Informe Final Taller 5: Optimización de Recorridos Utilizando Colonias de Agentes Cooperativos

VRP-mTW – Ant System

Inés Corrente, Sabrina Juárez, Verónica Varela

Tutores: Dr.Ing. Héctor Cancela, M.Sc.Ing. Omar Viera

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay.

Junio de 2001

Resumen

En el presente trabajo abordamos la resolución del problema de Ruteo de Vehículos con múltiples Ventanas de Tiempo (VRP-mTW), utilizando el método Ant System. Dicho problema consiste en un conjunto de clientes que presentan cierta disponibilidad horaria (ventanas de tiempo), y una demanda de bienes o servicios, que les debe ser entregada por un conjunto de vehículos desde un depósito central. El objetivo es encontrar un camino que visite cada cliente exactamente una vez, satisfaciendo su demanda, sin violar las restricciones de tiempo y capacidad, y minimizando el costo de traslado.
Ant System es un algoritmo de propósito general inspirado en el estudio del comportamiento de las colonias de hormigas. Se basa en una búsqueda cooperativa que es aplicable a la solución de problemas de optimización combinatoria. En la investigación que hemos realizado en la literatura disponible para este método, no se encontró ninguna aplicación para la resolución del problema VRP-mTW.
Hemos desarrollado un algoritmo Ant System mejorado, complementado con una post-optimización, que resuelve el problema mencionado, con resultados comparables con los mejores conocidos para el conjunto de casos de prueba comúnmente utilizado para testear este tipo de problemas (set de 56 problemas de Solomon).

Keywords: ruteo, VRP, optimización, Ant System.

INDICE

1. INTRODUCCION

Este Taller está enfocado a la resolución del problema de Ruteo de Vehículos con múltiples Ventanas de Tiempo (VRP-mTW). Dicho problema se puede describir como sigue: un conjunto de clientes geográficamente dispersos deben ser atendidos por una flota de vehículos inicialmente ubicada en un depósito dado. Cada cliente tiene una demanda y una o más ventanas de tiempo (time windows). Las ventanas de tiempo se definen mediante una hora de apertura y una hora de cierre, y representan el período de tiempo dentro del cual el cliente está disponible. Los clientes son atendidos por vehículos de capacidades limitadas y diferentes, que les entregarán el total de la demanda solicitada dentro de alguna de sus ventanas de tiempo. El objetivo es encontrar un camino que visite cada cliente exactamente una vez, satisfaciendo su demanda, sin violar las restricciones de tiempo y capacidad, que minimice el costo de traslado.

El problema VRP-mTW es un problema de tipo NP-Completo (ref.[43]), y por lo tanto su solución matemática exacta, es decir, su valor óptimo, es extremadamente costoso de hallar (podría llevar años de tiempo de CPU). Se han desarrollado diversas heurísticas que permiten resolver este tipo de problemas en forma aproximada (en ciertos casos alcanzando incluso el valor óptimo), a un costo razonable. Un conjunto de estas heurísticas son los Ant Algorithms, una familia de algoritmos basados en el comportamiento de las colonias de hormigas, que analizamos y aplicamos al problema VRP-mTW en este Taller.
El comportamiento de las colonias de hormigas y la forma en que combinan actividades complejas como la construcción de hormigueros y búsqueda de alimentos ha fascinado desde hace mucho tiempo a los investigadores de etología y comportamiento animal, quienes han propuesto varios modelos para explicar esas habilidades. Recientemente se han desarrollado algoritmos inspirados en el comportamiento de colonias de hormigas y se han aplicado a la resolución de muchos problemas de optimización, como por ejemplo el Problema del Viajero (Travelling Salesman Problem - TSP) o el Problema de Asignación Cuadrática (Quadratic Assignment Problem - QAP, ver [20]). Este nuevo método de optimización se conoce como Ant Colony Optimization (ACO).
El primer algoritmo ACO fue Ant System, presentado por Colorni, Dorigo y Maniezzo en 1991/1992 para la resolución del TSP (ver [1], [2] y [3]). Desde entonces se han agregado mejoras a ese primer algoritmo, aplicando los nuevos métodos ACO a varios problemas de optimización con muy buenos resultados.
En 1997 Bullnheimer, Hartl, Strauss presentaron una solución al problema VRP usando la variante Ant Colony System (ver [12] y [13]) de ACO. La solución al problema de VRP con una única ventana de tiempo fue propuesta por Gambardella, Taillard y Agazzi en el año 1999 (ver referencia [18]). No hay experiencias anteriores de aplicación de ACO para la resolución del problema VRP-mTW.

En este Taller hemos desarrollado e implementado un formalismo ACO que resuelve el problema VRP-mTW. Agregamos a este formalismo, una post-optimización de soluciones, que consiste de dos algoritmos: uno que se encarga de re-ordenar la solución obtenida buscando una de mejor costo, y otro que se encarga de reducir las esperas (y por lo tanto reducir el costo).
Se realizó una validación completa de la solución implementada, constatando que la utilización de post-optimización ayuda a mejorar los costos de las soluciones encontradas por el formalismo. La comparación realizada con las soluciones de otros

métodos, nos ha permitido concluir que nuestro algoritmo obtiene buenos resultados para este problema. En particular, se lograron disminuir los tiempos de espera en una buena proporción, lo que ocasionó que en promedio los costos fueran mejores que los obtenidos mediante los algoritmos comparados.

Este informe se organiza de la siguiente manera. En la Sección 2 describimos el problema VRP-mTW y lo formulamos matemáticamente. En la Sección 3 presentamos Ant System que es el método a utilizar para resolver nuestro problema. En la Sección 4 estudiamos la post-optimización de las soluciones obtenidas. En la Sección 5 proponemos una solución al problema VRP-mTW. En la Sección 6 presentamos el diseño y la implementación de esa solución, mientras que en la Sección 7 realizamos su validación. En la Sección 8 exponemos las conclusiones y posibles trabajos futuros, y por último en la Sección 9 la bibliografía utilizada.
En el Apéndice A presentamos el Estado del Arte de la metodología utilizada; en el Apéndice B los seudocódigos de algoritmos y funciones implementadas; en el Apéndice C los instructivos para la ejecución de los programas desarrollados; y por último en los Apéndices D y E estudiamos en detalle dos posibles trabajos futuros a realizar sobre nuestra solución.

2. DESCRIPCIÓN DEL PROBLEMA VRP-mTW

A continuación presentamos una formulación matemática del problema VRP-mTW. La definición para el problema básico se tomó del paper "Applying the Ant System to the Vehicle Routing Problem" de Bullnheimer, Hartl, Strauss (ref. [12]), adaptándola de acuerdo a las características y restricciones propias de nuestro problema.

Tenemos un conjunto de clientes $C = \{1,...,n_c\}$ a quienes le serán entregados bienes desde un depósito. Definimos $N = C \cup \{0, n_c+1\}$ el conjunto de localidades, donde 0 y n_c+1 representan el depósito.
Asociado a cada cliente i hay una demanda positiva q_i y un tiempo de servicio s_i. Además para cada cliente i tenemos w_i ventanas de tiempo, representadas como:
$W = \{(e_{i,1}, l_{i,1}). . (e_{i,wi}, l_{i,wi})\} = \{(e_{i,j}, l_{i,j}) / j = 1..w_i\}$, siendo $e_{i,j}$ y $l_{i,j}$ el comienzo y el fin de la ventana j para el cliente i.
El depósito tendrá también sus ventanas de tiempo al igual que los clientes, pero su demanda y su tiempo de servicio serán nulos.
En el depósito una flota de vehículos $V= \{1,...,n_v\}$ está disponible. Cada uno con capacidades diferentes, $Q = \{Q_1, .. Q_{nv}\}$, con $Q_i \geq \max \{q_j: j \in C\} \forall i = 1.. n_v$.
Observar que para un problema concreto n_v puede ser no especificado (infinito). En nuestro trabajo analizaremos el caso de cantidad finita de vehículos, con distintas capacidades. Además, consideramos que un mismo vehículo puede utilizarse para más de un recorrido, es decir, éste puede visitar algunos clientes, regresar al depósito para reabastecerse, y volver a salir visitando otros clientes.
Para dos localidades i,j conocemos la distancia d_{ij} del traslado de i a j, y el tiempo de traslado t_{ij}, asumiendo $t_{0,nc+1}= d_{0,nc+1}= 0$.
El objetivo es encontrar una asignación de clientes a vehículos tal que se respeten las restricciones de capacidad de cada vehículo y las restricciones de Time Windows; donde cada cliente sea visitado exactamente una vez.
El problema se modela con las siguientes variables de decisión:

x_{ij}^v = 1 si el vehículo v viaja directamente de i a j.
0 de lo contrario
b_i = tiempo en que comienza el servicio en el cliente i.
b_0^v = tiempo en el que el vehículo v abandona el depósito.
b_{nc+1}^v = tiempo en el que el vehículo v retorna al depósito.

Si el vehículo viaja de i a j y llega muy temprano a j (antes del comienzo de la ventana de tiempo más cercana), tendrá que esperar, es decir, b_j es tal que:

```
Para todas las ventanas hasta hallar_solucion
    Si b_i+s_i+t_ij < e_jk
        b_j=e_jk   // espero que abra la siguiente ventana
        hallar_solucion = true
    sino
        Si e_jk<= b_i+s_i+t_ij <=l_jk
            b_j=b_i+s_i+t_ij      // llegué dentro de una ventana
            hallar_solucion = true
        sino
            k = siguiente ventana
```

fin
fin
finPara
Si no hallé k en esas condiciones el cliente no puede ser visitado en esta oportunidad.

Se llama tiempo de espera en el cliente j a ($b_j - (b_i+s_i+t_{ij})$).
Definimos $tiempo_{ij}$ como el tiempo total insumido en ir del cliente i al j (incluye traslado, espera y servicio), es decir:

$tiempo_{ij} = s_i + t_{ij} + (b_j - (b_i+s_i+t_{ij}))$

Definimos y_i como la carga sobrante luego de atender al cliente i.

Restricciones
Modelamos el conjunto de soluciones posibles (x_{ij}^v) como sigue:

(1) $\sum_{v \in V} \sum_{j \in N} x_{ij}^v = 1 \qquad \forall i \in C$

Para todo nodo de salida i, existe un único vehículo, que va a salir de ese nodo hacia un único nodo de llegada j, exactamente una vez.

(2) $\sum_{j \in N} x_{ij}^v - \sum_{j \in N} x_{ji}^v = 0 \qquad \forall i \in C, v \in V$

Cada vez que un vehículo sale de un cliente, tiene que haber previamente llegado a él.

(3) si $x_{ij}^v = 1$ entonces $b_i + s_i + t_{ij} \le b_j \qquad \forall i,j \in C, v \in V$
Si un vehículo va de i a j entonces el tiempo de inicio de servicio en i, más el tiempo de servicio, más el tiempo de traslado de i a j debe ser menor o igual al tiempo de comienzo de servicio en j.

(4) si $x_{0j}^v = 1$ entonces $b_0^v + t_{0j} \le b_j \qquad \forall j \in C, v \in V$
El tiempo de salida del vehículo v del depósito más el tiempo de traslado a j, debe ser menor o igual al tiempo de comienzo de servicio en j.

(5) si $x_{i,nc+1}^v = 1$ entonces $b_i + s_i + t_{i,n+1} \le b_{nc+1}^v \qquad \forall i \in C, v \in V$
Si un vehículo va de i al depósito entonces el tiempo de inicio de servicio en i, más el tiempo de servicio, más el tiempo de traslado de i al depósito debe ser menor o igual al tiempo de retorno al depósito.

(6) $e_{ij} \le b_i \le l_{ij} \qquad \forall i \in C$, para algún $j = 1.. w_i$
El comienzo de servicio en i debe estar dentro de alguna ventana de tiempo de i.

(7) $e_{0j} \le b_0^v \le l_{0j} \qquad \forall v \in V$, para algún $j = 1..w0$
La salida del vehículo del depósito debe estar dentro de alguna ventana de tiempo del depósito.

(8) $e_{n+1,j} \le b_{n+1}^v \le l_{n+1,j} \qquad \forall v \in V$, para algún $j = 1.. w0$
El retorno del vehículo al depósito debe estar dentro de alguna ventana de tiempo del depósito.

(9) si $x_{ij}^v = 1$ entonces $y_i - q_j = y_j \quad \forall\, i, j \in N, v \in V$
Al visitar un cliente j baja la carga del vehículo en la cantidad entregada al cliente.

(10) $y_i \geq 0 \quad \forall\, i \in C \;,\; y_0 = Q_j \;, j \in \{1.. n_v\}$
Del depósito parten vehículos con diferentes cargas y al pasar por cada cliente la carga debe ser no negativa.

(11) $x_{ij}^v \in \{ 0, 1\}$
Cada arista se recorre una vez o ninguna.

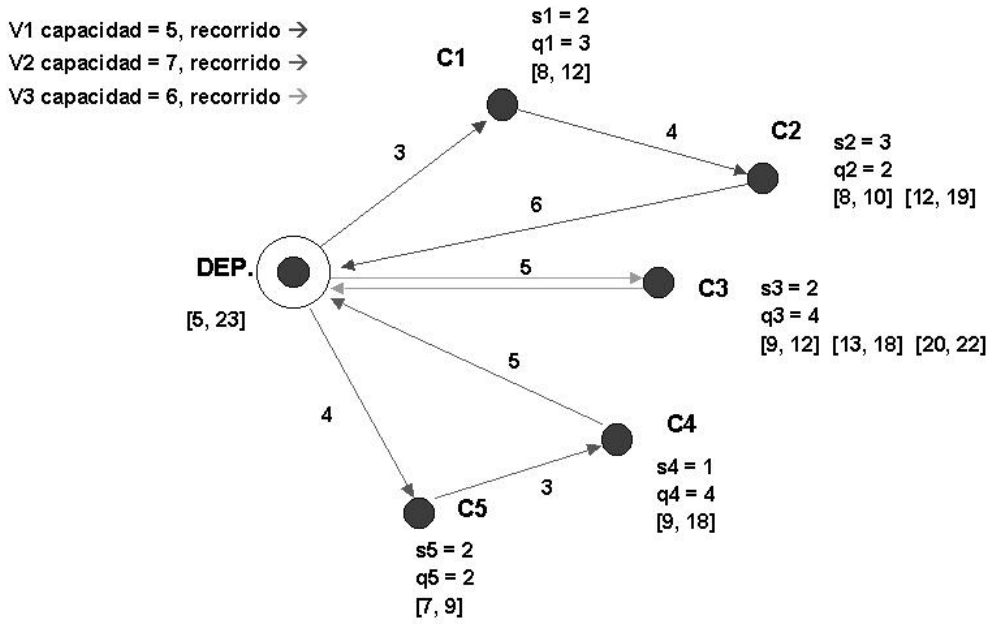
Función objetivo

Minimizar cost (X)= $\sum_{v \in N} \sum_{i \in N} \sum_{j \in N} c_{ij} * x_{ij}^v$

Mide el costo operacional de una solución basado en las distancias recorridas, y el tiempo total insumido (tiempos de traslado, tiempos de espera y tiempos de servicio).
Consideramos $c_{ij} = \varphi\, d_{ij} + \psi\, tiempo_{ij}$
donde φ es el factor de conversión de unidades de distancia a unidades de costo ($/unid.dist.), y ψ es el factor de conversión de unidades de tiempo a unidades de costo ($/unid.tiempo). Ambas son constantes ingresadas como parámetros.

Ejemplo

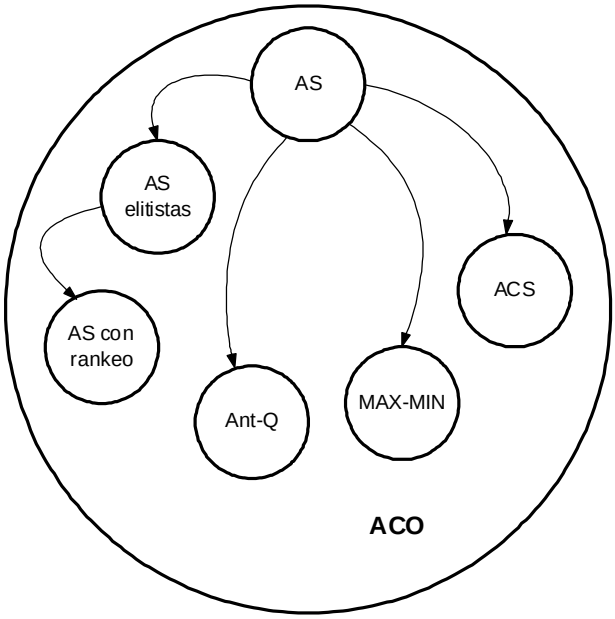
A continuación mostramos una posible solución (no necesariamente la mejor), a un problema VRP-mTW. Suponemos que las distancias son iguales a los tiempos de traslado, y que para las aristas que no se muestran en el dibujo éstos se obtienen a partir de las demás aristas (por ejemplo la arista (C5, C3) tendrá un tiempo de traslado de 9).



