INF-295: Inteligencia Artificial
Informe final: Milk Collection with Blending

Anghelo Carvajal
201473062-4

11 de enero de 2021

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen
Este trabajo investigativo presenta el actual estado del arte del problema *Milk Collection with Blending*, el cual es una sub-variante de *Vehicle Routing Problem* (VRP). Este problema busca recolectar leche sin procesar de un conjunto de granjas, donde la mayor diferencia con VRP es que los distintos tipos de leche sin procesar pueden mezclarse entre si, lo cual produce una leche de calidad inferior. Se definirá el problema, mostrando variables y restricciones típicas, y se mostrará un modelo matemático que modela el problema de forma certera.

1. Introducción

El presente documento tiene como propósito presentar y definir el problema *Milk Collection with Blending*, además de documentar el actual estado del arte de dicho problema.

Se comenzará definiendo el problema actual, hablando de forma general de las variables y restricciones típicas de este problema. Luego se expondrá el actual estado del arte del problema *Milk Collection with Blending*, documentando sus orígenes (de que tipo de problema proviene), como han sido enfrentados dichos orígenes, los algoritmos que intentan resolverlo, y las variantes de este mismo problema.

También se presentará un modelo matemático de programación lineal que puede representar este problema de forma certera. Finalmente se expondrán las conclusiones de este trabajo investigativo.

2. Definición del Problema

El problema *Milk Collection with Blending* es un problema NP-duro, el cual surge en el año 2016 [8].

Este problema consiste en encontrar un conjunto de rutas óptimas para el recorrido de cada uno de los camiones que se tienen a disposición. Cada una de estas rutas debe proveer un camino para cada camión, de modo que este recoja toda la leche de cada una de las granjas que se le asigne y la lleve a la planta procesadora.

Cada ruta puede empezar en cualquier parte, pero siempre debe terminar en la planta procesadora. Hay un costo asociado al desplazamiento de un camión entre una granja y la otra.

La leche se categoriza en distintos tipos según su calidad. Estos tipos son ordenables de mejor a peor calidad, y las ganancias monetarias también son distintas según dicha calidad. La planta procesadora exige una cantidad mínima de cada tipo (calidad) de leche.

Los camiones pueden transportar una cantidad limitada y no tienen compartimientos separados para cada tipo de leche, por lo que si un camión recoge leches de distintas calidades de las granjas, estas se mezclan dentro del camión, resultando en leche que se considera de la peor calidad de la mezcla. La ventaja de esto es reducir el costo de movilización de los camiones a cambio de menores ganancias por la calidad de la leche.

Los parámetros del problema son los caminos entre las granjas de leche (representadas por un grafo), los caminos entre las granjas y la planta, y el costo de desplazar a un camión a través de cada uno de estos arcos. La cantidad de camiones de las que se dispone y la capacidad que cada uno puede transportar, las clasificaciones para los tipos de leche, que tipo y cuanta cantidad de leche produce cada granja. Cuanta cantidad de leche de cada tipo exige la planta procesadora en total. Granja en que cada camión inicia su ruta.

Las variable principal de este problema es el orden en el cual cada camión recorre sus granjas, lo cual implica que tipos de leche recogería dicho camión de cada granja, que tipo de leche resultante entrega el camión a la planta y en que cantidad.

Este problema se ve restringido por la capacidad máxima que tiene cada camión, que cada camión recolecte toda la leche de la granja a la que va a recolectar, que cada granja sea visitada por a lo más un camión, cada camión tiene a lo más 1 ruta, se debe respetar la cantidad de leche de cada tipo que exige la planta como mínimo.

El objetivo de este problema es maximizar los beneficios monetarios, disminuyendo los recorridos de los camiones y aumentando la ganancia producida por la leche recolectada según su tipo.

También existen otras variantes de este problema, como que cada camión pueda poseer distintos compartimientos para transportar la leche [4], de modo que 1.- no habría mezcla de leches o 2.- que se minimice la mezcla de tipos de leche; que existan puntos de recolección a los cuales las granjas acercan la leche [7]; o que algunas granjas no sean accesibles por grandes camiones [1].

3. Estado del Arte

Este problema es una variación del problema *Vehicle Routing Problem* (VRP), el cual fue documentado por primera vez en el año 1959 [2]. En dicho problema se discutía el encontrar un conjunto de rutas óptimas para una determinada cantidad de vehículos los cuales deben entregar paquetes a los respectivos clientes.

Específicamente este problema no ha sido tan trabajado e investigado debido a ser un problema no tan antiguo. Algunos de los problemas similares y métodos confeccionados para resolver esos y este problema son:

- El primer VRP fue realizado y documentado por Dantzig y Ramser en 1959 [2]. Este se basaba en un problema de distribución de combustible, y

apuntaba a repartir paquetes a clientes geográficamente separados usando un conjunto de vehículos. Aquí se concluye que este tipo de problema es NP-completo, debido a que podía ser reducido a un problema de vendedor viajero.

