorial Optimization*, 1997.
- [26] Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.
- [27] Tsai-Yun Liao and Ta-Yin Hu. An object-oriented evaluation framework for dynamic vehicle routing problems under real-time information. *Expert Systems with Applications*, 38(10):12548–12558, 2011.
- [28] Shuguang Liu, Lei Lei, and Sunju Park. On the multi-product packing-delivery problem with a fixed route. *Transportation Research Part E: Logistics and Transportation Review*, 44(3):350–360, 2008.
- [29] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.
- [30] Jens Lygaard and Sanne Wøhlk. A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *European Journal of Operational Research*, 236(3):800–810, 2014.
- [31] Magalie Mourgaya and François Vanderbeck. The periodic vehicle routing problem: classification and heuristic. *Rairo-Operations Research*, 40(2):169–194, 2006.
- [32] Mohammad Kamran Mumtaz, MN Jalil, and Kamran Ali Chatha. Designing the milk collection network using integrated location routing approach. In *International Conference on Industrial Engineering and Operations Management*, 2014.
- [33] Luc Muyldermans and Gu Pang. On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research*, 206(1):93–103, 2010.
- [34] Germán Paredes-Belmar, Andrés Bronfman, Vladimir Marianov, and Guillermo Latorre-Núñez. Hazardous materials collection with multiple-product loading. *Journal of cleaner production*, 141:909–919, 2017.
- [35] Germán Paredes-Belmar, Armin Lüer-Villagra, Vladimir Marianov, Cristián E Cortés, and Andrés Bronfman. The milk collection problem with blending and collection points. *Computers and Electronics in Agriculture*, 134:109–123, 2017.
- [36] Germán Paredes-Belmar, Vladimir Marianov, Andrés Bronfman, Carlos Obreque, and Armin Lüer-Villagra. A milk collection problem with blending. *Transportation Research Part E: Logistics and Transportation Review*, 94:26–43, 2016.

- [37] Puca Huachi Vaz Penna, Anand Subramanian, and Luiz Satoru Ochi. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201–232, 2013.
- [38] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.
- [39] Victor Pillac, Christelle Guéret, and Andrés L Medaglia. An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, 54(1):414–423, 2012.
- [40] Lorena Pradenas, Boris Oportus, and Víctor Parada. Mitigation of greenhouse gas emissions in vehicle routing problems with backhauling. *Expert Systems with Applications*, 40(8):2985–2991, 2013.
- [41] Peerapon Prasertsri and Richard L Kilmer. Scheduling and routing milk from farm to processors by a cooperative. *Journal of Agribusiness*, 22(2):93S106, 2004.
- [42] Harilaos N Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of operations research*, 61(1):143–164, 1995.
- [43] Martin Reed, Alikí Yiannakou, and Roxanne Evering. An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing*, 15:169–176, 2014.
- [44] Gerhard Reinelt. TspLib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.
- [45] Jayaram K Sankaran and Rahul R Ubgade. Routing tankers for dairy milk pickup. *Interfaces*, 24(5):59–66, 1994.
- [46] Ahmet Şen and Kerem Bülbül. A survey on multi trip vehicle routing problem. *Sabancı University Research Database*, 2008.
- [47] Kanchana Sethanan and Rapeepan Pitakaso. Differential evolution algorithms for scheduling raw milk transportation. *Computers and Electronics in Agriculture*, 121:245–259, 2016.
- [48] Paulo Vitor Silvestrin and Marcus Ritt. An iterated tabu search for the multi-compartment vehicle routing problem. *Computers & Operations Research*, 81:192–202, 2017.
- [49] Marius M Solomon and Jacques Desrosiers. Survey paper—time window constrained routing and scheduling problems. *Transportation science*, 22(1):1–13, 1988.
- [50] Éric D Taillard. A heuristic column generation method for the heterogeneous fleet vrp. *RAIRO-Operations Research*, 33(1):1–14, 1999.

- [51] CD Tarantilis and CT Kiranoudis. A meta-heuristic algorithm for the efficient distribution of perishable foods. *Journal of food Engineering*, 50(1):1–9, 2001.
- [52] A Serdar Tasan and Mitsuo Gen. A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries. *Computers & Industrial Engineering*, 62(3):755–761, 2012.
- [53] Joao Teixeira, António Pais Antunes, and Jorge Pinho de Sousa. Recyclable waste collection planning—a case study. *European Journal of Operational Research*, 158(3):543–554, 2004.
- [54] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.