En este ejemplo, no es posible reutilizar los vehículo v1 o v2 para el recorrido DEP-C3-DEP, ya que a ninguno de estos dos vehículos les alcanza el tiempo para salir del depósito, visitar al cliente C3 y volver al depósito antes de que éste cierre. Tampoco es posible, por ejemplo, que el vehículo v1 haga el recorrido DEP-C5-C4-C3-DEP, ya que ese vehículo no tiene suficiente capacidad para satisfacer las demandas de estos tres clientes sin pasar por el depósito a reabastecerse (además de que ese recorrido tampoco cumpliría las restricciones de tiempo).

3. PRESENTACIÓN DE LOS ANT ALGORITHMS

Los Ant Algorithms son aplicaciones inspiradas en el comportamiento de las colonias de hormigas, una de sus principales ideas es la comunicación indirecta de los agentes (hormigas) basada en los rastros de feromonas. La meta-heurística ACO (Ant Colony Optimization) ha sido propuesta para ser un marco que unifique la mayoría de las aplicaciones basadas en Ant Algorithms (ver [16]).
Marco Dorigo es uno de los investigadores que comenzó a trabajar con estas heurísticas en 1991, definiendo el primer algoritmo ACO: Ant System (ver [1]). Él clasifica los ACO algorithms de la siguiente manera:



A continuación presentaremos en detalle el algoritmo Ant System, y luego las mejoras al mismo que proponen las variantes más conocidas. Estas variantes tienen como base la idea de Ant System, agregando mejoras en puntos específicos del algoritmo.

3.1. Ant System

El algoritmo Ant System fue desarrollado basándose en las características del comportamiento de las colonias de hormigas, y es aplicable a la solución de problemas computacionales de tipo NP-Completos (ver [43]).
La razón por la cual una hormiga puede seguir la ruta de sus compañeras es que las hormigas dejan una cierta cantidad de feromonas mientras caminan, y la probabilidad de que las próximas hormigas sigan un camino es proporcional a la cantidad de feromonas que hay en él. Una hormiga aislada se mueve al azar, pero otra que encuentra un rastro anterior va a elegir ese rastro con mayor probabilidad, y a su vez va a dejar sus propias feromonas, reforzando el rastro. Se introduce además el factor de evaporación, que permite que los caminos menos elegidos pierdan intensidad de rastro.
Este comportamiento es autocatalítico (“feedback loop”). La búsqueda de caminos por feedback loop sirve para obtener el camino seguido por la primer hormiga que llega al objetivo, pero este camino puede ser subóptimo.
La idea de Ant System es combinar un proceso autocatalítico con una fuerza greedy, ambos se complementan, ya que el proceso autocatalítico por si solo tiende a converger

a un camino subóptimo con velocidad exponencial. Pero juntos, el greedy guía al proceso autocatalítico, dejándolo converger a buenas soluciones (a veces óptimas), con gran rapidez.

Algoritmo Ant System (ref. [1])

Para introducir las ideas básicas de Ant System presentamos los conceptos aplicados al problema TSP (para VRP-mTW estas ideas básicas se mantienen).
El problema del viajante TSP consiste en un conjunto de ciudades y distancias establecidas para cada par de ellas, para las cuales es necesario hallar un camino que las visite exactamente una vez y minimice el costo total del recorrido.

Llamamos:

d_{ij} distancia entre las localidades i y j

τ_{ij} intensidad del rastro

$\rho < 1$ coeficiente de persistencia

$(1-\rho)$ evaporación

$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$ cantidad de rastro del camino de i a j

$\Delta\tau_{ij}^{k}$ rastro o feromona dejada por la hormiga k en el camino de i a j (cantidad /unidad de distancia)

$\eta_{ij} = 1/d_{ij}$ visibilidad

$visitadas_k$ = lista de las localidades visitadas. Cuando se completa el ciclo, se vacía la lista.

p_{ij} probabilidad de transición

Hay tres alternativas para el momento y la cantidad de feromona a dejar por las hormigas:

Ant-Cycle: Es la propuesta original. Luego que la hormiga concluye su ruta (visitando todas las ciudades exactamente una vez y volviendo al punto de partida), distribuye su feromona entre todos los tramos del camino, así, cuanto más corta sea la ruta final, más feromona va a haber en cada tramo de la ruta.

$$\Delta\tau_{ij}^{k} = \begin{cases} \Omega_3/L^k & \text{si la hormiga k va de i a j,} \\ & L^k \text{ largo total del ciclo de k.} \\ 0 & \text{si no} \end{cases}$$

Ant-Quantity y Ant-Density son extensiones del modelo original. La diferencia de éstos con el Ant-Cycle es que cada hormiga deja su feromona al final de cada paso en lugar de al final de cada ciclo.

Ant-Density: La hormiga deja una cantidad fija de feromona en cada camino.

$$\Delta\tau_{ij}^{k} = \begin{cases} \Omega_1 & \text{si la hormiga k va de i a j} \\ 0 & \text{si no} \end{cases}$$

Ant-Quantity: La cantidad dejada es proporcional al largo del camino.

$$\Delta\tau_{ij}^{k} = \begin{cases} \Omega_2/d_{ij} & \text{si la hormiga k va de i a j} \\ 0 & \text{si no} \end{cases}$$

Siendo $\Omega_1, \Omega_2, \Omega_3$ constantes prefijadas.

La fórmula de probabilidad que utiliza el método Ant System es la Random Proportional:

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \notin \text{visitadas}_k} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} & \text{Si } j \notin \text{visitadas}_k \\ 0 & \text{si no} \end{cases}$$

donde α y β son parámetros para controlar el peso relativo de rastro y visibilidad.
La actualización de rastros se realiza para todas las hormigas, usando la fórmula:

$$\tau_{ij}^{new} = \rho.\tau_{ij}^{old} + \Delta\tau_{ij}$$

Esqueleto del algoritmo

1. Inicialización de parámetros.

2. Repetir hasta llenar listas de visitadas
   2.1. Para cada localidad
      Para cada hormiga k en esa localidad
         Elegir la localidad a la cual se mueve la hormiga k con probabilidad p_{ij} y moverla hacia la misma.
         Insertar la localidad en $visitadas_k$.
      FinPara
   FinPara

   2.2. Calcular $\Delta\tau_{ij}^{k}$

   2.3. Calcular $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$

   2.4. Calcular τ_{ij} , p_{ij}

3. Almacenar el camino más corto encontrado hasta ahora y vaciar las listas de visitadas.

4. Si no se cumple condición de fin entonces
   Inicializar $\Delta\tau_{ij} = 0, \forall i,j$.
   Ir al paso 2.
   sino
   devolver el camino más corto
   finSi

3.2. Mejoras

Luego de la aparición de Ant System, se han presentado varios métodos que incluyen mejoras. Algunos de ellos son:

- Ant-Q (ref: [6], [9] y [16])
- Ant Colony System - ACS (ref: [7], [16])
- *MAX-MIN* Ant System - MMAS (ref: [10], [11], [15] y [16])
- Hormigas elitistas - $AS_{elitistas}$ (ref: [1])
- Hormigas elitistas rankeadas - AS_{rank} (ref: [13], [16])

En este Taller hemos analizado cada uno de estos métodos, decidiéndonos finalmente por el método ACS para la implementación de nuestra solución.
A continuación describimos las mejoras que presenta ACS, introduciendo previamente el método Ant-Q, ya que está ligado fuertemente a éste. En el Apéndice A se puede encontrar la descripción de los restantes métodos.

Método Ant-Q

Está basado en Ant System con influencias de Q-learning. Se le llama Q-learning a un tipo de refuerzo de rastros (reinforcement learnig). La diferencia de Ant-Q con Ant System es que tiene dos formas de actualizar los rastros, una local y otra global. La actualización global es similar a la de Ant System sólo que a la mejor hormiga se le permite actualizar el rastro. La actualización local de rastros es para hacer menos deseables las aristas ya elegidas, e incluye un término de refuerzo que depende del rastro máximo.

Método Ant Colony System (ACS)

Se basa en la misma idea que Ant–Q, donde el término de refuerzo ahora es constante. Se ha probado que utilizando una constante pequeña se obtienen resultados aproximadamente con la misma performance que Ant-Q, entonces debido a su simplicidad se utiliza ACS.
A continuación se detallan las mejoras que propone ACS sobre AS:

Una de las mejoras es la utilización de la fórmula Seudo Random Proportional para el cálculo de la probabilidad. Dicha fórmula consiste básicamente en lo siguiente:
Una hormiga k ubicada en la ciudad r elige la ciudad s (que no pertenece a la memoria $visitadas_k$) a la cual trasladarse según:

$$s = \begin{cases} \arg\max(\tau_{ru}^{\alpha} . \eta_{ru}^{\beta}) & q \le q_0 \\ u \notin visitadas_k & \\ S & \text{si no} \end{cases}$$

donde:
- $visitadas_k$ es la memoria de visitadas de la hormiga k
- q variable aleatoria con distribución uniforme(0, 1)
- q_0 parámetro , $0 \le q_0 \le 1$

- S variable aleatoria con distribución:

$$p_{rs}^{k} = \begin{cases} \dfrac{\tau_{rs}^{\alpha} \cdot \eta_{rs}^{\beta}}{\sum_{u \notin \text{visitadas}_k} \tau_{ru}^{\alpha} \cdot \eta_{ru}^{\beta}} & s \notin \text{visitadas}_k \\ 0 & \text{si no} \end{cases}$$

Con probabilidad q_0 se sigue la experiencia, con probabilidad (1- q_0) se exploran las diferentes aristas y aquellas con más intensidad de rastro y más cortas, tienen mayor probabilidad de ser elegidas.
Observar que Random Proportional es análogo a Seudo Random Proportional con $q_0 = 0$.

Otra de las mejoras es que sólo la hormiga que obtuvo el mejor camino actualiza el rastro, según una formula de actualización global.
Existen dos opciones para la actualización global de rastros:

Global-best: Toma como mejor solución la obtenida desde el comienzo del algoritmo.

Iteration-best: La mejor solución pertenece a la iteración actual, no se tiene en cuenta las mejores soluciones de iteraciones anteriores.

Para problemas pequeños las diferencias entre la global-best y iteration-best son mínimas, pero para problemas más grandes, el uso de la global-best otorga resultados bastante mejores. Por ese motivo ACS utiliza la opción global-best.

$$\tau ij^{new} = \rho.\tau ij^{old} + (1-\rho)\, \Delta\tau_{ij}^{best}$$
$$\Delta\tau_{ij}^{best} = 1/L^{best}$$

Además ACS incluye una actualización local para diversificar la exploración:

$$\tau_{ij} = (1-\xi).\tau_{ij} + \xi.\tau_0$$

donde τ_0 y ξ son parámetros.

4. POST-OPTIMIZACIÓN DE LAS SOLUCIONES

En la bibliografía estudiada se recomienda complementar los algoritmos ACO con una post-optimización, ya que la experiencia ha demostrado que combinando estos métodos se obtienen muy buenas soluciones.

Para la post-optimización de una solución previamente construida por el algoritmo, optamos por la aplicación de un método que reordene los clientes dentro de la ruta buscando una combinación de menor costo; y otro que ajuste los horarios de salida de los vehículos permitiendo la reducción de los tiempos de espera (y por consiguiente del costo).

Para la implementación del primer método, hemos analizado y comparado los algoritmos 2-opt, 3-opt y OR-opt. A continuación se detallan estos algoritmos, así como también el algoritmo utilizado para el ajuste de tiempos.

Métodos 2-opt y 3-opt

Los algoritmos 2-opt y 3-opt son del tipo r-opt propuesto por Lin en 1965 (ref: [44]). Este consiste en tomar r arcos del tour y reconectarlos de todas las maneras posibles, quedándonos con la de menor costo. Requiere de n^r operaciones.
Un tour se llama 2-optimal (2-opt) si no hay posibilidad de acortar el tour con el intercambio de dos arcos. En la heurística 2-opt, se sacan dos arcos de la ruta y las dos subrutas resultantes son reconectadas con otros dos arcos (notar que hay una sóla forma de reconectar la ruta). Se toman decisiones localmente óptimas: Greedy.
Primero se agrupan los clientes y después se generan rutas de vehículos 2-opt para cada grupo. Al final de cada iteración del Ant System, cada ruta es chequeada para 2-opt y es mejorada si es posible. Solo ahí se calcula el valor objetivo total y los rastros son actualizados.

Método OR-opt

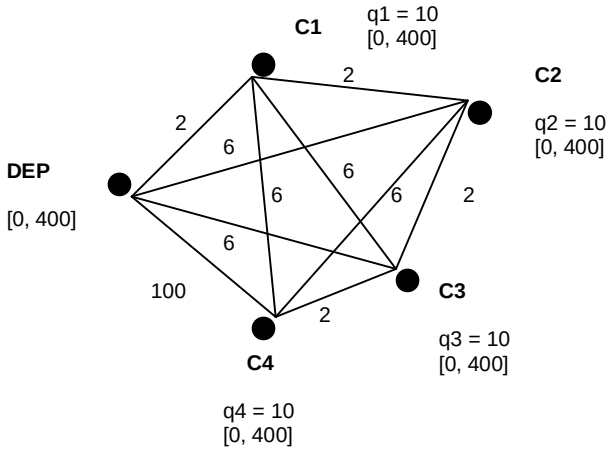
Es un método propuesto por OR en 1976 (ref: [44]). El algoritmo consiste en:

Paso 1 – Considerar un tour inicial y setear t = 1, s = 3.
Paso 2 – Sacar del tour la cadena de s vértices consecutivos que comienza en t.
Simular la inserción entre todo par de vértices consecutivos de la ruta:
- Si alguna inserción disminuye el costo de la ruta, implementar la inserción, setear t = 1 , ir al paso 2.
- Si ninguna inserción disminuye el costo,
  setear t = t + 1
  Si t = n + 1 ir al paso 3
  sino ir al paso 2

Paso 3 – Setear t = 1, s = s – 1
Si s > 0 ir al paso 2
sino parar.

Para nuestro problema en particular, como tenemos ventanas de tiempo, y las vamos considerando al armar la ruta, no es conveniente usar 2-opt, ya que este cambia el

sentido de los arcos. Al elegir entre OR-opt y 3-opt, nos quedamos con OR-opt, ya que requiere solo n^2 operaciones, y produce resultados tan buenos como 3-opt.
Ejemplo:
Para simplificar suponemos que los tiempos de servicio y las distancias son cero.



Solución original: DEP-C1-C2-C3-C4-DEP Costo: 108
Solución mejorada por OR-opt: DEP-C2-C3-C4-C1-DEP Costo:18

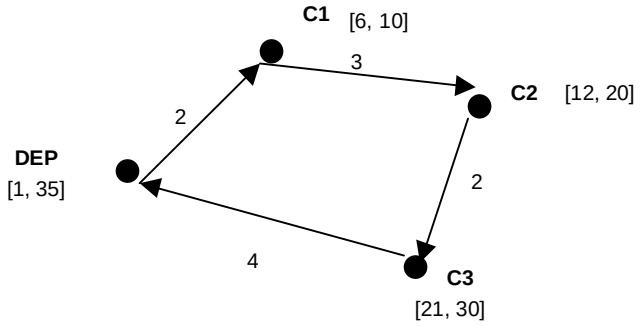
Ajuste de tiempos

Aquí nuestro objetivo es modificar el horario de salida de los vehículos de cada recorrido (sin violar las restricciones de tiempo de los clientes), permitiendo reducir en los casos que sea posible, el tiempo de espera y por consiguiente el costo.
El algoritmo consiste en:

```
Setear un valor máximo para horas_libres
Setear i = primer cliente del recorrido
Mientras (no llegué al fin del recorrido y horas_libres > 0 )
        j = proximo cliente
        horas_disponibles = tiempo que se puede atrasar la atención del cliente i,
                            sin salirse de la ventana de tiempo
        horas_libres = mínimo(horas_libres, horas_disponibles)
        espera = tiempo de espera al ir de i a j
        Si espera < horas_libres
                horas_libres = horas_libres – espera
                ajustar = espera
        sino
                ajustar = horas_libres
                horas_libres = 0
        finSi
        actualizar horas visitado para el cliente i y todos los clientes visitados
        antes, sumándoles el valor “ajustar”
finMientras
```

La idea es mantener una variable (horas_libres) que me dice cuantas horas se puede atrasar la salida del vehículo. Esta variable va decreciendo en base a las horas disponibles de los clientes, y se actualiza cada vez que se elimina una espera. Cuando se agotan las horas libres, finaliza el algoritmo.