- A la fecha, la implementación que mejores resultados ha dado es la de Taillard [11], la cual se basa en algoritmos de búsqueda tabú, en donde distribuye a los nodos en dos formas distintas, uno de forma uniforme y la otra de forma no euclidiana.
- Otro tipo de problema basado en VRP es la variante con múltiples productos (MPVRP), el cual ha sido enfrentado con algoritmos genéticos [3], el algoritmo de Dijkstra [6] y modelos de programación lineal entera en conjunto a un solver [5] por nombrar algunos.
- Una variante similar a MPVRP es el problema de enrutamiento de vehículos con múltiples compartimientos (MCVRP). Un método notable es el de El Fallahi et. al. [4], dado que lo resuelven de tres formas distintas; con una heurística constructiva sin iteraciones de mejora, con busqueda tabú y con un algorítmo memético (el cual es a su vez una extensión del los algoritmos genéticos).
- La primera vez que se atacó de la recolección de leche como una variante específica de VRP fue en el año 1994 por Sankaran y Ubgade [9] para resolver un caso real de 70 granjas en India. Ellos consideraron camiones de diferentes capacidades y no poder exceder ciertos límites de tiempo entre cada recolección dado a las distintas condiciones climáticas de la zona.
- El problema de la recolección de leche que admite la mezcla de los distintos tipos de leche (nuestro problema) fue abordado por primera vez el año 2016 por Germán Paredes-Belmar et. al [8]. En dicha ocasión se formuló el problema a través de un modelo entero mixto. Para resolver instancias medianas usaba un algoritmo de bifurcación y corte. Para instancias más grandes se utilizaba un procedimiento heurístico, el cual consiste en dividir la instancia real en conjuntos de granjas, las cuales eran repartidas según su ubicación. En este estudio se discute el mezclar los tipos de leches vs no mezclarlos al transportarlos en los camiones, y concluía que mezclar las leches predominaba por sobre no mezclarlos, debido a la viabilidad y relajación que esta otorga. También estudia como afecta los camiones con múltiples compartimientos y con compartimiento único a esta variante.
- En el año 2017, los mismos autores enfrentan una variante del problema [7]. En este problema se tenía una cantidad aún mayor de granjas, por lo que el modelo anterior no podía encontrar una solución en tiempo razonable, por lo que agregaron puntos de recolección para cada conjunto de granjas, lo cual permitió encontrar soluciones en tiempos mucho más acotados.
- Villagran propone 2 acercamientos, basados técnicas de búsqueda local, para este problema, en el año 2019 [12]. Estos algoritmos eran basados en las técnicas *hill-climbing* e *iterated local search*, las cuales ambas entregaron soluciones de buena calidad, de modo que abre la puerta a la posibilidad de no usar técnicas completas para la resolución de este problema. En las pruebas del autor, estos algoritmos lograron obtener el óptimo global en la mayoría de instancias utilizadas. Cabe destacar que ambas implementaciones entregaban mejores resultados, tanto en calidad de la solución como en tiempo de ejecución, que los algoritmos del estado del arte hasta esa fecha.
- En el año 2019, Soto [10] propone un acercamiento basado en la meta-heurística *Simulated Annealing*, el cual tiene un muy buen desempeño en instancias pequeñas y medianas, comparables al solver CPLEX.

4. Modelo Matemático

A continuación se presenta un modelo de programación entera mixta, tal como es descrito en [12], el cual está basado en [8].

4.1. Función objetivo

Este problema tiene como objetivo la maximización del beneficio monetario, por lo que se considera la diferencia entre la ganancia producida por la leche recolectada vs los costos de transporte de las rutas construidas.

Esto se representa a través de:

$$Max \left\{ \sum_{t \in T} \sum_{r \in T} \alpha^r v^{tr} - \sum_{(i,j,k \in AK)} c_{ij}^k x_{ij}^k \right\} \quad (1)$$

4.2. Parámetros

Los parámetros de este problema son los siguientes:

- 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, $N = 0, ..., n$. 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.

4.3. Variables de decisión

Este modelo posee 5 variables de decisión. De estas, 3 son binarias:

- x_{ij}^k, la cual vale 1 si el camión k viaja directamente del nodo i al nodo j. Espacio de búsqueda: $2^{|N|*|N|*|K|}$
- y_i^{kt}, la cual vale 1 si el camión k recoge leche de calidad t de la granja i. Espacio de búsqueda: $2^{|N|*|K|*|T|}$
- z^{kt}, la cual vale 1 si el camión k entrega leche de calidad t a la planta. Espacio de búsqueda: $2^{|K|*|T|}$

Las otras 2 variables no binarias:

- w^{kt}, la cual indica el volumen de leche de calidad t que el camión k entrega a la planta. Espacio de búsqueda: $|Q|^{|K|*|T|}$
- v^{tr}, la cual indica el volumen de leche de calidad t entregada a la planta, mezclada para su uso como leche de calidad r. Espacio de búsqueda: $|Q|^{|T|*|T|}$

4.3.1. Espacio de búsqueda

El espacio de búsqueda es:

$$2^{|N|*|N|*|K|}*2^{|N|*|K|*|T|}*2^{|K|*|T|}*|Q|^{|K|*|T|}*|Q|^{|T|*|T|} \tag{2}$$

Lo cual puede ser reescrito como:

$$2^{|K|*(|N|^2+|N|*|T|+|T|)} * |Q|^{|T|*(|K|*|T|)} \tag{3}$$

4.4. Restricciones

Este modelo se ve restringido por las siguientes restricciones:

La restricción 4 limita la cantidad de leche que puede recolectar cada camión de acuerdo a su capacidad. Se considera una flota heterogénea.

$$\sum_{r \in T} \sum_{i \in N:(i,j) \in IT} q_i^t y_i^{kt} \leq Q^k, \forall k \in K \tag{4}$$

La restricción 5 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.