Ejemplo:
Para simplificar el ejemplo, suponemos que los tiempos de servicio y las distancias son nulas.



Solución original:
Horas de visitado: DEP = 1, C1 = 6, C2 = 12, C3 = 21, DEP = 25
Costo: 24
Solución mejorada por "ajustar_tiempos"
Horas de visitado: DEP = 8, C1 = 10, C2 = 13, C3 = 21, DEP = 25
Costo: 17

5. SOLUCIÓN AL PROBLEMA VRP-mTW PROPUESTA

5.1. Formalismo

Para la elección del formalismo a utilizar, comparamos los distintos ACO algorithms existentes, presentados en la Sección 3. Ant System fue el primer formalismo y luego se propusieron las mejoras. La principal característica común en estas mejoras propuestas es que explotan más las mejores soluciones encontradas durante la búsqueda. Típicamente, esto se hace dándole mayor peso a las mejores soluciones en la actualización de rastros de feromona; y generalmente permitiendo que se deposite feromona adicional en los arcos de la mejor solución encontrada. En particular, en $AS_{elitistas}$ y $AS_{rankeadas}$ se agrega un fuerte refuerzo adicional a la actualización de rastros de la mejor solución, además en $AS_{rankeadas}$ algunas de las mejores hormigas de la iteración también agregan feromona adicional; en ACS y MMAS, sólo la mejor hormiga (ya sea la global-best o iteration-best) depositan feromona. Obviamente al aprovechar más las mejores soluciones, los arcos contenidos en los mejores tours reciben una fuerte realimentación adicional y por lo tanto son elegidos con mayor probabilidad en subsecuentes iteraciones del algoritmo.
Aún así, un problema asociado con este mayor aprovechamiento de la experiencia de búsqueda, puede ser el estancamiento de la búsqueda (situación en la cual todas las hormigas siguen el mismo camino). Por lo tanto algunos de los algoritmos ACO propuestos, en particular ACS y MMAS, introducen características adicionales para evitar el estancamiento. En ACS esto se logra con la regla de actualización local de rastros, que decrementa la cantidad de feromona en cada arco por el que pasa una hormiga, haciéndolo menos atractivo para las hormigas siguientes. De esta forma se favorece la exploración de arcos todavía no visitados. En MMAS el estancamiento se evita imponiendo límites en la cantidad de feromona permitida en cada arco.

En base a lo expuesto anteriormente, elegimos ACS para la elaboración del formalismo que resolverá nuestro problema VRP-mTW. Además, para la actualización de rastros elegimos la variante Ant Cycle, por ser estadísticamente la que proporciona mejores resultados.

5.2. Decisiones de diseño

Recordamos la definición del problema:
Se tiene un conjunto de clientes dispersos en una región geográfica y un depósito que almacena cierto producto. Cada cliente tiene una demanda solicitada de ese producto y una o más ventanas de tiempo que delimitan el período de tiempo en el cual el cliente debe ser atendido. Existe una cierta cantidad de vehículos de capacidades limitadas y diferentes, que pueden salir del depósito una o más veces con su capacidad completa, y se encargarán de distribuir a cada cliente de la región el total de su demanda solicitada, dentro de alguna de sus ventanas de tiempo. El objetivo es encontrar un camino que visite cada cliente exactamente una vez, satisfaciendo su demanda, sin violar las restricciones de tiempo y capacidad, que minimice el costo de traslado.

El esquema básico de un algoritmo ACS es el siguiente:
Se define un cierto número de hormigas y una condición de fin. En cada iteración cada una de las hormigas se encarga de encontrar un camino, cuando todas han completado sus rutas se comparan entre sí y se elige la mejor, luego se compara ésta con la mejor hasta el momento, actualizando los rastros con la solución resultante. Se testea la condición de fin y si está no se cumple, continúa con la siguiente iteración.

Con respecto a la iteración principal, tenemos dos opciones básicas:
a) Iterar sobre clientes: nos serviría para "simular" una paralelización (porque hacemos que todas las hormigas se muevan un paso al mismo tiempo). Este tipo de iteración se utiliza cuando se implementa la versión Ant Density o Ant Quantity de actualización de rastro.
b) Iterar sobre hormigas: es la versión más evidente cuando se utiliza Ant Cycle para actualizar rastros.

Como nuestro método de actualización de rastros es el Ant Cycle, elegimos la opción de iterar sobre hormigas. Además, al utilizar ACS solamente la mejor hormiga de la iteración va a actualizar rastros, por lo tanto el iterar sobre hormigas nos permite almacenar solamente la mejor ruta construida hasta el momento, ahorrándonos el espacio de memoria y tiempo de ejecución que consumiría el ir actualizando las rutas construidas por todas las hormigas si se utilizara la iteración sobre clientes.

Dentro de este lineamiento general, tenemos distintas opciones para cada particularidad:
- elección de vehículos
- elección del próximo cliente a visitar
- tratamiento soluciones no factible
- ordenamiento de soluciones
- criterio para aplicación de post-optimización

Elección de vehículos

Al comenzar a construir la ruta de una hormiga, y cada vez que un vehículo llega al depósito, se comienza un nuevo recorrido, al cual debe asignarse un vehículo.
Tenemos distintos criterios para elegir el vehículo a utilizar en ese nuevo recorrido:
i) Según el tiempo disponible del vehículo:
1) dar prioridad a los que tienen más horas disponibles.
2) dar prioridad a los que tienen menos horas disponibles.

ii) Según la capacidad del vehículo: dar prioridad a los que tengan más carga disponible.

Para que la decisión quede a cargo del usuario, definimos una función de probabilidad para la elección del vehículo, la cual depende de dos parámetros γ y δ:

$$prob = \frac{h^{\gamma} * q^{\delta}}{\Sigma\, h^{\gamma} * q^{\delta}}$$

donde h son las horas disponibles del vehículo y q su capacidad.
Entonces:
- para la opción i.1) se elige $\gamma > 0$
- para la opción i.2) se elige $\gamma < 0$
- para la opción ii) se elige $\delta > 0$

Elección del próximo cliente a visitar

Para la elección del próximo cliente a visitar utilizamos la fórmula de Random Proportional del método ACS (ver Sección 3.2), con algunas variantes.

- Para el cálculo de la visibilidad del cliente actual i, a cada cliente c (η_{ic}) consideramos no sólo la distancia, sino también el tiempo de entrega (formado por el tiempo de servicio en i, más el tiempo de traslado y el tiempo de espera). Además multiplicamos distancia y tiempo por los factores de conversión a unidad de costo correspondientes, para permitir que se de mayor o menor peso a uno u otro factor, de acuerdo a los valores ingresados.

  $costo_{ic} := \varphi.\ distancia_{ic} + \psi.\ t\_entrega_{ic}$

  $\eta_{ic} := 1 / costo_{ic}$

- Para el cálculo de las probabilidades introducimos un factor más que permite minimizar la cantidad de veces que se vuelve al depósito, lo cual en la mayoría de los casos significa una minimización en los costos. Logramos esto haciendo que el depósito tenga una probabilidad de ser elegido inversamente proporcional a la carga restante del vehículo que se está utilizando en el momento, de este modo cuanto mayor sea la carga restante del vehículo (q_v), menor será la probabilidad de volver al depósito. Para ello al calcular la probabilidad de visitar el depósito multiplicamos por $(1/q_v)^{\omega}$, donde ω es un parámetro que le da mayor o menor peso a este factor y que debe ser mayor o igual a 0.
  Fórmula Seudo Random Proportional adaptada:

$$s = \begin{cases} \arg\max(\tau_{ru}^{\alpha} . \eta_{ru}^{\beta} . \boldsymbol{\varepsilon_u}) & q <= q_0 \\ u \notin \text{visitadas}_k & \\ S & \text{si no} \end{cases}$$

donde:
- visitadas_k es la memoria de visitadas de la hormiga k
- q variable aleatoria con dist uniforme(0, 1)
- q_0 parámetro , $0 <= q_0 <= 1$
- S variable aleatoria con distribucion:

$$p_{rs}^{k} = \begin{cases} \dfrac{\tau_{rs}^{\alpha} . \eta_{rs}^{\beta} . \boldsymbol{\varepsilon_s}}{\sum_{u \notin \text{visitadas}k} \tau_{ru}^{\alpha} . \eta_{ru}^{\beta} . \boldsymbol{\varepsilon_u}} & s \notin \text{visitadas}_k \\ 0 & \text{si no} \end{cases}$$

$$\text{donde } \varepsilon_u = \begin{cases} (1/ q_v)^{\omega} & \text{si } u = \text{depot} \\ 1 & \text{si } u \neq \text{depot} \end{cases}$$

Observar que si $\omega = 0$, $\varepsilon_{\text{deposito}} = 1$ y entonces la fórmula se convierte en la original.

Tratamiento de soluciones no factibles

Disponemos de varias opciones para el tratamiento de soluciones no factibles:

- *Desecharlas*. No es muy conveniente porque se pierden soluciones, que en el caso en que todas las soluciones de la iteración sean no factibles, podrían ser útiles para llegar a alcanzar una solución factible.
- *Factibilizarlas*. Existen métodos para factibilizar soluciones (por ejemplo mediante intercambio de arcos), pero son demasiado costosos y además no garantizan que la solución se pueda factibilizar.
- *Guardarlas con un costo alto*. Es la opción que elegimos por considerarla la más adecuada. Para implementarla asignamos a las soluciones no factibles un costo que depende de la cantidad de clientes no visitados, asegurándonos de poner un valor lo suficientemente alto como para no confundir con las soluciones factibles. Esto nos permite ordenar las soluciones no factibles, y en caso que en una iteración todas las soluciones obtenidas sean no factibles, eligiríamos la "más cercana a factible" permitiendo que el algoritmo pueda converger a una solución factible.

  costo = $(k_1 + k_2)$* cant de clientes no visitados

  donde $k_1 = \Sigma_i \Sigma_j d_{ij} + 1$

  $k_2 = n_v * (l_{0,w0} - e_{0,1}) + 1$

  Elegimos k1 mayor a la suma de todos los caminos del grafo completo que representa el problema, pues todas las rutas factibles tendrán necesariamente una distancia menor a ese valor.

  Elegimos k2 mayor al tiempo que tardarían todos los vehículos si viajaran durante todo el horario en que el depósito está abierto.

Ordenamiento de soluciones

En el ordenamiento de las soluciones, la primer prioridad la tienen los menores costos. Como segunda prioridad se pueden elegir:
- menor distancia
- menores tiempos de espera
- menor cantidad de vehículos

Elegimos la opción de menor cantidad de vehículos como segunda prioridad, por considerar que es el factor de mayor peso económico para una organización.
Entonces, si al comparar dos rutas éstas tienen costos idénticos, nos quedaremos con la que utiliza menor cantidad de vehículos.

Criterio para aplicación de Post-optimización

En cuanto a cuáles soluciones aplicar Post-optimización, tenemos tres opciones:
- optimizar todas las soluciones conseguidas por todas las hormigas de la iteración.
- optimizar solamente la solución de la mejor hormiga de la iteración.
- optimizar sólo la mejor solución de la corrida (o sea, la mejor entre todas las iteraciones).

Elegimos la segunda, por ser una opción intermedia que no incrementaría tanto los tiempos de ejecución como la primera, y que nos permitiría obtener una retroalimentación ya que en cada iteración se utilizaría el camino optimizado para la actualización de rastros.

5.3. Algoritmo

La estructura básica del algoritmo es la siguiente:

1) Inicialización
2) Para cada hormiga
    Elijo vehículo para primer recorrido
    Loop hasta que todos los clientes estén visitados
        Elijo próximo cliente a moverme
        Si cliente elegido es el depósito
            Guardo ciclo actual
            Comienzo un nuevo ciclo
            Elijo un vehículo para el nuevo ciclo
            Elijo próximo cliente a moverme
        Marco el cliente como visitado
3) Guardo mejor camino.
4) Post optimización.
5) Actualizo intensidad de rastros para próxima iteración.
6) Si condición de fin, termina
    sino vuelve a 1) – *iteración siguiente*

6. DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN

6.1. Elección del lenguaje

La implementación de este proyecto la debíamos realizar en un lenguaje orientado a objetos: Java o C++.

Para la elección del lenguaje no tuvimos en cuenta las herramientas para construir interfase gráfica (por ejemplo JavaBuilder o Visual C++), ya que nuestro trabajo se concentra en construir una biblioteca de funcionalidades. Esta biblioteca la construiremos pensando en que luego sea posible adaptar una interfase gráfica para la carga de datos y ejecución (que podría ser en un lenguaje simple como VisualBasic), pero principalmente pensando en que en un futuro pueda formar parte de otro proyecto más grande.

Teniendo en cuenta estos aspectos, comparamos las ventajas y desventajas de estos dos lenguajes para poder elegir el que más se ajuste. Elegimos entonces C++, por las razones siguientes:

- Java se precompila y luego este precompilado se puede interpretar desde distintas plataformas. Esto es útil para programas Internet pero ese no es nuestro caso ya que nuestro programa correrá localmente.
  Esa "portabilidad" que nos ofrece Java de todos modos la podemos lograr con C++: utilizando el estándar ANSI y compilando los fuentes con distintos compiladores (Unix y Windows), tendremos un programa ejecutable que correrá más rápido que el programa precompilado, para cada una de estas plataformas.
- Otro de los factores para elegir C++ fue que no debíamos invertir tiempo del proyecto en aprender el lenguaje, ya que previamente habíamos trabajado con esta herramienta.

6.2. Estructuras de datos y diseño de clases

Estructuras de datos

En el modelo tenemos los siguientes elementos: depósito, clientes, información de trayectos entre clientes y depósito, vehículos y rutas.

Los clientes contendrán la siguiente información:
- identificación del cliente
- demanda
- tiempo de servicio
- ventanas de tiempo

La identificación del cliente está dada por la numeración secuencial de los mismos. Si la cantidad de clientes es n_c, los clientes estarán numerados como $1...n_c$.
Al depósito lo consideramos como un cliente más, con identificador 0 y demanda nula. Por lo tanto los clientes y el depósito se almacenarán juntos en una lista de clientes. Esta lista la implementamos como un array de tamaño $n=n_c+1$.

Para almacenar la información de trayectos entre clientes y el depósito disponemos de un grafo. Este grafo lo podemos pensar como una matriz de n*n, donde cada elemento (i,j) de la misma contiene la información del trayecto (arista) entre los clientes de identificación i y j. Dicho elemento tendrá entonces la siguiente información:
- distancia
- tiempo
- rastro

Debido a que el problema VRP que estamos modelando es simétrico, el grafo es no dirigido y por lo tanto la matriz es simétrica. Por este motivo almacenamos sólo un par (i,j) para mantener la información entre los clientes i y j. A la hora de implementar, elegimos un array unidimensional en el cual guardamos la parte superior de la matriz simétrica en forma secuencial, por lo tanto el tamaño de este array es (n*n + n)/2. Con la ayuda de una función de mapeo, podemos acceder a la información entre los clientes i y j con lo cual esta representación es transparente a la hora de utilizar el grafo.

De los vehículos nos interesan estos datos:
- identificación del vehículo
- carga
- horario

La identificación de los vehículos es la numeración secuencial de los mismos, es decir, si la cantidad de vehículos es n_v, estarán numerados como $0..(n_v - 1)$.
La carga representa la cantidad de mercadería que está almacenando en ese momento el vehículo. El horario es el tiempo en el cual se encuentra el vehículo.
Almacenamos los vehículos en una lista, la cual implementamos como un array de tamaño igual a la cantidad de vehículos.

Cada ruta representa un camino recorrido por una hormiga. En la ruta tendremos la siguiente información:
- identificación de la hormiga que la construyó
- distancia
- tiempo

- cantidad de vehículos utilizados
- recorridos realizados por cada vehículo
- cantidad de clientes no visitados

La identificación de las hormigas es la numeración secuencial de las mismas. Si la cantidad de hormigas es m, estas estarán numeradas como 1...m.
La distancia y el tiempo son el total de distancia y tiempo insumidos en esa ruta.
La cantidad de clientes no visitados es 0 cuando la solución es factible. Nos interesa tener esa información para poder guiar las iteraciones hacia una convergencia cuando todas las rutas encontradas por las hormigas son no factibles, permitiendo a la hormiga que tuvo menos clientes no visitados actualizar los rastros. Cuando la solución es no factible el costo se calcula como la distancia y el tiempo máximos en el grafo y se le suma la cantidad de clientes no visitados. De ese modo, si en una iteración todas las soluciones encontradas por las hormigas son no factibles, nos quedaremos con la más cercana a factible, ya que tendrá menor cantidad de clientes no visitados y por lo tanto menor costo.

Dentro de una ruta, los recorridos realizados por cada vehículo se agrupan en una lista. Cada recorrido consta de la siguiente información:

- identificación del vehículo que lo realizó
- hora de salida
- hora de llegada
- distancia recorrida
- clientes visitados

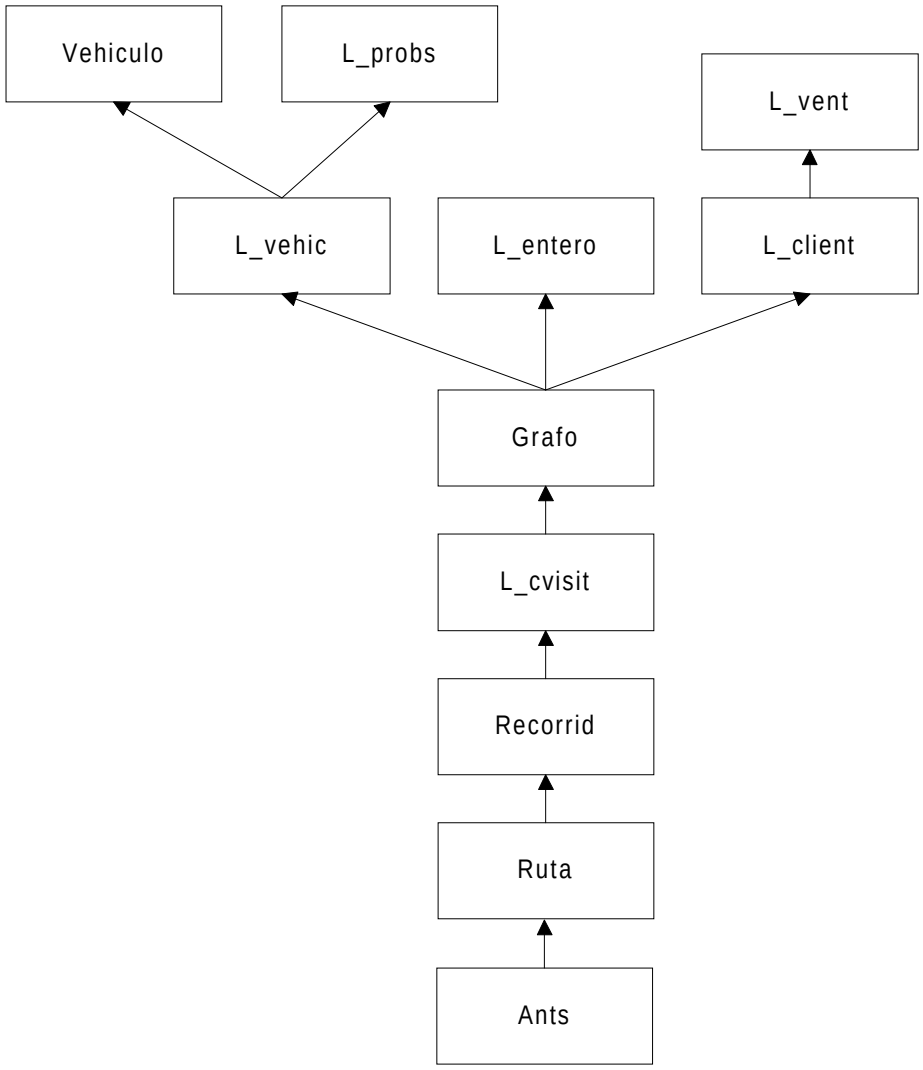
Los clientes visitados se agrupan en una lista, que implementamos como un array dinámico. En esa lista, los clientes se almacenan en el orden en que fueron visitados. De cada cliente visitado nos interesa la siguiente información:

- identificación del cliente
- hora de visitado
- número de ventana
- carga

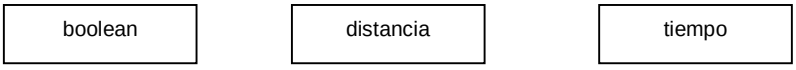
La hora de visitado es el tiempo en el cual dicho cliente comenzó a ser atendido. El número de ventana nos dice la ventana de tiempo del cliente que se utilizó para visitarlo.
La carga es la dejada por el vehículo al visitar a ese cliente, que en nuestro caso será igual a la demanda del cliente ya que no trabajamos con cargas fraccionadas.
El depósito será también para nosotros un cliente visitado, con la particularidad de que se puede visitar varias veces y que tendrá carga nula. Recordemos que el vehículo al llegar al depósito no deja mercancía, en su lugar lo que hace es reabastecerse de carga.

En cada iteración las hormigas construyen sus rutas y nos quedamos con la mejor de todas. Al finalizar la iteración, comparamos la mejor ruta obtenida en la misma con la mejor ruta de todas las iteraciones hasta el momento, sustituyéndola si es mejor.

Diseño de clases



Tipos de datos auxiliares.



Se utilizan en todas las clases.

6.4. Entrada de datos

Archivo de parámetros

El archivo de parámetros se llamará "param.dat".

Formato

```
<cant_horm>
<max_iter>
<rho>
<alfa>
<beta>
<gamma>
<delta>
<omega>
<q0>
<xi>
<tau0>
```

Los parámetros tienen las siguientes características:

Nombre	Descripcion	Tipo	Rango
cant_horm	cantidad de hormigas con que se desea correr el algoritmo.	entero	1 .. 500 (*)
max_iter	cantidad de iteraciones que se desean realizar.	entero	1 .. 500 (*)
rho	factor de persistencia del rastro -ρ-.	real	0 .. 1
alfa	peso relativo del rastro -α-.	real	> 0
beta	peso relativo de la visibilidad -β-.	real	> 0
gamma	peso relativo del tiempo restante del vehículo -γ-.	real	> 0
delta	peso relativo de la carga restante del vehículo para el cálculo de la probabilidad de elegir ese vehículo -δ-.	real	> 0
omega	peso relativo de la carga restante del vehículo para el cálculo de la probabilidad de volver al depósito -ω-.	real	> 0
q0	parámetro para la elección del próximo cliente mediante el método Seudo Random Proportional -q_0-.	real	0 .. 1
xi	factor de evaporacion del rastro para actualización local -ξ- .	real	0 .. 1
tau0	constante para actualización local de rastro -τ_0-.	real	0 .. 1 (*)

Nota: los rangos marcados con (*) no son obligatorios pero sí recomendados.

Archivo de datos

Se diseñó un nuevo formato de archivo de datos para los problemas VRP-mTW, al que llamamos “formato ants”.
El archivo de datos formato ants se divide en cuatro bloques:

1) El primero contiene la información sobre los clientes. En el primer renglón debe ir la cantidad de clientes (k), y en los siguientes k renglones: el número de cliente (nro. consecutivo de 0 a k), la cantidad de ventanas (n), las horas de apertura y cierre para cada ventana, y por último la demanda y el tiempo de servicio del cliente.
2) El segundo contiene la información de distancia y tiempo entre pares de clientes. En el primer renglón debe ir la cantidad de pares brindados (m) y en los siguientes m renglones: el número de cliente origen, el número de cliente destino, y la distancia y tiempo entre los mismos. Nota: se pide el número de pares ya que no es obligatorio informar los datos para todos los pares de clientes posibles.
3) El tercer bloque contiene el factor de conversión de unidades de distancia a unidades de costo y el factor de conversión de unidades de tiempo a unidades de costo. Estos parámetros permiten regular el peso que se le quiere dar a la distancia o al tiempo en el cálculo de los costos mínimos.
4) El cuarto y último bloque contiene la información sobre los vehículos. En el primer renglón debe ir la cantidad de vehículos (m), y en los siguientes m renglones el número de vehículo (consecutivo de 0 a m) y la capacidad del mismo.

Formato

```
<can_cli>
0 <can_ven> <ap_ven1> <ci_ven1>...<ap_venN> <ci_venN> <dem> <t_ser>
   .
   .
   .
K-1 <can_ven> <ap_ven1> <ci_ven1>...<ap_venN> <ci_venN> <dem> <t_ser>
<can_pares_cli>
<cli_i> <cli_j> <dist> <tiempo>
   .
   .
   .
<cli_i> <cli_j> <dist> <tiempo>
<peso_dist>
<peso_tiempo>
<can_vehic>
0 <carga>
   .
   .
   .
M <carga>
```

Los datos tienen las siguientes características:

Nombre	Descripcion	Tipo	Rango
can_cli	cantidad de clientes del problema	entero	1 .. m_c
nro_cli	identificador del cliente (en orden consecutivo ascendente de 0 a can_cli)	entero	0 .. can_cli
can_ven	cantidad de ventanas del cliente	entero	0 .. m_{cv}
ap_ven	hora de apertura de la ventana	tiempo	0 .. m_h
ci_ven	hora de cierre de la ventana	tiempo	0 .. m_h
dem	demanda	real	0 .. m_q
t_ser	tiempo de servicio	tiempo	0 .. m_h
can_pares_cli	cantidad de pares de clientes informados	entero	0 .. k
cli_I	nro. cliente	entero	0 .. can_cli
cli_j	nro. cliente	entero	0 .. can_cli
dist	distancia entre los clientes i y j	real	0 .. m_d
tiempo	tiempo del recorrido entre los clientes i y j	entero	0 .. m_h
peso_dist	factor de conversión de unidades de distancia a unidades de costo	real	0 .. m_d
peso_tiempo	factor de conversión de unidades de tiempo a unidades de costo	real	0 .. m_h
can_vehic	cantidad de vehículos	entero	0 .. m_v
nro_vehic	identificador del cliente (en orden consecutivo ascendente de 0 a can_vehic)	entero	0 .. can_vehic
carga	carga del vehículo	entero	0 .. m_q

Nota1: el tiempo debe expresarse como un número entero que corresponda al tiempo en minutos. Por ejemplo, para expresar la hora 04:30 se deberá escribir 270 (que se calcula como 4 *60 + 30).

Nota2: los valores m_c, m_h, m_{cv}, m_v, m_d denotan:

m_c = máxima cantidad de clientes

m_h = horario máximo

m_{cv} = máxima cantidad de ventanas

m_v = máxima cantidad de vehículos

m_d = distancia máxima

m_q = carga máxima

Cabe señalar que estos máximos en principio son valores grandes, es decir teóricamente no acotados, pero que en la práctica se encuentran delimitados por la implementación elegida, la interrelación entre los datos del programa, y que dependen además de la máquina y la plataforma a utilizar en la corrida del programa. Por ejemplo, si se utilizan tiempos de servicio exageradamente grandes para cada cliente, el tiempo de la ruta que contiene a todos los clientes

puede sobrepasar el número máximo que es capaz de almacenar la máquina (constante maxint), generando errores de ejecución al calcular los costos.

Transformación de formato

Se realizaron tres programas auxiliares para transformar al formato ants los formatos existentes en las bibliotecas de problemas estudiados: trtsplib, trvrplib, trsolom. Cada uno de ellos se ejecuta en línea de comandos escribiendo el nombre del programa seguido por el nombre del archivo, por ejemplo: `trtsplib <nombre_archivo>`.
En el Apéndice C se puede ver un instructivo para la ejecución de estos programas.

Para cada formato se asumieron fijas las propiedades que se detallan a continuación.

1) trtsplib

Transforma al formato ants archivos de problemas .vrp que son una extensión del formato TSPLIB para problemas de ruteo de vehículos con capacidad finita (ref [32]).

Estos archivos contienen una parte de especificación del problema bajo el rótulo de NAME, COMMENT y TYPE. Luego viene la etiqueta DIMENSION que indica la cantidad de clientes incluido el depósito. La siguiente sección es EDGE_WEIGHT_TYPE que asumimos en EUC_2D (distancia euclideana para dos dimensiones), En el cálculo de la distancia se realiza un redondeo al entero más cercano.

CAPACITY indica la capacidad de los vehículos en alguna unidad de masa, y CAPACITY_VOL la capacidad en metros cúbicos. Para trabajar con mayor generalidad solo tenemos en cuenta el valor indicado bajo CAPACITY.

En la sección NODE_COORD_SECTION las columnas son: número correspondiente al cliente, coordenada *x*, coordenada *y*. Como en este formato los clientes se numeran a partir del uno, y en nuestro caso los numeramos a partir del cero, hacemos un mapeo en el que cada cliente va a corresponderse con el número de cliente anterior.

Luego viene la sección DEMAND_SECTION que especifica la demanda para cada cliente, con el mismo criterio de numeración de clientes. Son tres columnas; número de cliente, demanda en unidad consistente con la de CAPACITY y la tercer columna es la demanda según CAPACITY_VOL. Como en nuestro caso no consideramos a este último tampoco consideramos la tercer columna.

Para mantenernos dentro de los márgenes de nuestro caso de estudio no tuvimos en cuenta la PICKUP_SECTION, ya que sería aplicable solamente si de los clientes se levantara mercadería.

En TIME_WINDOW_SECTION tenemos número de cliente, hora de apertura, hora de cierre de la ventana, en ese orden, y es posible que el mismo cliente aparezca varias veces en la misma sección (lo que significa que tiene varias ventanas de tiempo). Para nuestro estudio aplicado a este formato de archivos TSPLIB limitamos el número de ventanas a cinco, debido a que pensamos que es un número razonable de ventanas y a su vez acotar este número ayuda a optimizar el proceso de transformación de un formato a otro. Esta sección no es obligatoria, y en caso de no estar presente para algún cliente, le asignamos al mismo una ventana máxima.

STAND_TIME_SECTION lista los tiempos de servicio de cada cliente.

En DEPOT_SECTION se especifican los clientes que son depósito, pero como en nuestro estudio solo hay un depósito y es el cliente cero, asumimos por simplicidad que este es siempre el caso.
La cantidad de vehículos y la proporción entre distancia y tiempo son datos adicionales que no proporciona este formato ya que asume cantidad infinita de vehículos y tiempo de traslado igual a distancia; para dar mayor flexibilidad damos la opción de ingresar aquellos datos por pantalla.
Se piden además por pantalla los factores de ponderación para distancia y tiempo (ψ y φ), que se utilizarán en el algoritmo para calcular los costos.
A continuación presentamos un ejemplo de este formato de archivo (nota: se resumieron partes repetitivas del archivo utilizando "..." para que pueda visualizarse con mayor claridad)

```
NAME : engezf\1a2b3a4a.knd
COMMENT :
TYPE : CVRP
DIMENSION : 201
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 12600
CAPACITY_VOL : 69
NODE_COORD_SECTION
1 23452 56766
2 23136 55261
3 24389 55556
4 24005 55379
...
201 23289 56817
DEMAND_SECTION
1 0 0
2 2290 0
3 1250 0
4 2030 0
...
201 3720 0
PICKUP_SECTION
1 0 0
2 0 0
3 0 0
4 0 0
...
201 0 0
TIME_WINDOW_SECTION
2  7:00 18:00
3  7:00 18:00
4 11:15 12:15
...
201 14:45 15:15
STANDTIME_SECTION
2 27
3 17
4 25
...
201 42
DEPOT_SECTION
1
-1
EOF
```

2) trvrplib

Estos son archivos para problemas de cien clientes más un depósito con dos ventanas de tiempo cada uno (ref [31]).

Cada renglón del archivo contiene información sobre un cliente: número de cliente, coordenadas x e y, tiempos de apertura y cierre de sus dos ventanas, demanda, tiempo de servicio, e indicador si es PICKUP o DELIVERY.

Debido a que para nuestro estudio solo consideramos el reparto de mercaderías, la especificación de PICK o DELIVERY en este formato se ignora.

Las distancias son euclideanas y el cálculo se realiza redondeando al entero más cercano.

El tiempo de traslado se calcula a partir de la distancia mediante un factor que se solicita a través de pantalla.

La cantidad de vehículos y su carga también serán solicitados en pantalla de forma de llevar el problema a un número finito de vehículos con capacidad finita.