$$\sum_{k \in K_i} y^{kt} = 1, \forall i \in N, t \in T : (i, j) \in IT \tag{5}$$

La restricción 6 establece que cada camión debe tener como máximo una ruta la cual comienza desde la planta.

$$\sum_{j:(0_k,j,k) \in AK} x_{0_k j}^k \leq 1, \forall k \in K \tag{6}$$

La restricción 7 permite controlar el flujo para el orden de las visitas de los nodos por parte de cada camión.

$$\sum_{i:(i,j,k) \in AK} x_{ij}^k = \sum_{h:(j,h,k) \in AK} x_{jh}^k, \forall k \in K_j, j \in N_0 \tag{7}$$

La restricción 8 establece que cada camión que visita cada granja debe detenerse y recoger su leche.

$$\sum_{p:(p,i,k) \in AK} x_{pi}^k = y_i^{kt}, \forall k \in K_i, i \in N, t \in T : (i, t) \in IT \tag{8}$$

Las restricciones 9, 10, 11 y 12 establecen las reglas de mezcla de leche. La restricción 9 controla el tipo de leche de cada camión de acuerdo a cada una de las granjas que ha visitado. La restricción 10 controla que cada camión entrega en planta solo un tipo de leche. La restricción 11 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 12 mide la cantidad efectivamente recolectada de cada tipo de leche considerando las granjas visitadas por cada camión.

$$z^{kt} \leq 1 - \sum_{r \in D^t : r \neq t, (i,r) \in IT} y_i^{kr}, \forall k \in K_i, i \in N, t \in T \tag{9}$$

$$\sum_{t \in T} z^{kt} \leq 1, \forall k \in K \tag{10}$$

$$w^{kt} \leq z^{kt} Q^k, \forall k \in K, t \in T \tag{11}$$

$$w^{kt} \leq \sum_{r:t \in D^r} \sum_{h \in N^r} q_h^r y_h^{kr}, \forall k \in K, t \in T \tag{12}$$

La restricción 13 se encarga de forzar que cada camión se lleve toda la leche producida por cada granja a la planta.

$$\sum_{k \in K} \sum_{t \in T} w^{kt} = \sum_{(i,t) \in IT} q_i^t \tag{13}$$

La restricción 14 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.

$$\sum_{r \in D^t} v^{tr} = \sum_{k \in K} w^{kt}, \forall t \in T \tag{14}$$

La restricción 15 se encarga de controlar la satisfacción de las cuotas de leche de cada tipo.

$$\sum_{t \in T} v^{tr} \geq P^r, \forall r \in D^t \tag{15}$$

La restricción 16 evita mezclas de leche prohibidas.

$$y_i^{kt} + y_i^{kr} \leq 1, \forall (t,r) \in PM; (i,t), (j,t) \in IT \tag{16}$$

La restricción 17 evita la aparición de sub-ciclos en las rutas de cada camión.

$$\sum_{i \in S} \sum_{j \in S} x_{ij}^k \leq |S| - 1, \forall S \subseteq N, k \in K \tag{17}$$

Las restricciones 18, 19 y 20 controlan la naturaleza de las variables de decisión del modelo planteado. La restricción 18 controla la naturaleza binaria de las variables asociadas a los tipos de leche recolectados y mezclados en ruta. La restricción 19 controla la naturaleza de la variable binaria que controla las secuencias de visitas de los camiones. La restricción 20 controla la naturaleza no negativa de las variables de volúmenes de leche entregados y mezclados en planta.

$$y_i^{kt}, z^{kt} \in \{0,1\}, \forall i \in N, k \in K_i, t \in T : (i,t) \in IT \tag{18}$$

$$x_{ij}^k \in 0, 1, \forall (i,j,k) \in AK \tag{19}$$

$$w^{kt}, v^{tr} \geq 0, \forall k \in K; t, r \in T, r \in D^t \tag{20}$$

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

5. Representación

Como este problema tiene como objetivo encontrar rutas óptimas para una determinada cantidad de camiones, donde cada ruta consiste en el conjunto ordenado de granjas que recorrería el camión. Sabiendo esto, una buena representación matemática consiste en un "arreglo de arreglos" o "vector de vectores" (no confundir con matriz).

Cada uno de estos vectores interiores modelarían las rutas que se le asignarían a los camiones, de modo que cada uno de estos vectores interiores contendrían los identificadores de las granjas a recorrer que se están asignando a esta ruta, y en el orden en el que se deben recorrer. De esta forma, el vector exterior contendría el listado de rutas que se le asignarían a cada uno de los camiones.

Se usa esta representación debido a que se adapta bien al problema, ya que:

- Permite tener rutas de distintos con distintas cantidades de granjas a recorrer (a diferencia de una matriz de tamaño fijo).
- Como existen tantas rutas como camiones y cada camión está asignado a una ruta distinta, se maneja en parte la restricción 6.
- Al indicar simplemente que granjas debe recorrer en cada ruta y no almacenar más información, se fuerza a que el camión deba recoger toda la leche de cada granja que recorre, manejando la restricción 8 y 13 de forma implicita.
- Al ser vectores que contienen granjas que no se repiten dentro del mismo vector u en otras rutas, se imposibilita que se generen sub-ciclos, por ende manejando la restricción 17.

A nivel de implementación, se utilizaron clases que ayudan a cuidar la lógica interna y el estado de las variables. La solución final se ve representada por la clase `Solution`, la cual maneja las rutas usando un `std::vector<Route>`.