Se piden además por pantalla los factores de ponderación para distancia y tiempo (ψ y φ).

Como ejemplo de este formato presentamos el problema 10T2.dat, resumido:

```
  0   50   50    0  960  960  960    0    0 0
  1   45    2    0   30   50   80   10    0 0
  2   48    2    0   30   50   80   10    0 0
  3   52    2    0   30   50   80   10    0 0
  4   55    2    0   30   50   80   10    0 0
 ...  ...  ... ...  ...  ...  ...  ...    ...
 99   52   98    0   30   50   80   10    0 0
100   55   98    0   30   50   80   10    0 0
```

3) trsolom

Son los problemas de Solomon (ref [30]). Estos problemas están divididos en tres categorías, identificadas por un prefijo en el nombre del archivo de datos. Los problemas cuyos nombres empiezan con “R” son aquellos en los que los clientes están distribuidos aleatoriamente de manera uniforme. Los que tienen prefijo “C” son los que presentan agrupaciones en el mapa de clientes. La combinación de ambos está dada por el prefijo “RC”.

El formato consiste en un título en primera línea que coincide con el nombre del archivo, y le siguen a esto las etiquetas VEHICLE, NUMBER y CAPACITY con la información debajo.

Luego viene la sección CUSTOMER que consta de siete columnas:

5) CUST NO. enumera los clientes del cero a n, con n múltiplo de 100. Se interpreta al cliente cero como el depósito.
6) XCOORD, YCOORD coordenadas x e y del cliente en el mapa.
7) DEMAND demanda del cliente.
8) SERVICE TIME tiempo de servicio.
9) READY TIME y DUE DATE apertura y fin de la única ventana de tiempo del cliente.

Las consideraciones con respecto al tiempo de traslado y las distancias fueron tomadas en forma análoga a los dos casos anteriores.

Igualmente, se piden por pantalla los factores de ponderación para distancia y tiempo (ψ y φ).

A continuación presentamos el problema c101.txt (resumido), como ejemplo de este formato:

```
C101

VEHICLE
NUMBER     CAPACITY
  25         200

CUSTOMER
CUST NO.  XCOORD.   YCOORD.    DEMAND   READY TIME  DUE DATE   SERVICE TIME

    0      40         50          0          0       1236          0
    1      45         68         10        912        967         90
    2      45         70         30        825        870         90
    3      42         66         10         65        146         90
    4      42         68         10        727        782         90
  ...     ...        ...        ...        ...        ...        ...
   99      55         80         10        743        820         90
  100      55         85         20        647        726         90
```

Ejecución del algoritmo

Para la ejecución del algoritmo se desarrolló el programa *vrp_ants* que permite correr el algoritmo varias veces y elegir la parametrización a utilizar.
El programa se ejecuta desde línea de comandos escribiendo:

```
vrp_ants
```

Luego se piden una serie de respuestas por parte del usuario, que permitirán configurar la ejecución: cantidad de corridas, realización o no de post-optimización, parámetros a utilizar. Por último se solicita el nombre del archivo que contiene el problema en formato ants.
El programa devolverá dos archivos de resultados: “result.dat” y “mejores.dat”. En el primer archivo se devuelven el costo y la cantidad de vehículos de cada corrida, más el costo promedio y mejor entre todas las corridas. En el segundo se devuelve información más detallada sobre las rutas obtenidas en cada corrida.

En el Apéndice C se puede ver un instructivo para la ejecución del programa *vrp_ants*.

7. TESTEO

Dividimos el trabajo de testeo en cuatro etapas.

- Ajuste de parámetros
- Comparación de resultados antes y después de agregar post-optimización
- Comparación con resultados obtenidos mediante otros métodos conocidos
- Demostración del correcto funcionamiento con múltiples ventanas

En las tres primeras etapas nos enfocaremos en probar con sets de problemas conocidos y ya probados anteriormente mediante otros métodos.
Debido a que no se ha experimentado anteriormente con sets de problemas con más de una ventana de tiempo, creamos nuestros propios problemas con múltiples ventanas para las pruebas del último punto.
Para las pruebas se utilizaron máquinas de dos tipos:

1) Procesador AMD-K6-2 500MHz, 56 MB RAM, Sistema Operativo Linux Red Hat
2) Procesador Pentium II 350MHz Intel, 64 MB RAM, Sistema Operativo Unix Solaris 7.0

7.1. Ajuste de parámetros

Nuestro trabajo consistió en realizar una serie de pruebas con distintos juegos de parámetros, para comparar los resultados obtenidos y poder definir la combinación de valores de parámetros más adecuada. En primer lugar, ajustamos los parámetros propios de ACS, y luego ajustamos el parámetro ω (introducido por nosotros).

7.1.1. Parámetros de ACS

Los parámetros que ajustamos fueron los siguientes: α, β, ρ y q_0.
Para la elección del rango de valores a testear para cada parámetro, consultamos la bibliografía sobre Ant System recabando información sobre tests ya realizados. Allí se recomiendan valores de: $1 \leq \alpha \leq 1.5$, $1 \leq \beta < 5$, $0.5 \leq \rho \leq 0.9$, y se muestra que para valores fuera de esos rangos los resultados son malos. Aunque para q_0 se recomienda 0.9 en la mayoría de los papers, creímos pertinente probar con otros valores porque con ese valor tan grande se da poca chance a la probabilidad y se puede caer en soluciones subóptimas.
En cuanto a los parámetros para rastro local, ξ y τ_0, elegimos los valores sugeridos en el paper MACS-VRPTW (ref[18]), que son $\xi = \rho$ y $\tau_0 = 1/ (n*Sol\_NN)$, con n igual a cantidad de clientes y Sol_NN igual a alguna solución obtenida mediante el método de Nearest Neighbours (ref [45]).
Obs: como para el problema de prueba (problema reducido) no disponemos de solución conocida, utilizamos la solución del problema completo.

Las pruebas fueron realizadas sobre el problema r112-20.txt. Este problema fue tomado del problema r112.txt de 100 clientes del set de 56 problemas de Solomon (ref [30]), y para disminuir los tiempos de corrida se redujo la cantidad de clientes a los primeros 20 del archivo, y la cantidad de vehículos a los primeros 5.

Dejamos entonces fijos los valores: m = 20, cant. iteraciones = 200, γ =1 y δ=1, $\tau_0 = 1/ (20*1600) = 3.12e\text{-}05$, y probamos los siguientes valores para los demás parámetros:

$\alpha = \{1, 1.3, 1.5\}$
$\beta = \{1, 3, 5\}$
$\rho = \xi = \{0.5, 0.7, 0.9\}$
$q_0 = \{0.1, 0.5, 0.9\}$

Son entonces 81 instancias de prueba, que dividimos en 9 casos para facilitar su visualización. Se realizaron 3 corridas del algoritmo para cada instancia de prueba, obteniendo en cada caso el promedio correspondiente.
Se utiliza tiempo de traslado entre dos clientes igual a la distancia y para el cálculo del costo se utilizaron factor de distancia y factor de tiempo iguales a 1. Por lo tanto los costos están formados por la suma de: distancias, tiempos de espera, tiempos de traslado y tiempos de servicio.

Caso 1) $\rho = 0.5$, $q_0 = 0.1$

$\alpha = 1$	1132.00	941.33	896.00
$\alpha = 1.3$	1116.17	926.67	893.30
$\alpha = 1.5$	1117.00	916.67	891.00
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 2) $\rho = 0.5$, $q_0 = 0.5$

$\alpha = 1$	955.00	904.00	891.00
$\alpha = 1.3$	955.67	909.67	888.33
$\alpha = 1.5$	954.67	890.33	888.67
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 3) $\rho = 0.5$, $q_0 = 0.9$

$\alpha = 1$	889.67	902.67	888.67
$\alpha = 1.3$	888.33	888.33	902.00
$\alpha = 1.5$	891.00	897.67	889.67
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 4) $\rho = 0.7$, $q_0 = 0.1$

$\alpha = 1$	1114.67	943.33	897.67
$\alpha = 1.3$	1140.33	921.67	909.67
$\alpha = 1.5$	1085.33	907.67	890.67
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 5) $\rho = 0.7$, $q_0 = 0.5$

$\alpha = 1$	1007.67	897.33	900.00
$\alpha = 1.3$	966.00	896.33	**886.00**
$\alpha = 1.5$	989.33	911.00	888.67
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 6) $\rho = 0.7$, $q_0 = 0.9$

$\alpha = 1$	890.00	887.00	900.67
$\alpha = 1.3$	891.00	892.33	902.00
$\alpha = 1.5$	910.33	888.33	900.67
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 7) $\rho = 0.9$, $q_0 = 0.1$

$\alpha = 1$	1128.00	977.67	890.67
$\alpha = 1.3$	1128.67	943.67	905.67
$\alpha = 1.5$	1130.33	905.67	900.33
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 8) $\rho = 0.9$, $q_0 = 0.5$

$\alpha = 1$	992.67	906.33	894.00
$\alpha = 1.3$	1049.33	905.67	913.33
$\alpha = 1.5$	1028.00	904.67	893.00
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Caso 9) $\rho = 0.9$, $q_0 = 0.9$

$\alpha = 1$	905.67	900.67	919.67
$\alpha = 1.3$	904.67	905.33	911.67
$\alpha = 1.5$	919.67	922.67	911.33
	$\beta = 1$	$\beta = 3$	$\beta = 5$

Análisis de resultados obtenidos

- Para valores de $q_0 = 0.1$ se obtuvieron resultados subóptimos para todas las combinaciones de los demás valores testeadas (casos 1, 4 y 7). Lo mismo sucedió para valores de ρ=0.9 (casos 7, 8 y 9). En la mayor parte de estos casos los mejores valores se obtuvieron para α=1.5 y β=5.
- Para $q_0 = 0.9$ (casos 3, 6 y 9) se obtienen valores cercanos al óptimo para todas las combinaciones de los demás parámetros, pero éste no es alcanzado. El resultado más cercano al óptimo se obtuvo para $\rho = 0.7$, $\alpha = 1$ y $\beta = 3$ (caso 6).
- Para $q_0 = 0.5$ (casos 2, 5 y 8) se obtienen malas soluciones cuando $\beta = 1$. Se alcanzó el óptimo para $\rho = 0.7$, $\alpha = 1.3$ y $\beta = 5$ (caso 5).

Concluímos entonces que los mejores valores estarían en los rangos:
$0.5 \leq \rho \leq 0.7$, $1.3 \leq \alpha \leq 1.5$, $3 \leq \beta \leq 5$ y $0.5 \leq q_0 \leq 0.9$

Para las pruebas siguientes elegimos los parámetros ρ=0.7, α=1.3, β=5 y q_0=0.5, por ser los que demostraron obtener mejores resultados.

7.1.2. Parámetro adicional (omega)

Luego de ajustar los parámetros del método ACS, comenzamos a probar los problemas del set de Solomon comparando las soluciones obtenidas con las de otros métodos. Al realizar estas pruebas notamos que las soluciones halladas por nuestro algoritmo no resultaban tan buenas como esperábamos; y al analizar las rutas de las soluciones, vimos que la causa aparente era que se volvía muchas veces al depósito. En ese entonces el criterio para volver al depósito era, al elegir el próximo cliente a visitar, considerar el depósito como un cliente más. Para solucionar esto, en el cálculo de la probabilidad de elección del depósito agregamos un factor más, cuyo efecto es que cuanto menor sea la carga restante del vehículo, mayor sea la probabilidad de elegir el depósito como próximo cliente a visitar (ver Sección 5.2. Decisiones de Diseño, Elección del próximo cliente a visitar). Para regular la influencia de dicho factor en comparación con los restantes (rastro y visibilidad), se lo eleva a un parámetro ω, que es el que ajustaremos en estas pruebas.
En la Tabla 1 se presentan los resultados obtenidos al variar ω, para tres problemas del set de Solomon (uno de cada grupo: C, R y RC).
Se utilizó para el cálculo del costo un peso de la distancia = 0 y un peso del tiempo = 1, por lo tanto los costos están compuestos por:
- tiempo de traslado } costo variable
- tiempo de esperas }
- tiempo de servicio costo fijo

El tiempo de servicio es un costo fijo, y es de 9000 para los sets C1 y C2, y 1000 para los demás sets.

	ω	costo	costo variable	cant.veh
C102	**0**	12051,67	**3051,67**	**15,00**
	2	11195,67	**2195,67**	**11,33**
	20	11195,67	**2195,67**	**11,33**
R112	**0**	2312,00	**1312,00**	**12,67**
	2	2242,67	**1242,67**	**11,00**
	20	2242,67	**1242,67**	**11,00**
RC202	**0**	4320,00	**3320,00**	**6,00**
	2	3066,33	**2066,33**	**4,00**
	20	3054,33	**2054,33**	**4,00**

Tabla 1 Comparación de resultados usando distintos valores de ω

Comparación de resultados utilizando distintos valores de ω

En la Figura 1 se muestra la comparación de la cantidad de vehículos para los distintos valores de omega utilizados, en los tres problemas de prueba. En la Figura 5 se muestra la misma comparación para los costos.

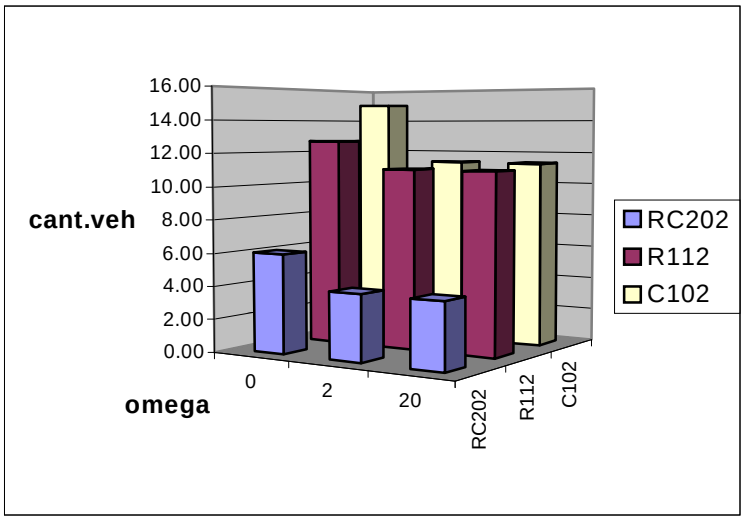


Fig. 1 Gráfico de comparación de cantidad de vehículos utilizados

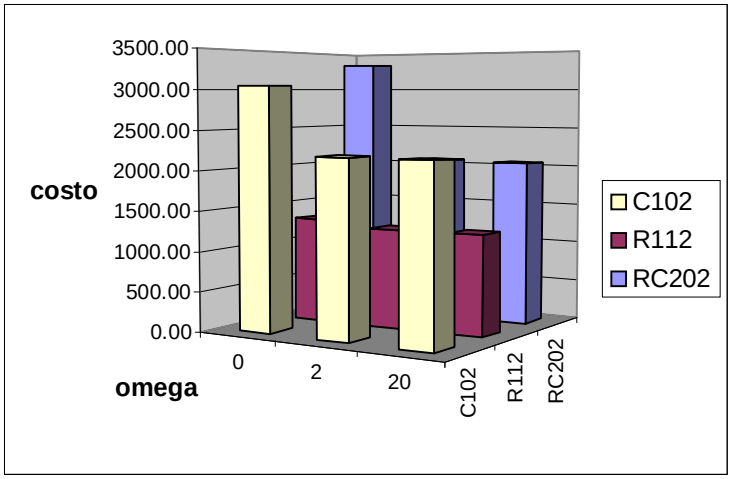


Fig. 2 Gráfico de comparación de costos

Análisis de resultados obtenidos

- Observamos en las Figuras 1 y 2 que al utilizar ω = 2 u ω = 20 logramos disminuir el costo y la cantidad de vehículos respecto a los obtenidos con ω = 0. En la Tabla 1 podemos ver que los costos variables se reducen hasta en un 40%, mientras que la cantidad de vehículos se reduce aproximadamente un 25% para la mayoría de los casos.

Por lo tanto, concluimos que la inclusión del nuevo factor para la decisión de volver al depósito fue una decisión acertada. Para las pruebas siguientes elegimos entonces ω = 20.

7.2. Pruebas de Optimización

Aquí comparamos los resultados obtenidos con el algoritmo Ants común y con el algoritmo Ants mejorado con post-optimización.
En la Tabla 2 presentamos los resultados comparativos para tres problemas, uno de cada grupo de Solomon (C, R y RC).

Al igual que en las pruebas de ajuste de ω, los costos están compuestos por:
- tiempo de traslado } costo variable
- tiempo de esperas }
- tiempo de servicio costo fijo (9000 para sets C1 y C2, y 1000 para el resto)

	Con/Sin Post-opt	costo	costo variable	cant.veh
C102	sin post-opt	11195.67	**2195.67**	11.33
	con post-opt	10746.33	**1746.33**	12.33
R112	sin post-opt	2242.67	**1242.67**	11.00
	con post-opt	2147.33	**1147.33**	10.67
RC202	sin post-opt	3054.33	**2054.33**	4.00
	con post-opt	2799.00	**1799.00**	4.00

Tabla 2 Comparación de resultados sin post-optimización y con post-optimización

Para algunos problemas comparamos el tiempo de ejecución de una corrida simple y de una con post-optimización. Por ejemplo para el problema R112.TXT, la corrida simple llevó 1413 segundos de tiempo de CPU y la corrida con post-optimización 1595 segundos. En otros problemas medidos, la diferencia entre ambos tiempos de CPU también se situaba aproximadamente en un 15% más para la post-optimización.

Comparación de resultados al incluir Post-optimización

En la Figura 3 se muestra la comparación de costos en los tres problemas de prueba, para el algoritmo simple y para el algoritmo con Post-optimización.

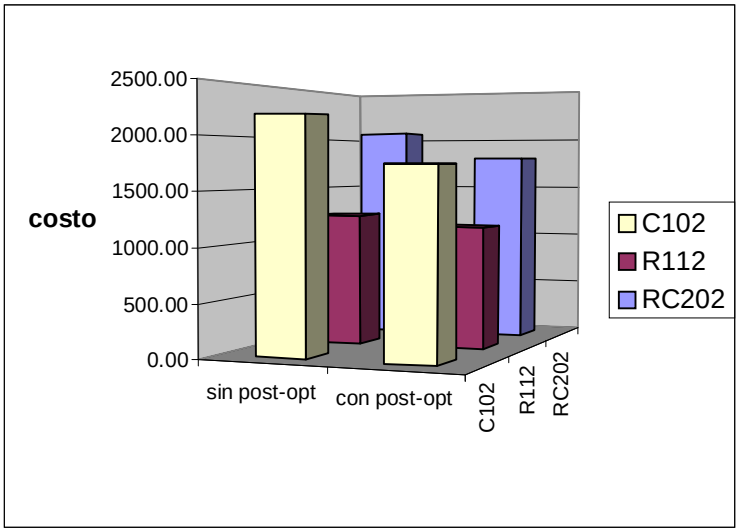


Fig. 3 Comparación de costos

Análisis de resultados obtenidos

- Observamos en la Figura 3, que agregando post-optimización en todos los casos se obtienen mejoras importantes en los costos variables. Mirando la Tabla 2, vemos que los costos variables disminuyen hasta en un 20%, mientras que no se observan diferencias notorias entre la cantidad de vehículos utilizados en uno y otro caso.

Concluimos entonces que es conveniente la utilización de Post-optimización, ya que se obtienen soluciones considerablemente mejores, mientras que el costo computacional extra resulta razonable.

7.3. Pruebas de Performance

En primer lugar realizamos una serie de pruebas para comparar los resultados obtenidos por nuestro algoritmo, con los resultados de otros algoritmos conocidos. Luego, para un conjunto representativo de problemas, medimos el tiempo de ejecución de nuestro algoritmo, para evaluar el costo computacional.

7.3.1. Comparación de resultados

Realizamos una serie de corridas sobre el set de 56 problemas de Solomon, comparando los resultados obtenidos con los del método "Insertion I" (ref [45]), y con los del algoritmo Haskell utilizado por el Taller5 "Ruteo de Vehículos con Dos Ventanas de Tiempo" (ref [41]).
Para estas pruebas se utilizaron los siguientes valores de parámetros: cant.hormigas=100, cant. iteraciones=200, ρ=0.7, α=1.3, β=5, q_0=0.5, ω=20, γ=1, δ=1, τ_0 = 1/ (20*SolNN).
Se realizaron 3 corridas para cada problema de prueba (56 problemas en total), obteniendo para cada problema el promedio y la mejor solución. Los costos se calcularon como la suma de distancias, esperas y tiempos de atención. Se asume que el tiempo de traslado es igual a la distancia, y para evitar que este factor se incluya dos veces en cálculo del costo que hace el algoritmo, se utilizan peso distancia = 0 y peso tiempo =1.

En la Tabla 3 presentamos los resultados obtenidos para cada problema de prueba, y también los promedios por grupo de problema. Recordamos que los grupos son C ("clustered" o aglomerados), R ("regular" o uniformes) y RC (mezcla de los dos anteriores).

	MEJOR				PROMEDIO	
PROB	**costo**	**cant. veh**	**dist.**	**esperas**	**costo**	**cant.veh**
C101	10211	12	1207	4	10235.33	11.33
C102	10639	12	1606	33	10712.33	12.00
C103	10895	12	1855	40	10993.00	12.67
C104	10786	11	1695	91	10799.00	11.33
C105	10420	11	1329	91	10505.33	11.67
C106	10584	12	1575	9	10686.67	12.00
C107	10664	12	1664	0	10700.33	12.33
C108	10481	12	1481	0	10577.33	11.67
C109	10515	11	1515	0	10557.33	11.00
prom.C1	**10577.22**	**11.67**	**1547.44**	**29.78**	**10640.74**	**11.78**
C201	9692	4	692	0	9725.00	4.00
C202	10714	4	1476	238	10769.00	4.33
C203	10917	5	1772	145	11017.70	5.00
C204	10714	4	1556	158	10733.00	4.00
C205	9806	4	795	11	9921.33	4.33
C206	9947	4	947	0	10010.00	4.33
C207	9905	4	905	0	9952.33	4.00

C208	9917	4	917	0	9975.33	4.00
prom.C2	**10201.50**	**4.13**	**1132.50**	**69.00**	**10262.96**	**4.25**
R101	3032	19	1881	151	3080.00	20.00
R102	2959	21	1835	124	2974.00	20.33
R103	2733	18	1649	84	2767.00	17.33
R104	2367	13	1352	15	2380.67	14.00
R105	2589	15	1573	16	2603.33	15.00
R106	2526	15	1502	24	2546.33	15.00
R107	2366	14	1347	19	2422.33	13.67
R108	2189	12	1189	0	2228.67	12.00
R109	2392	14	1392	0	2427.00	13.67
R110	2285	12	1285	0	2334.33	13.00
R111	2344	12	1344	0	2388.00	13.33
R112	2119	11	1119	0	2156.33	11.00
prom.R1	**2491.75**	**14.67**	**1455.67**	**36.08**	**2525.67**	**14.86**
R201	2724	4	1720	4	2735.67	4.00
R202	2575	4	1551	24	2607.33	4.00
R203	2414	4	1414	0	2435.67	4.00
R204	2062	3	1060	2	2100.33	3.00
R205	2312	3	1312	0	2342.33	3.00
R206	2209	3	1209	0	2220.67	3.00
R207	2097	3	1097	0	2125.67	3.00
R208	1880	3	880	0	1927.67	3.00
R209	2219	3	1179	40	2179.00	3.00
R210	2347	3	1347	0	2356.00	3.33
R211	1934	3	934	0	1978.00	3.00
prom.R2	**2252.09**	**3.27**	**1245.73**	**6.36**	**2273.49**	**3.30**
RC101	3003	18	1983	20	3018.67	17.33
RC102	2835	15	1786	49	2881.00	16.00
RC103	2649	14	1647	2	2686.67	14.00
RC104	2470	12	1457	13	2479.67	12.33
RC105	3041	18	2011	30	3046.67	18.00
RC106	2616	14	1616	0	2639.33	14.33
RC107	2531	14	1531	0	2557.33	13.00
RC108	2411	12	1411	0	2422.67	12.00
prom.RC1	**2694.50**	**14.63**	**1680.25**	**14.25**	**2716.50**	**14.62**
RC201	2913	5	1893	20	2951.67	4.67
RC202	2749	4	1714	35	2799.00	4.00
RC203	2444	4	1444	0	2459.33	4.00
RC204	2101	4	1097	4	2143.67	3.33
RC205	2877	4	1871	6	2933.33	4.67
RC206	2420	3	1411	9	2454.00	3.33
RC207	2393	4	1393	0	2418.33	3.67
RC208	2093	3	1093	0	2107.00	3.00
prom.RC2	**2498.75**	**3.88**	**1489.50**	**9.25**	**2533.29**	**3.83**

Tabla 3 Resultados obtenidos por nuestro algoritmo, para cada problema de prueba de Solomon

En la Tabla 4 comparamos los promedios de las mejores soluciones que obtuvimos para cada set (tomándolos de Tabla 3), con las soluciones obtenidas mediante la utilización de los demás métodos.
La última fila corresponde a los promedios de todos los problemas.

		Insertion I (Solomon)	Taller Haskell	Ants (nuestro)
C1	Veh	**10**	**10.89**	**11.67**
	Sch	**10104.2**	**10579.45**	**10577.22**
	Dist	951.9	958.67	1547.44
	Esp	152.3	620.78	29.78
C2	Veh	**3.1**	**4.13**	**4.13**
	Sch	**9921.3**	**11370.88**	**10201.50**
	Dist	692.7	784.38	1132.50
	Esp	228.6	1586.5	69.00
R1	Veh	**13.6**	**14.75**	**14.67**
	Sch	**2695.5**	**2630.33**	**2491.75**
	Dist	1436.7	1373.58	1455.67
	Esp	258.8	256.75	36.08
R2	Veh	**3.3**	**4.64**	**3.27**
	Sch	**2578**	**3516.09**	**2252.09**
	Dist	1402.4	1196.82	1245.73
	Esp	175.6	1319.27	6.36
RC1	Veh	**13.5**	**14.88**	**14.63**
	Sch	**2775**	**2760.38**	**2694.50**
	Dist	1596.5	1598.5	1680.25
	Esp	178.5	161.88	14.25
RC2	Veh	**3.9**	**6**	**3.88**
	Sch	**2955.3**	**4259.38**	**2498.75**
	Dist	1682.1	1407.25	1489.50
	Esp	273.2	1852.13	9.25
TOTAL	Veh	**7.90**	**9.22**	**8.71**
	Sch	**5171.55**	**5852.75**	**5119.30**
	Dist	**1293.72**	**1219.87**	**1425.18**
	Esp	**211.17**	**966.22**	**27.45**

Nota:
Veh = cantidad de vehículos
Sch = “schedule time” o costo
Dist = distancia = tiempo de traslado
Esp = esperas

Tabla 4 Resultados para cada set de prueba de Solomon, para los métodos InsertionI, Haskell y Ants

Comparación de resultados promedios

En la Figura 4 se muestra la comparación entre los costos, distancias y esperas para los tres métodos; mientras que en la figura 5 se muestran la comparación de la cantidad de vehículos.

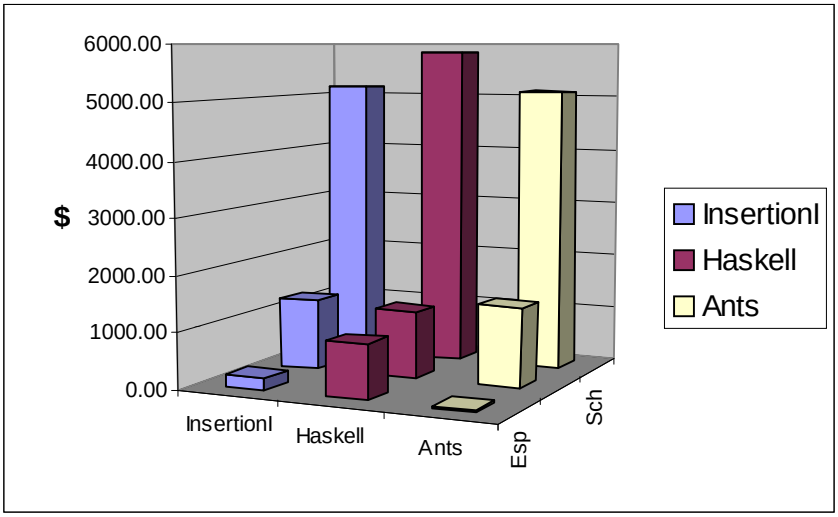


Fig. 4 Comparación de costos, distancias y esperas

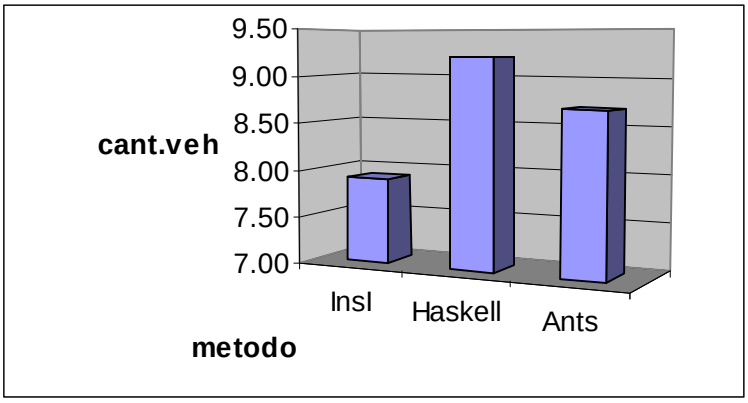


Fig. 5 Comparación de Cantidad de vehículos utilizados

Análisis de resultados obtenidos

Como se ve en los totales de la Tabla 4, en promedio se lograron mejorar los costos respecto a los algoritmos comparados, debido a una importante disminución en las esperas, que logró contrarrestar el hecho de que las distancias obtenidas no mejoraran respecto a los demás métodos.
En particular, para todos los sets se obtienen mejores costos que Haskell y para cuatro de los seis sets se obtienen mejores costos que Insertion I. A su vez, para todos los sets

se obtienen mejores esperas que en los dos algoritmos comparados; y en los sets R2 y RC2 incluso se obtienen mejores distancias que el método Insertion I.
Atribuimos la importante disminución de esperas a la contribución de tres grandes factores: el considerar las esperas al momento de elegir el próximo cliente a visitar; la utilización del algoritmo de ajustar tiempos (que se corre luego de la Optimización local or-opt, que también tiene en cuenta las esperas al reordenar los clientes); y a que se aprovecha la retroalimentación del método ACS al aplicar esta post-optimización en cada iteración (por lo tanto se utiliza la solución post-optimizada para la actualización de rastros).
En cuanto a la cantidad de vehículos de cada solución, vemos que en promedio se utilizan menos vehículos que en el algoritmo Haskell, y más que en el algoritmo Insertion I. Se debe tener en cuenta que a diferencia de este último algoritmo, el nuestro no fue diseñado con el objetivo de priorizar la minimización de la cantidad de vehículos utilizados; aunque gracias a la inclusión del parámetro ω (ver Sección 7.1.2), se lograron obtener buenos resultados para esta variable.

7.3.2. Tiempo de ejecución

Al realizar las pruebas de la Tabla 3, medimos los tiempos de ejecución de nuestro algoritmo para algunos problemas representativos, para tener una idea del costo computacional que éste implica. Los tiempos fueron medidos en una máquina con procesador AMD-K6-2 500MHz, 56 MB RAM, Sistema Operativo Linux Red Hat.
En la Tabla 5 presentamos los tiempos de ejecución obtenidos para el set R de Solomon.

Problema	Tiempo CPU (s)
R101	2055.57
R102	1369.25
R103	881.50
R104	1321.81
R105	1425.73
R106	1037.07
R107	1078.81
R108	1344.87
R109	1321.75
R110	1224.27
R111	1094.62
R112	1417.55
Promedio	**1126.44**

Tabla 5 Tiempos de CPU (en segundos) para los problemas del set R de Solomon

Como vemos en la Tabla 5, para el set R cada problema tardó en promedio 1126 segundos de tiempo de CPU. No fue posible comparar este tiempo con otros métodos, debido a que en la bibliografía analizada no había información de este tipo en la

mayoría de los casos. Para algunos casos sí se indicaba el tiempo de CPU, pero éste no era comparable con el nuestro pues se utilizaban máquinas distintas.