La clase `Route` modela una ruta asignada a un camión y que transporta un tipo determinado de calidad de leche, además de manejar las granjas a recorrer por dicho camión usando un `std::vector<Node*>`. Se prefirió usar punteros a `Node` en lugar de usar la clase directamente para evitar la reinstanciación al mover una granja de una ruta a otra.

La clase `Node` simplemente contiene los datos de la granja que representa, tal como el identificador, la posición, la cantidad de leche que produce, entre otros.

Si se ignora la abstración de las clases antes mencionadas, esta implementación se puede interpretar como `std::vector< std::vector<int> >`, lo cual es

muy similar a la representación matemática antes expuesta.

Cabe destacar que una `Route` no incluye la planta procesadora, ya que un camión siempre debe comenzar y terminar su recorrido en ella, por lo que almacenarla de forma explicita sería un desperdicio de memoria y complicaría los movimientos a realizar. Además, al modelarlo de esta manera, se maneja en parte la restricción 6.

6. Descripción del algoritmo

El algoritmo propuesto para resolver el problema es Hill Climbing. Dado que esta es una técnica incompleta, se separa en los pasos de la generación de solución inicial, función de evaluación, el mismo hill climbing y los movimientos a realizar. Los movimientos planteados son del tipo “Alguna mejora”.

Cabe destacar que por como está construido el algoritmo para la generación de la solución inicial, este puede entregar soluciones infactibles como soluciones iniciales. Dado esto, los movimientos están planteados de modo que intentan priorizar el encontrar soluciones "menos infactibles"si la solución con la que están trabajando es infactible (a pesar de que puedan disminuir la calidad de la solución), y en segundo lugar intentan mejorar la calidad de la solución. Si la solución entregada a un movimiento ya es factible, entonces solo se preocupa de mejorar la calidad de la solución.

6.1. Generación de solución inicial

La generación de la solución inicial consiste en una asignación básica de granjas a los camiones disponibles, llamándole rutas a este conjunto.

Como primer paso, se generan tantas rutas como tipos de leche hay disponible, y a cada una de estas rutas se les asigna un tipo de leche distinto (líneas 4 a 8).

Luego se asignan de forma aleatoria los camiones a estas rutas (líneas 10 a 23). Si durante la asignación aleatoria se intenta asignar un camión sin la suficiente capacidad para suplir la cuota del tipo de leche asociado a la ruta, se reinicia la asignación de camiones (Líneas 11 y 16).

Finalmente, se seleccionan las granjas de forma aleatoria y se intentan agregar a las rutas con el tipo de leche asignado (Líneas 25 a 36).

```
1  function initialSolution(instance):
2      solution ← vector vacio de rutas
3
4      for milk in instance.milktypes in random order do
5          route ← ruta vacia
6          assign milk to route
7          add route to solution
8      end for
9
10     valid ← false
11     while not valid do
12         valid ← true
13         for index in solution in random index order do
14             route ← solution[index]
15             truck ← instance.trucks[index]
16             if truck.capacity < route.quota do
17                 valid ← false
18                 break
19             end if
20
21             assign truck to route
22         end for
23     end while
24
25     farms ← instance.farms
26     while farms.length > 0 do
27         selected_farm ← select random from farms
28         for route in solution do
29             if route.milk_type != selected_farm.milk_type do
30                 continue
31             end if
32             remove selected_farm from farms
```

```
            add selected_farm to route
            selected_farm ← select random from farms
        end for
    end while
    return solution
end function
```

6.2. Función de evaluación

La función de evaluación de la solución consiste simplemente en un ciclo que itera por cada una de las rutas y suma la calidad de cada ruta.

```
1 function evaluateSolution(solution, instance):
2     result ← 0
3     for route in solution do
4         result += evaluateRoute(route, instance.initialNode)
5     end for
6     return result
7 end function
```

Por otro lado, la función de evaluación de la ruta es levemente más compleja. Consiste en sumar la leche producida de cada granja (Línea 7) y calcular la distancia total recorrida por el camión (Líneas 5, 8, 9 y 11). Luego se multiplica la cantidad de leche transportada por la ganancia del tipo de leche final y se le descuenta la distancia recorrida (Líneas 13 y 14). Además se penaliza la calidad de la ruta en caso de que se sobrepase la capacidad del camión (Lineas 15 a 17) y/o que no se alcance la cuota de este tipo de leche (Líneas 18 a 20).

```
1 function evaluateRoute(route, initial_node):
2     milk_total ← 0
3     total_distance ← 0
4
5     previous_farm ← initial_node
6     for farm in route do
7         milk_total += farm.produced
8         total_distance += distance(previous_farm, farm)
9         previous_farm ← farm
10    end for
11    total_distance += distance(previous_farm, initial_node)
12
13    quality ← milk_total * milk_profit_percentage
14    quality -= total_distance
15    if milk_total > capacity do
16        quality -= milk_total - capacity
17    end if
18    if milk_total < quota do
19        quality -= quota - milk_total
20    end if
21
22    return quality
23 end function
```

6.3. Hill climbing

El algoritmo principal de hill climbing está levemente modificado para poder realizar más de un único tipo de movimiento durante la ejecución. Estos movimientos se explican en las siguientes subsecciones.

Se define una variable `movements`, la cual contiene los movimientos posibles a realizar (4 especificamente) y se copia la solución inicial a una nueva variable `solution`.

Se usa un ciclo `while` para controlar la cantidad de iteraciones que se realizan y que no se sobrepase el límite engregado como parámetro. De esta forma se define el primer criterio de parada del algoritmo. El segundo criterio de parada es en caso de que se detecte que la solución llegó a un máximo local, si es así se detiene la ejecución y se entrega la solución final. Esto se controla a traves de la variable `is_better_solution`.

Como se tiene más de un posible movimiento a realizar por iteración, se decide de forma aleatoria cual movimiento realizar en cada iteración (Línea 10 y 11).

Cuando las funciones que ejecutan los movimientos reciben la encuentran cualquier mejor solución que la actual, estas retornan `true` y guardan dicha solución mejor en la solución que recibieron por parámetro. Si

recorrieron todo el vecindario sin encontrar una solución mejor, retornan `false` y la solución que recibieron por parámetro queda exactamente igual que como la recibieron.

Se ejecutan los movimientos en algún orden aleatorio sobre la solución hasta que esta mejore. Cuando mejora, se termina el orden actual de movimientos y se genera uno nuevo (Línea 14 a 16). Si se aplican todos los movimientos disponibles sobre la solución actual y ninguno logra mejorar la solución, se entiende que se encontró el óptimo local y se retorna la solución.

```
1  function hillClimbing(initial_solution, instance, K):
2      solution ← initial_solution
3      movements ← lista de movimientos
4      i ← 0
5      is_better_solution ← true
6      while i < K && is_better_solution do
7          is_better_solution ← false
8          current_quality = evaluateSolution(solution, instance);
9
10         for movement in movements in random order do
11             is_better_solution ← movement(solution, instance, current_quality)
12
13             i += 1
14             if is_better_solution do
15                 break
16             end if
17         end for
18     end while
19
20     return solution
21 end function
```

A continuación se explican los 4 movimientos existentes.

6.3.1. Move node between routes

Este movimiento consiste en seleccionar un nodo de una ruta (Línea 3 y 9), removerlo de esta ruta (Línea 18) e insertarlo en una posición aleatoria de cualquier otra ruta (Línea 20 y 25).

Además, se realizan algunas verificaciones antes de mover al nodo al nuevo destino para así evitar que la ruta se vuelva infactible. En la línea 10 se verifica que la elcamión asignado a la ruta en la que se va a insertar el nodo actual tenga capacidad suficiente para recibir este nodo. Las funciones `canRemoveFarm` (Línea 14) y `canAddFarm` (Línea 21) verifican si la ruta se volvería infactible al quitar o agregar el nodo a la ruta respectivamente.

Después de haber movido el nodo de ruta se revisa si la solución era infactible, y de ser así se verifica si la solución dejó de ser "menos infactible", ya sea porque se sobrecarga menos el camión (`didCapacitiesLeftImproved`, Línea 28) o porque se está alcanzando el mínimo de la cuota pedido por el tipo de leche (`didQuotasDiffImproved`, Línea 29), si alguno de esos se cumple, se entiende que la solución es mejor y se retorna `true`.

Si la solución es actualmente factible o si no se mejoró la factibilidad de la solución al realizar este movimiento, se calcula la nueva calidad de la solución y si es mejor que la anterior se retorna `true` (Líneas 35 a 38).

Si la solución no era mejor de ninguna forma al insertar el nodo, se remueve el nodo inserado para poder intentar insertarlo en otra posición de esta ruta (Línea 40).

Si no hay posición en esta ruta que al insertar el nodo mejore la solución, se agrega a su posición original de la ruta original para volver a empezar con otro nodo (Línea 43).

Deshacer los cambios si no se encuentra una mejor solución permite que no sea necesario generar toda la vecindad desde el principio, si no que se vaya construyendo paso a paso mientras se requiere, ahorrando memoria y ciclos del procesador.

Si se probaron todas las posibilidades que entrega este movimiento (es decir, se recorrió todo el vecindario) y ninguna mejoró la solución actual, se retorna `false`.

```
1  function moveNodeBetweenRoutes(solution, instance, old_quality):
2      was_feasible ← isFeasible(solution)
3      for src_route in solution in random order do
4          for dst_route in solution in random order do
5              if src_route == dst_route do
6                  continue
7              end if
8
9              for src_farm in src_route in random order do
```

```
            if dst_route.capacity_left < src_farm.produced do
                continue
            end if

            if not canRemoveFarm(src_route, src_farm) do
                continue
            end if

            remove src_farm from src_route

            for dst_farm in dst_route in random order do
                if not canAddFarm(dst_route, src_farm) do
                    continue
                end if

                add src_farm to dst_route

                if not was_feasible do
                    capacities_improved ← didCapacitiesLeftImproved(solution)
                    quotas_improved ← didQuotasDiffImproved(solution)
                    if capacities_improved || quotas_improved do
                        return true
                    end if
                end if

                new_quality ← evaluateSolution(solution, instance)
                if new_quality > old_quality do
                    return true
                end if

                remove src_farm from dst_route
            end for

            add src_farm to src_route
        end for
    end for
end for

return false
end function
```

6.3.2. 2opt intra-route

Este movimiento consiste en un movimiento 2-opt sobre alguna ruta, hasta encontrar alguna que mejore la solución.

Los valores de retorno son los mismos que los otros movimientos.