7.4. Pruebas con Múltiples Ventanas

Debido a que no se ha experimentado anteriormente con sets de problemas con más de una ventana de tiempo, no tenemos referencias en las cuales basarnos para realizar pruebas comparativas de la eficiencia de nuestra solución frente a otras propuestas.
Sin embargo, disponemos de dos formas de comprobar la eficacia del método, que se plantean a continuación:

Ventanas de tiempo continuas

Tomamos un problema del set de Solomon (RC107) y lo modificamos agregando ventanas a un conjunto de clientes elegidos.
Para poder comprobar que el algoritmo se comporta correctamente para múltiples ventanas, para los clientes elegidos tomamos su ventana original y la partimos en n ventanas de tal forma que la apertura de la primer ventana coincida con la apertura de la ventana original y el cierre de la última ventana coincida con el de la ventana original. Además, para no se modifiquen las horas de visitado de dichos clientes se tomaron apertura ventana (i) = cierre ventana (i-1), para $1 \leq i \leq (n-1)$.
Como los resultados del algoritmo de post-optimización "ajustar_tiempos" dependen de la hora de apertura y cierre de la ventana de visitado del cliente, no corrimos post-optimización para estas pruebas porque los resultados no serían comparables.
En la Tabla 6 se muestran la forma en que fueron modificadas las ventanas, para los clientes elegidos.

Nro. Cliente	Cant. Vent.	apert.	cierre	apert.	cierre	apert.	cierre	apert.	cierre
2	2	32	50	50	97	-	-	-	-
4	3	71	90	90	150	150	193	-	-
6	4	55	70	70	80	80	100	100	164

Tabla 6 Clientes con ventanas modificadas

	costo	Cant.veh	cliente 2		cliente 4		cliente 6	
			hora vis.	vent.vis	hora vis.	vent.vis	hora vis.	vent.vis
RC107	2718	13	32	0	179	0	97	0
RC107 mult.vent.	2718	13	32	0	179	2	97	2

Tabla 7 Resultados Obtenidos

En la Tabla 7 comparamos los resultados obtenidos para el problema con los obtenidos para el problema modificado. Los resultados fueron iguales en cuanto a costo, cantidad de vehículos y horas de visitado para ambos problemas; cambiando solamente los números de ventana de visitados de aquellos clientes con múltiples ventanas, como era de esperar.

Ventanas de tiempo que incluyen alguna solución conocida

Elegimos arbitrariamente dos problemas del set de Solomon (C202, RC107) y con cada uno de ellos procedimos de la siguiente manera: para algunos de los 100 clientes del set observamos la hora de visitado que les correspondía según alguna de las posibles soluciones encontradas al correr el algoritmo. Para esos clientes dividimos el espacio de tiempo asegurándonos que la hora de visitado de ese cliente para la solución elegida estuviera contenida en alguna de las ventanas de tiempo agregadas.
Formalmente: si llamamos h_j a la hora de visitado del cliente j para una solución S, subdividimos el tiempo en k ventanas condicionado a que:
apertura ventana (i) $\leq$ h_j $\leq$ cierre ventana (i) para alguna ventana i tal que $0 \leq i \leq (k-1)$.

La idea básica de estas pruebas es que partiendo del conocimiento de que existe una solución para ese problema original en el cual hay una única ventana de tiempo, disponemos de esa solución, y a su vez esta solución se encuentra en el espacio de soluciones del nuevo problema con múltiples ventanas (por definición del problema), entonces, es de esperar que si no se halla otra solución se encuentre la conocida.
La Tabla 8 muestra para el problema C202, la hora de visitado de cada cliente elegido en el caso de una única ventana (según resultados obtenidos anteriormente); y en el resto de las columnas la división en varias ventanas realizada por nosotros.

Nro. Cliente	Hora visitado	Cant. Vent.	apert.	cierre	apert.	Cierre
88	1597	2	1546	1670	1690	1706
51	2291	2	2200	2295	2300	2360
16	2029	2	550	1900	1934	2094
36	1253	2	0	1250	1253	1413
55	1202	2	1065	1225	1300	3000

Tabla 8 Hora de visitado de los clientes según solución para el caso una sola ventana del problema C202, y división de las ventanas

Realizamos tres corridas para cada uno de los problemas. Para el problema C202 comparamos los resultados obtenidos al correr el algoritmo con post-optimización, para el caso de una única ventana y el caso de múltiples ventanas. La Tabla 9 presenta el mejor costo y la cantidad de vehículos obtenidos en las pruebas realizadas.

	costo	Cant.veh
Unica ventana	10714	4
Múltiples ventanas	10556	4

Tabla 9 Mejor costo obtenido para C202, problema original y con múltiples ventanas

Análogamente, para el problema RC107, primero obtuvimos una solución para el caso de una única ventana, y en base a la misma dividimos en múltiples ventanas (Tabla 10).

Nro. Cliente	Hora visitado	Cant. Vent.	apert.	cierre	apert.	cierre	apert.	cierre
100	206	2	160	185	199	210	-	-
21	52	2	50	60	75	99	-	-
97	149	3	100	135	140	145	147	315
32	169	3	150	160	162	165	167	300
13	77	3	2	60	63	80	101	145
43	123	3	100	125	128	140	150	210

Tabla 10 Hora de visitado de los clientes según solución obtenida para el caso de una única ventana del problema RC107, y división de las ventanas.

Para este problema, comparamos los resultados obtenidos al correr el algoritmo sin post-optimización, para el caso de una única ventana y el caso de múltiples ventanas. Los resultados se presentan en la Tabla 11.

	costo	Cant.veh
Unica ventana	2680	12
Multiples ventanas	2558	12

Tabla 11 Mejor costo obtenido para RC107, problema original y con múltiples ventanas

Para los casos probados en esta instancia se encontraron nuevas soluciones (tanto para el caso que incluye post-optimización como para el que no), y esto es justificable debido a la lógica de nuestro algoritmo. Esto es porque un factor que se tiene en cuenta cuando se va a elegir el próximo cliente a visitar, es la apertura de la ventana de tiempo del mismo en relación al horario del vehículo que está haciendo el recorrido. Debido a eso, al realizar la división de ventanas provocamos que las hormigas realizaran recorridos diferentes y condujeran a resultados muy diferentes, e incluso mejores en algún caso.

Concluimos entonces en base a las pruebas realizadas, que el algoritmo se comporta correctamente en la resolución de problemas con múltiples ventanas.

8. CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentan las conclusiones generales del proyecto y los posibles trabajos futuros a realizarse.

8.1. Conclusiones Generales

En este proyecto se ha estudiado la aplicación de un método Ant System para la resolución del problema VRP-MTW. Se ha implementado una biblioteca de funciones en lenguaje C++ para la representación de vehículos, clientes y la ubicación de los mismos (mapa de la ciudad). A su vez se implementó un algoritmo Ant System que encuentra la solución al problema de ruteo utilizando las funciones mencionadas. Se han generado casos de prueba para la validación y comparación de la calidad de las soluciones obtenidas con respecto a otros algoritmos conocidos. También se ha construido un sitio web para la difusión del trabajo realizado.

El trabajo de este taller consistió en tres grandes etapas:
- relevamiento de información y adaptación de una solución Ant System para el problema VRP-mTW.
- implementación del algoritmo diseñado.
- validación de los resultados.

Para la resolución de nuestro problema desarrollamos un formalismo ACO que realiza la búsqueda de caminos; y luego agregamos una post-optimización que mejora las soluciones halladas por dicho formalismo.

En la fase de pruebas nos ocupamos en primer lugar de realizar un ajuste de los parámetros que utiliza el algoritmo. Luego de obtenidos los valores óptimos para dichos parámetros elegimos algunos casos de prueba representativos y comparamos las soluciones brindadas por el algoritmo original y el algoritmo con post-optimización.
Nuestra solución da al usuario la opción de ejecutar o no la post-optimización. Sin embargo, basándonos en los resultados obtenidos, y el poco costo adicional que implica correr la post-optimización, recomendamos la ejecución de la misma. Por otra parte la bibliografía estudiada también aconseja la utilización del algoritmo Ant System en combinación con diversos métodos de post-optimización.
En la siguiente etapa de pruebas tomamos un amplio conjunto de problemas y comparamos los resultados obtenidos por nuestro algoritmo con los obtenidos con otros métodos conocidos. La conclusión a la que llegamos fue que nuestro algoritmo logró obtener, en promedio, mejores costos que los demás algoritmos comparados.

8.2. Trabajo Futuro

Los trabajos futuros que consideramos sería interesante realizar son:
- paralelización del algoritmo.
- inclusión de una interfase con Sistemas de Información Geográfica.

Por otra parte, una posible mejora al algoritmo implementado sería darle la opción al usuario de elegir un criterio más complejo para ordenar las mejores soluciones. Por ejemplo un criterio podría ser ordenar las soluciones por menor cantidad de vehículos, y

ante soluciones de igual costo dar preferencia a las que tienen menor distancia o menor tiempo de espera.

Paralelización del algoritmo

La búsqueda de una solución paralela al problema de este Taller estaría motivada por la obtención de menores tiempos de ejecución, así como también la obtención de mejores soluciones en un menor tiempo para problemas más grandes (es decir con mayor cantidad de clientes).
Concretamente el objetivo a alcanzar sería el permitir la generación de razonablemente buenas soluciones para instancias de problemas grandes, en tiempos también razonables.
Según la bibliografía y las pruebas realizadas en este trabajo, se comprueba que la calidad de la solución obtenida crece cuando el número de hormigas involucradas en el problema crece (hasta un óptimo que es considerado ser igual al número de clientes).
De allí se deduce que para obtener buenas soluciones es necesario que aumente la complejidad del problema (en términos de tiempo de ejecución debido a la cantidad de hormigas), a medida que aumenta el tamaño del problema. Estas características señalan al problema como candidato a la paralelización. En el Apéndice D se puede encontrar un análisis más detallado sobre Paralelización y su posible aplicación a nuestro problema.

Inclusión de una interfase con Sistemas de Información Geográfica

Alternativamente a nuestra elección de hacer un sistema que consta básicamente de una biblioteca de funcionalidades, sería de interés orientarlo hacia un sistema visual respaldado por un software de gran potencialidad como es un GIS (Geographic Information System).
Un GIS provee al usuario de las utilidades de un mapa, es decir, extraer información de diferente tipo (calles, vegetación, etc), adicionando a esto el hecho de que los datos están almacenados en una computadora, lo que hace posible el análisis y la modelación.
Es decir, con las utilidades visuales que provee el GIS y nuestro algoritmo de ruteo como proceso “batch”, el usuario podría interactuar con el sistema en forma dinámica a través de la pantalla en vez de hacerlo a través de archivos. En el Apéndice E se puede encontrar un análisis más detallado sobre los GIS y su posible adaptación como interfase de nuestro algoritmo.

9. BIBLIOGRAFÍA

En la primer etapa de este proyecto nos dedicamos a recabar información sobre el formalismo Ant System.
Nuestra búsqueda se centró en relevar literatura en Internet y en la biblioteca del InCo de la Facultad de Ingeniería. Debido a que Ant System es un tema de investigación bastante reciente, Internet fue la principal fuente de información utilizada.

9.1. Internet

Papers sobre TSP y VRP

[1]. Dorigo, Maniezzo, Colorni (1991). "Ant System: An Autocatalytic Optimizing Process" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[2]. Colorni, Dorigo, Maniezzo (1991). "Distributed Optimization by Ant Colonies" http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html
Documento consultado en marzo 2000.

[3]. Colorni, Dorigo, Maniezzo (1992). "An investigation of some properties of an Ant Algorithm" http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html
Documento consultado en marzo 2000.

[4]. Bramel, Simchi-Levi (1993). "Probabilistic Analyses and Practical Algorithms for the Vehicle Routing Problem with Time Windows"
http://www.uwasa.fi/~d74302/research.html
Documento consultado en marzo 2000.

[5]. Antes, Derigs (1995). "A new Parallel Tour Construction Algorithm for the Vehicle Routing Problem with Time Windows" http://www.uwasa.fi/~d74302/research.html
Documento consultado en marzo 2000.

[6]. Dorigo, Gambardella (1996). "A study of some properties of Ant-Q" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html
Documento consultado en marzo 2000.

[7]. Dorigo, Gambardella (1996). "Ant Colonies for the Traveling Salesman Problem" http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html
Documento consultado en marzo 2000.

[8]. Dorigo, Maniezzo, Colorni (1996). "The Ant System: Optimization by a colony of cooperating agents" http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html
Documento consultado en marzo 2000.

[9]. Gambardella, Dorigo (1996). "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem"
http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

Documento consultado en marzo 2000.

[10]. Stützle, Hoos (1996). "Improvements on the Ant System: Introducing the MAX-MIN Ant System" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[11]. Stützle, Hoos (1996). "MAX-MIN Ant System and Local Search for the Traveling Salesman Problem" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[12]. Bullnheimer, Hartl, Strauss (1997). "Applying the Ant System to the Vehicle Routing Problem" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[13]. Bullnheimer, Hartl, Strauss (1997). "An improved ant system algorithm for the Vehicle Routing Problem" http://www.bwl.univie.ac.at/bwl/prod/hartl/index.htm#papers
Documento consultado en marzo 2000.

[14]. Dorigo, Gambardella (1997). "Ant Colony System: A cooperative learning approach to the Travelling Salesman Problem"
http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[15]. Stützle, Hoos (1997). "Improving the Ant System: A Detailed Report on the MAX-MIN Ant System"
http://www.kimes.inferenzsysteme.informatik.tu_darmstadt.de/~tom/pub.html
Documento consultado en marzo 2000.

[16]. Stützle, Dorigo (1999). "ACO Algorithms for the Traveling Salesman Problem"
http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html#ACO_meta
Documento consultado en marzo 2000.

[17]. Taillard (1999). "Ant Systems"
http://www.sunst50.einev.ch/prive/pro/taillard/articlesdir/articles.html
Documento consultado en marzo 2000.

[18]. Gambardella, Taillard, Agazzi (1999). "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows"
http://www.idsia.ch/~luca/macs-vrptw
Documento consultado en mayo 2000.

[19]. Gehring, Homberger. "A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows" http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm
Documento consultado en marzo 2000.

Papers sobre otros problemas de optimización

[20]. Gambardella, Taillard, Dorigo (1996) "Ant Colonies for the QAP"
http://sunst50.einev.ch/prive/pro/taillard/articles.dir/articles.html
Documento consultado en marzo 2000.

[21]. Forsyth, Wren (1997). "An Ant System for Bus Driver Scheduling" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html
Documento consultado en marzo 2000.

[22]. Gambardella, Dorigo (1997). "HAS-SOP: Hybrid Ant System for the Sequential Ordering Problem" http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html
Documento consultado en marzo 2000.

[23]. Stützle (1997). "MAX-MIN Ant System for the Quadratic Assignment Problem" http://iridia.ulb.ac.be/~stuetzle/pub.html#ACO
Documento consultado en marzo 2000.

[24]. Taillard, Gambardella (1997). "An Ant Approach for Structured Quadratic Assignment Problems" http://sunst50.einev.ch/prive/pro/taillard/articles.dir/articles.html
Documento consultado en marzo 2000.

[25]. van der Zwaan, Marques (1997). "Ant Colony Optimization for Job Shop Scheduling"

[26]. Tao, Michalewicz (1998). "Inver-Over Operator for the TSP" http://www.coe.uncc.edu/~zbyszek/papers.html
Documento consultado en marzo 2000.

[27]. Baüer, Bullnheimer, Hartl, Strauss (1999). "Minimizing Total Tardiness on a Single Machine using Ant Colony Optimization" http://www.bwl.univie.ac.at/bwl/prod/hartl/index.htmpapers
Documento consultado en marzo 2000.

[28]. Leguizamón, Michalewicz (1999). "A new version of Ant System for Subset Problems" http://www.coe.uncc.edu/~zbyszek/papers.html
Documento consultado en marzo 2000.

Problem sets

[29]. Instances. http://www-apache.imag.fr/~paugerat/VRP/INSTANCES/
Consultado en agosto 2000

[30]. Solomon. http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm
Consultado en setiembre 2000

[31]. Alex Van Breedam's VRPLIB. http://zeus.ruca.ua.ac.be/TEW/PHP/publicat.htm
Consultado en agosto 2000

[32]. TSPLIB. http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/
Consultado en setiembre 2000

[33]. ZIB's VRPLIB. ftp://ftp.zib.de/pub/Packages/mp-testdata/vehicle-rout/vrplib/index.html
Consultado en setiembre 2000

Páginas Web

[34]. Travelling Salesman Problem by Neural Approaches.
http://wayne.cs.nthu.edu.tw/~roland/nn/index.html
Introducción a heurísticas para TSP, profundizando en Neural Network y Ant System. Roland Fox, Huei-Juen Lin, Fang-Yi Jiang (1998).

[35]. An Introduction to Ant Colony Algorithms.
http://www.cogs.susx.ac.uk/users/johannf/ant/ants.html
Johann Bragi Fjalldal, 1999.