Este movimiento no considera mejorar la factibilidad de la solución, ya que al invertir el orden de un subconjunto de nodos no afecta a ninguna restricción.

```
function intraRoute2Opt(solution, instance, old_quality):
    for route in solution in random order do
        for left_farm_index in route in random index order do
            for right_farm_index in route[left_farm+1:] in random order do
                reverseOrder(route, left_farm_index, right_farm_index)

                new_quality ← evaluateSolution(solution, instance)
                if new_quality > old_quality do
                    return true
                end if

                reverseOrder(route, left_farm_index, right_farm_index)
            end for
        end for
    end for
```

```
    return false
end function
```

6.3.3. Remove node

Este movimiento consiste en eliminar algún nodo cualquiera de cualquier ruta y ver si mejora la solución.

Se confeccionó este movimiento debido a que se notó que durante la fase de pruebas del algoritmo muchas soluciones finales eran infactibles. Más especificamente, las soluciones proponían rutas que saturaban la capacidad de los camiones, principalmente rutas de la peor calidad. Como al mover una granja de mala calidad a una ruta de mejor calidad la empeora, no era posible quitar deshacerse de los nodos que saturaban al camión. Este movimiento intenta solucionar ese problema.

```
function removeNode(solution, instance, old_quality):
    was_feasible ← isFeasible(solution)
    for route in solution in random order do
        for farm in route in random order do
            if not canRemoveFarm(route, farm) do
                continue
            end if

            remove farm from route

            if not was_feasible do
                capacities_improved ← didCapacitiesLeftImproved(solution)
                quotas_improved ← didQuotasDiffImproved(solution)
                if capacities_improved || quotas_improved do
                    return true
                end if
            end if

            new_quality ← evaluateSolution(solution, instance)
            if new_quality > old_quality do
                return true
            end if

            add farm to route
        end for
    end for

    return false
end function
```

6.3.4. Interchange node between routes

Finalmente, este movimiento consiste en tomar 1 nodo de dos rutas distintas e intercambiarlos.

Este movimiento fue planteado debido a que algunas soluciones entregadas por el programa podían mejorar al intercambiar un par de nodos en la misma iteración, en vez de mover un nodo de una ruta a otra en la primera iteración y luego lo mismo en la siguiente. Esto ocurre ya que hay mayor flexibilidad al realizar el movimiento, debido a que hay menos probabilidad de que alguna de las rutas se vuelvan infactibles al hacer este cambio.

```
function interchangeNodeBetweenRoutes(solution, instance, old_quality):
    was_feasible ← isFeasible(solution)
    for src_route_index in solution in random index order do
        src_route ← solution[src_route_index]

        for src_farm_index in src_route in random index order do
            src_farm ← src_route[src_farm_index]

            for dst_route in solution[src_route_index+1:] in random order do
                if not isMilkTypeCompatible(dst_route, src_farm) do
                    continue
                end if

                for dst_farm_index in dst_route in random index order do
```

```
                dst_farm ← dst_route[dst_farm_index]
                if not isMilkTypeCompatible(src_route, dst_farm) do
                    continue
                end if

                src_route[src_farm_index] ← dst_farm
                dst_route[dst_farm_index] ← src_farm

                if not was_feasible do
                    capacities_improved ← didCapacitiesLeftImproved(solution)
                    quotas_improved ← didQuotasDiffImproved(solution)
                    if capacities_improved || quotas_improved do
                        return true
                    end if
                end if

                new_quality ← evaluateSolution(solution, instance)
                if new_quality > old_quality do
                    return true
                end if

                src_route[src_farm_index] ← src_farm
                dst_route[dst_farm_index] ← dst_farm
            end for
        end for
    end for
end for

return false
end function
```

7. Experimentos

El algoritmo fue probado con 12 instancias[1] de distintos tamaños. Además, se le agregó colores al output final del programa para poder saber más fácilmente si el resultado final es factible o no y para saber que calidad de leche produce cada granja asignada a cada ruta.

Si la solución final es infactible, se muestra en rojo el beneficio total de esta solución y la letra de la ruta infactible. Si la solución no es factible debido a que una ruta no cumple con la cuota mínima de ese tipo de leche, se muestra en rojo el número que indica la cantidad total de leche que se está llevando de este tipo de esa ruta. Además, se le asignó un color a las granjas dependiendo del tipo de leche que producen, si la granja es de tipo 'A' se mostrará en color verde; si es de tipo 'B' se mostrará en color amarillo y así sucesivamente.

Esta metodología permitió detectar fácilmente que se generaban soluciones infactibles, analizar los datos y confeccionar nuevos movimientos y modificar los ya existentes para mejorar la factibilidad de las soluciones.

El entorno de experimentación consiste en un equipo con procesador Intel Core i5-6400 2.70GHz, con 24GB de memoria RAM a 2133MHz y un disco de duro de 2TB y 5400rpm. El equipo cuenta con Pop!_OS 20.10.

El algoritmo requiere que se le entregue como parámetro los datos de la instancia y el máximo de iteraciones a realizar (llámese K).

Para poder ejecutar el programa, se le debe entregar como parámetro la ruta a un archivo que contiene los datos de la instancia y el número entero K.

Este archivo indíca la siguiente información:

- La cantidad de camiones y la capacidad de cada uno.
- La cantidad de tipos de leche, la cuota mínima que se requiere de cada tipo y la ganancia que aporta cada uno.
- La cantidad de granjas y la información asociada a cada uno. Se indica un identificador numérico, posiciones x e y, una letra indicando la calidad de esta y la cantidad de leche producida. Cabe destacar que el primero de esta lista no es una granja, si no que se refiere a la planta procesadora.

El programa usa los datos de este archivo como parámetro para el algoritmo.

Se utilizaron 12 instancias distintas, las cuales todas tenían 3 camiones y 3 tipos de leche en cada instancia (variando la cuota pedida y la capacidad de los camiones). La diferencia principal de las instancias son en la cantidad de nodos, variando desde 22 hasta 80 nodos, donde las instancias con mayor cantidad

[1] https://github.com/AngheloAlf/2020-2_IA_Milk_collection_problem_with_blending/tree/master/instances

de nodos requerían una mayor cantidad de iteraciones para terminar.

El algoritmo tiene 2 criterios de término. El primero es si la cantidad de iteraciones sobrepasa al parámetro K, entendiendose una iteración como la realización de alguno de los movimientos propuestos en la sección anterior. El segundo criterio de parada consiste en la detección de un óptimo local, de modo que ninguno de los movimientos propuestos puede mejorar la calidad de la solución.

Todas las instancias fueron probadas repetidas veces con un $K = 1000$, pero ninguna ejecución sobrepasó las 500 iteraciones. Normalmente oscilaban entre las 60 (instancias pequeñas) y las 460 iteraciones (instancias más grandes).

8. Resultados

A continuación se peresentan los mejores resultados obtenidos con cada una de las instancias. Todas fueron ejecutadas con un $K = 1000$, pero ninguna requirió de todas esas iteraciones.

El formato de salida es el siguiente:

- Primera línea:
  - Calidad de la solución.
  - Costos totales del transporte.
  - Ganancia bruta según el tipo de leche.
- N líneas que muestran la información de las N rutas que fueron asignadas a los camiones (mostradas según el orden de los camiones).
  - Granjas asignadas a esta ruta, en el orden en el que se deben recorrer. También se muestra al inicio y al final el nodo correspondiente a la planta procesadora.
  - Costo de transporte de esta ruta.
  - Cantidad de leche transportada en esta ruta.
  - Calidad de la leche resultante.
- Finalmente, se muestra el tiempo total de la ejecución, medido en milisegundos.

```
instances/a36.txt

27903.57 1076.43 28980.00

01-27-21-15-03-36-24-20-06-12-33-30-18-01  391.60 14400 B
01-17-02-23-14-32-26-11-08-35-29-01        361.09 14700 A
01-16-09-07-04-13-10-28-25-22-34-31-19-01  323.75 14000 C

Tiempo total de ejecución: 1.545[ms]
```

```
instances/a44.txt

39025.97 1264.03 40290.00

01-32-08-29-20-35-05-26-23-14-44-41-02-11-38-17-01  475.50 24400 A
01-09-06-33-12-27-18-21-36-24-30-39-03-42-15-01     405.01 17000 B
01-37-19-31-04-13-40-16-28-25-34-43-01              383.53 13300 C

Tiempo total de ejecución: 9.846[ms]
```

```
instances/a55.txt

24563.85 1206.15 25770.00

01-26-05-08-32-47-35-38-11-17-50-44-02-53-29-23-20-14-01        403.65 11750 A
01-42-30-51-24-45-36-48-39-41-54-06-33-09-27-21-15-18-03-12-01  372.85 12400 B
01-22-13-37-43-31-28-55-25-49-19-07-46-10-40-16-52-04-34-01     429.64 17800 C

Tiempo total de ejecución: 21.602[ms]
```

```
instances/a64.txt

23090.88 1194.12 24285.00

01-11-53-14-62-47-59-05-32-56-50-29-44-23-26-20-35-38-01                        399.52 10900 A
01-57-43-25-61-48-64-03-55-06-40-45-31-04-52-21-49-34-46-16-58-18-02-13-19-07-01  434.56 20000 C
01-30-39-12-54-63-15-36-27-41-51-60-09-33-24-01                                 360.04 10550 B

Tiempo total de ejecución: 8.640[ms]
```

```
instances/a80.txt

29392.65 1557.35 30950.00

01-50-74-14-41-02-08-32-26-47-65-71-68-59-77-23-56-20-62-17-44-38-35-80-29-53-11-01  491.97 16000 A
01-54-75-30-18-60-06-45-24-63-72-12-03-09-69-21-27-36-57-42-39-51-33-05-01           509.80 14500 B
01-43-37-67-73-46-10-70-48-76-28-61-40-04-78-13-22-64-49-19-25-07-31-79-58-01        555.58 16000 C

Tiempo total de ejecución: 22.863[ms]
```

```
instances/c50.txt

65425.98 734.02 66160.00

01-41-11-32-26-50-17-02-44-23-29-05-38-14-20-08-35-47-01  257.64 33100 A
01-27-18-33-51-45-24-42-03-30-06-48-21-09-15-12-39-36-01  252.40 34500 B
01-13-40-10-04-19-25-43-34-07-31-22-37-49-16-28-46-01     223.97 29700 C

Tiempo total de ejecución: 7.205[ms]
```

```
instances/c75.txt

42135.53 999.47 43135.00

01-41-08-59-26-56-17-74-29-05-68-35-53-20-38-62-23-65-02-44-01                  333.93 18100 A
01-27-18-45-33-51-50-24-42-63-75-03-69-30-48-72-71-06-14-47-09-36-54-15-12-66-32-11-39-01  324.55 25050 B
01-76-28-46-31-49-22-70-37-61-21-16-58-55-60-67-13-52-07-34-64-43-57-25-19-04-10-40-01     340.98 25000 C

Tiempo total de ejecución: 27.682[ms]
```

```
instances/eil22.txt

15911.27 578.73 16490.00

01-13-04-07-10-16-22-19-01  214.48  5500 C
01-15-21-18-12-09-06-03-01  186.81  7200 B
01-11-08-02-05-14-20-17-01  177.44  9800 A

Tiempo total de ejecución: 0.517[ms]
```

```
instances/eil23.txt

6878.71 797.59 7676.30

01-20-02-14-11-05-08-01  269.44  5925 A
01-09-06-03-17-21-15-01  304.78  1634 B
01-13-10-12-18-16-01     223.37  2025 C

Tiempo total de ejecución: 0.414[ms]
```

```
instances/eil30.txt

4492.09 760.41 5252.50

01-23-07-25-19-16-13-10-15-22-01  251.60  3525 C
01-21-05-06-30-24-12-01           225.13  2850 B
01-11-08-14-17-20-02-26-29-01     283.68  2200 A

Tiempo total de ejecución: 0.499[ms]
```

```
instances/eil33.txt

20383.95 943.05 21327.00

01-31-16-28-25-22-19-10-07-13-04-01     276.68  6480 C
01-03-12-06-33-09-21-24-18-15-30-27-01  325.98 11690 B
01-05-08-11-20-23-26-17-29-32-14-02-01  340.39 11200 A

Tiempo total de ejecución: 1.751[ms]
```

```
instances/eil76.txt

90825.64 964.36 91790.00

01-68-35-47-53-05-38-71-14-20-08-59-11-32-56-26-41-50-17-74-02-65-23-62-29-01        326.35 45000 A
01-13-76-07-34-43-64-52-04-25-19-10-40-67-55-28-46-58-16-61-37-70-22-49-31-01        332.02 42800 C
01-27-54-39-66-12-60-15-36-09-30-06-21-72-48-75-69-03-63-44-42-57-24-18-45-51-33-01  306.00 48500 B

Tiempo total de ejecución: 18.349[ms]
```

9. Conclusiones

Como cualquier otro problema NP-duro, este problema CSOP ha demostrado ser difícil de modelar de modo que entregue una solución óptima en un tiempo prudente.

Debido a que es un problema bastante joven (aunque VRP data de 1959 [2], esta variante especifica se puede encontrar en la literatura desde el año 2016 [8]) se han desarrollado escasos modelos y métodos de resolución para este problema específico.

De todas formas, las técnicas que se han usado para trabajar *Milk Collection with Blending* suelen atacar al mismo problema base, pero con leves diferencias (como los tamaños de las instancias, o la opción de tener múltiples compartimientos en los camiones). Se han usado tanto técnicas completas, las cuales han demostrado ser viables para instancias pequeñas a medianas, algoritmos basados en técnicas completas y técnicas incompletas, como lo son *hill-climbing* e *iterated local search*. Estás últimas han demostrado mejores resultados que las técnicas completas

Una variante a considerar para investigaciones futuras podría ser la incorporación de múltiples plantas de procesado de leche en lugar de una única planta central.

El algoritmo propuesto entregar soluciones de calidad en un tiempo más que razonable con las instancias utilizadas.

Durante el periodo de pruebas, ninguna de las instancias requirió más de 500 iteraciones para encontrar el óptimo local. Esto lleva a la sospecha de que el algoritmo de hill climbing se está estancando demasiado rápido en óptimos locales. Este estancamiento se puede explicar por la naturaleza inherente a este algoritmo, debido a que este está diseñado de modo que solo explota la solución con la que está trabajando, casi sin explorar otras posibilidades.

10. Bibliografía

Referencias

[1] Massimiliano Caramia and Francesca Guerriero. A milk collection problem with incompatibility constraints. *Interfaces*, 40(2):130–143, 2010.

[2] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 10 1959.

[3] Dooley, AE, 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.

[4] El Fallahi, Abdellah, Christian Prins, and Roberto Wolfler Calvo. A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Computers & Operations Research*, 35(5):1725–1741, 2008.

[5] Leonardo Junqueira, José F. Oliveira, Maria Antónia Carravilla, and Reinaldo Morabito. An optimization model for the vehicle routing problem with practical three-dimensional loading constraints. *nternational Transactions in Operational Research*, 20(5):645–666, 2013.

[6] Shuguang Liu, Lei Le, 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.

[7] Germán Paredes Belmar, Armin Lüer-Villagra, Vladimir Marianov, Cristián Cortés, and Andrés Bronfman. The milk collection problem with blending and collection points. *Computers and Electronics in Agriculture*, 134:109–123, 01 2017.

[8] 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, 10 2016.

[9] Jayaram K Sankaran and Rahul R Ubgade. Routing tankers for dairy milk pickup. *Interfaces*, 24(5):59–66, 1994.

[10] Constanza Andrea Soto Caviedes. Un acercamiento meta-heurÍstico para el problema de recolecciÓn de leche con selecciÓn y mezcla. *Memoria de titulación, UTFSM*, 10 2019.

[11] Éric Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[12] JORGE ANDRÉS VILLAGRÁN MUÑOZ. Acercamientos basados en bÚsqueda local para el problema de recolecciÓn de leche con mezcla. *Memoria de titulación, UTFSM*, 10 2019.