[36]. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows
http://www.idsia.ch/~luca/macs-vrptw
PhD estudent Position, Post Doctoral Position. Gambardella, Taillard, Agazzi. Universidad Istituto Dalle Molle di Studi sull' IntelligenzaArtificiale, Lugano, Switzerland (IDSIA), 1999.

[37]. Ant Colony Metaheuristic Applied to On-line Optimization Problems
http://www.idsia.ch/~luca/AntPHD99.html
PhD Position Available. Gambardella, Universidad Istituto Dalle Molle di Studi sull' IntelligenzaArtificiale, Lugano, Switzerland (IDSIA), 1999.

[38]. U.S. Geological Survey.
http://www.usgs.gov
Información sobre Sistemas de Información Geográfica, consultado en marzo 2001.

9.2. Biblioteca del InCo

Informes de proyectos de años anteriores

[39]. Marcelo Lorenzo, Fabián Mota, Fernando Nozar (1997). "TSP-TSPTW-TSPmTW-Ant Systems". Informe Final Taller V. Facultad de Ingeniería, Universidad de la República Oriental del Uruguay.

[40]. Gabriel Rodríguez, Diego Caggiano (1997). "Ruteo de Vehículos Utilizando Colonias de Agentes Cooperativos". Informe Final Taller V. Facultad de Ingeniería, Universidad de la República Oriental del Uruguay.

[41]. Lucía Ferraro, Lourdes Percíbale (2000). "Ruteo de Vehículos con Dos Ventanas de Tiempo". Informe Final Taller V. Facultad de Ingeniería, Universidad de la República Oriental del Uruguay.

Libros

[42]. Stefan Voss, Silvano Martello et al (1999). "Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization". Editorial Kluwer.

[43]. Mitchael R. Garey, David S. Johnson (1979). "Computers and intractability : a guide to the theory of NP completeness". Editorial Freeman.

Artículos

[44]. Gilbert Laporte (1992). " The Travelling Salesman Problem: An overview of exact and approximate algorithms". European Journal of Operational Research 59 (1992) 231-247 North-Holland

[45]. Marius M. Solomon (1985). "Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows constraints". Operations Research, Vol.35, Nº 2 (1987), Operations Research Society of America.

APÉNDICE A: ESTADO DEL ARTE

Dentro de los Ant Algorithms, existen distintas variantes además de la opción que hemos elegido para la solución al problema VRP-mTW.
En este apéndice se presentan las alternativas no elegidas dentro de los algoritmos ACO estudiados y dentro de las mejoras aplicables a VRP. Por último se presenta un análisis realizado sobre una de las soluciones Ant System aplicada a un problema similar al nuestro.

1. Otros algoritmos ACO

Además del método ACS elegido, existen varios algoritmos ACO que introducen mejoras a Ant System. A continuación describimos algunos de ellos:

MAX-MIN **Ant System-MMAS** (ref: [10], [11], [15] y [16])

Es una mejora directa sobre Ant System, en la cual solo la mejor hormiga actualiza los rastros (como en ACS).
Al observar que se obtenían buenos resultados para el A.S. cuando solo se les permitía actualizar los rastros a las hormigas con mejores tours, se concluyó que la fuerza de los rastros debería tener más importancia. Así, se introducen un máximo y un mínimo de rastro en los arcos (τ^{max} , τ^{min}), de ahí el nombre *MAX-MIN*. Esto permite además evitar el estancamiento de la búsqueda (cuando todas las hormigas hallan el mismo camino y no se encuentran nuevas soluciones).

Actualización de rastros:

$\tau_{ij} := \rho\tau_{ij} + \Delta\tau_{ij}^{best}$
Si $\tau_{ij} < \tau^{min}$
    $\tau_{ij} := \tau^{min}$
sino si $\tau_{ij} > \tau^{max}$
    $\tau_{ij} := \tau^{max}$
fin

donde $\Delta\tau_{ij}^{best} = 1/L^{best}$, y L^{best} es el largo del mejor camino (puede ser global best solution o iteration best solution).

El rango de intensidad de rastro hace que haya un rango de probabilidades (p_{min},p_{max}) para p_{ij}, donde $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$.
El rastro de cada arco es inicializado en τ^{max} al comenzar la primer iteración.

Si el rastro es muy alto en ciertos arcos, a largo plazo se va a reducir el espacio de búsqueda de esos arcos, evitando que se exploren otros. Para evitar esto se introduce el factor de ramificación "λ–branching factor", presentado por primera vez en [9].

Suavizamiento de rastros:
Cuando el factor λ es muy chico se ajustan las intensidades de los rastros de acuerdo a una actualización proporcional: el rastro en el arco (i,j) es incrementado proporcionalmente a la diferencia entre $\tau_{max} - \tau_{ij}(t)$.

Hormigas elitistas-$AS_{elitistas}$ (ref: [1])

Fue la primer mejora que se introdujo al Ant System. La idea es dar un refuerzo adicional a los arcos pertenecientes al mejor camino hallado desde el comienzo del algoritmo (global best solution o gb). Esto se logra actualizando los rastros de los arcos de dicho camino como si σ hormigas (llamadas hormigas elitistas), los hubieran usado. Es necesario encontrar el número σ adecuado, ya que si se elige muy grande la búsqueda se concentra en torno a soluciones subóptimas y eso lleva a un estancamiento de la búsqueda.
El rastro de los arcos se actualiza según:

$$\tau_{ij}^{new} = \rho.\tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \sigma \, . \, 1/L^{gb}$$

Hormigas elitistas rankeadas-AS_{rank} (ref: [13], [16])

Al igual que en hormigas elitistas todos los arcos pertenecientes a la mejor solución L^{gb} se incrementan como si σ hormigas los hubieran usado. Además se ranquea las hormigas elitistas según la calidad de su solución y se les permite actualizar el rastro en un valor ponderado de acuerdo a su ranking.
La fórmula queda:

$$\tau_{ij}^{new} = \rho.\tau_{ij}^{old} + \sum_{r=1}^{\sigma - 1} (\sigma - r) / L^{r} + \sigma \, . \, 1 / L^{gb}$$

2. Otras mejoras aplicables al problema VRP

Ahorro

En la bibliografía estudiada se presentan dos variantes de este método como una mejora a los Ant Algorithms para VRP.

Ahorro y capacidad (ref: [12])

En VRP no solo importa la locación relativa de dos clientes, la locación relativa con respecto al depósito es también esencial para el largo del tour. El ahorro mide la conveniencia de combinar dos clientes i y j a un tour y se puede cuantificar con

$$\mu_{ij}=d_{i0} + d_{0j} - d_{ij}$$

Altos ahorros indican que visitar al cliente j después del i es una buena elección.
Entonces podemos tomar :
$pij \sim \mu_{ij}^{\gamma}$

Con γ regulando la influencia relativa de los ahorros.

Para problemas con capacidad restringida y vehículos de igual capacidad Q, parece razonable asegurar un alto grado de uso de capacidad de los vehículos. Entonces altos valores $\kappa_{ij} = (y_i+qj)/Q$ indican alto uso de la capacidad a causa de la visita al cliente j después de visitar al i (y_i es la capacidad total usada hasta visitar al cliente i, q_j es la demanda del cliente j). Esto se puede usar para el Ant System dándole a esos clientes una alta probabilidad de ser seleccionados.
$pij \sim \kappa_{ij}^{\lambda}$ con λ determinando la influencia relativa de κ_{ij}.
La distribución de probabilidad para seleccionar al cliente j después de i es:

$$pij = \frac{\tau_{ij}^{\alpha} . \eta_{ij}^{\beta} . \mu_{ij}^{\gamma} . \kappa_{ij}^{\lambda}}{\sum_{h \notin \text{visitadas}_k} \tau_{ih}^{\alpha}.\eta_{ih}^{\beta}.\mu_{ih}^{\gamma}.\kappa_{ih}^{\lambda}} \quad \text{Si } j \notin \text{visitadas}_k$$

$$0 \quad \text{si no}$$

Ahorro mejorado (ref: [13])

Aunque la inclusión de ahorro y uso de la capacidad lleva a mejores resultados, esto es relativamente costoso en términos de tiempo de ejecución (ya que tienen que ser calculadas cada paso de la iteración) , por lo que se introducen los parámetros f y g y se usa la siguiente función paramétrica de ahorro para la visibilidad:

$$\eta_{ij} = d_{i0} + d_{0j} – g.d_{ij} + f.| d_{i0} – d_{0j}|$$

Lista de candidatos (candidate set)

El método es calcular para cada cliente un candidate set, y al elegir el próximo cliente a visitar buscar sólo dentro de dicho set (ref: [10], [11], [13]). Cuando ya se han visitado todos los clientes del set, recién ahí se pueden visitar los que no pertenecen al mismo.
Esto reduce el tiempo de construcción de la ruta a O(n) y la iteración a $O(n^2)$, eso sí, con una constante alta.

Elección del candidate set:

- Seleccionar un nro. fijo de vecinos más cercanos
- Comenzar con todos los clientes en el candidate set y luego ir restringiendo el conjunto de clientes, dependiendo de la fuerza del rastro de los arcos (los arcos menos visitados tendrán rastro más débil). Se puede usar *MAX-MIN* para construir el candidate set.

3. Estudio de las soluciones disponibles a problemas similares

El problema más parecido al nuestro del cual se ha presentado una solución es el MACS-VRPTW (Multiple Ant Colony System para VRP con una Time Window), propuesto por Gambardella, Taillard y Agazzi en el año 1999 (ref: [18]).
En este problema tanto los clientes como el depósito tienen una sola ventana de tiempo, y todos los vehículos tienen la misma capacidad. El objetivo en este caso es, además de minimizar el costo de la ruta, calcular la cantidad mínima de vehículos necesarios para realizarla.

El método presentado utiliza dos colonias de hormigas que trabajan simultáneamente, una de las cuales se dedica a minimizar el número de vehículos usados (ACS-VEI), y la otra a minimizar el tiempo (ACS-TIME). ACS-VEI se dedica a ir encontrando una solución que utilice un vehículo menos que la solución encontrada hasta el momento, y ACS-TIME optimiza el tiempo total de las soluciones que utilizan tantos vehículos como la mejor solución encontrada.
En ACS-TIME se usa el procedimiento new_active_ant que encuentra la solución de la hormiga k, es similar al ACS común pero acá η_{ij} (parámetro usado en la fórmula de la probabilidad de elegir cada cliente), se calcula según:

$$delivery\_time = \max(current\_time_k + t_{ij}, b_{ij})$$
$$delta\_time_{ij} = delivery\_time_j - current\_time_k$$
$$distance_{ij} = delivery\_time_j - current\_time_k$$
$$distance_{ij} = \max(1.0, distance_{ij} - IN_j)$$
$$\eta_{ij} = 1.0 / distance_{ij}$$

donde IN_j tiene la cantidad de veces que el cliente j fue insertado en una solución (ya que al correr la ACS-VEI las soluciones intermedias encontradas en gral no incluyen a todos los clientes). Se usa IN_j para favorecer a los clientes que son incluídos menos frecuentemente en las soluciones.
La idea al calcular η_{ij} así, es calcular la distancia de forma que aumente si aumenta el tiempo de entrega (tiempo de traslado + tiempo de espera), o según cuanto falte para cerrar la ventana. De esta forma si falta poco para cerrar la ventana, la distancia disminuye, lo que hace que η_{ij} aumente, y hace que lo probabilidad de atenderlo antes sea mayor.
Con respecto a la actualización de rastros, se tienen en cuenta un actualizamiento global y uno local. El global, al igual que en algunas de las alternativas que utilizamos, incrementa solo las aristas del mejor camino al final del ciclo. El local en cambio, decrementa cada arista en seguida de que una hormiga pasa por ella, esto es para cambiar dinámicamente la atracción de las aristas. Lo hace según la fórmula:

$$\tau_{ij} = (1 - \rho).\tau_{ij} + \rho.\tau_0$$

donde τ_0 es es valor inicial de los rastros.

Comparación con nuestro problema

En nuestro problema, nos es dado un conjunto de clientes a visitar cada uno con una demanda, un depósito y número finito de vehículos con capacidades limitadas y diferentes. Tanto el depósito como los clientes, tienen una o más ventanas de tiempo,

dentro de las cuales deben ser atendidos. Al ser finito el número de vehículos, estos se pueden repetir, es decir que un vehículo puede hacer un ciclo, volver al depósito a cargar y salir a visitar nuevos clientes. Nuestro objetivo consiste en minimizar el costo (una ponderación entre distancia y tiempo) de la ruta total.
No es aplicable a nuestro problema la optimización del número de vehículos que hace la colonia ACS-VEI, ya que para nosotros los vehículos tienen capacidades distintas y son finitos. Un aspecto que sí sería aplicable a nuestro problema, es la actualización local de rastros.

APÉNDICE B: SEUDOCÓDIGOS

Seudocódigo del cuerpo del algoritmo

```
// Carga de datos
cargar_clientes(lista_clientes)
cargar_grafo(ciudad)
cargar_vehiculos(lista_vehiculos)

//Inicialización de valores
mejor_camino = camino_con_costo_maximo

Repetir
    mejor_camino_iter = camino_con_costo_maximo
    Para k:=1 hasta m                    // para cada hormiga
        //Inicializar variables auxiliares
        no_visitados = lista_clientes – [deposito]
        cli_factibles = lista_clientes
        vehiculos_k = lista_vehiculos
        ruta_k = ruta_vacia
        rec = recorrido_vacio

        //Comenzar la ruta visitando el deposito
        elegir_vehículo(vehiculos_k, v, cli_factibles)      // devuelve el vehículo para
                                                            // realizar el ciclo y  actualiza
                                                            // la lista de clientes factibles.
        visitar_cliente(deposito, rec, no_visitados)
        elegir_prox_cliente(v, depósito, cli_factibles, i)
        visitar_cliente (i,rec, no_visitados)
        actualizar_rastro_local(deposito, i)
        actualizar_carga(v)          // y_v = y_v – q_i
        actualizar_horario(v)        // horario(v) = b_i
        crear_factibles(i, v, cli_factibles)
        Mientras no_visitados_k ≠ vacía y cli_factibles ≠ [deposito]
                elegir_ prox_cliente(v, i, cli_factibles, j)
                Si j = depósito
                        visitar_cliente (deposito, rec, no_visitados)
                        actualizar_rastro_local(i, deposito)
                        actualizar_horario(v)
                        asignar_carga(rec)          // carga:= Q_v – y_v
                        asignar_hor_llegada(rec)    // hora_llegada:= horarios(v)
                        inicializar_carga(v)        // y_v:= Q_v
                        cli_factibles = no_visitados + [deposito]
                        elegir_ vehículo(vehiculos_k, v, cli_factibles)
                        Si cli_factibles ≠ [deposito]   y no_visitados ≠ vacia
                                guardar_recorrido_en_ruta(rec, ruta_k)
                                manejo_sol_no_factible( )
                                break;
```

```
                    sino
                        Si no_visitados = vacia
                            //ya se visitaron todos los clientes
                            break;
                        sino
                            // Seguir visitando clientes
                            guardar_recorrido_en_ruta(rec, ruta_k)
                            rec = recorrido_vacio
                            elegir_prox_cliente(v, depósito, cli_factibles, i)
                            visitar_cliente (i, rec, no_visitados)
                            actualizar_rastro_local(deposito, i)
                            actualizar_horario(v)
                        finSi
                    finSi
                    elegir_prox_cliente(v, i, cli_factibles, j)
            finSi
            visitar_cliente (j, rec, no_visitados)
            actualizar_rastro_local(i, j)
            actualizar_carga(v)     // y_v = y_v - q_j
            //Inicializar valores para próximo cliente
            i = j
            crear_factibles(i, v, cli_factibles)
    finMientras
    Si no_visitados ≠ vacía
            manejo_sol_no_factible( )
    sino
            //Visita el depósito para cerrar la ruta
            visitar_cliente(deposito, rec, no_visitados)
            actualizar_rastro_local(i, deposito)
            asignar_carga(rec)              // carga:= Q_v – y_v
            asignar_hor_llegada(rec)        // hora_llegada:= horarios(v)
            guardar_recorrido_en_ruta(rec, ruta_k)
    finSi

    // Si la ruta hallada tiene mejor costo, o tiene igual costo pero menor cantidad
    // de vehiculos, que la mejor solucion de la iteración hasta el momento,
    // la sustituye.
    Si  costo(ruta_k) < costo(mejor_camino_iter)
            mejor_camino_iter = ruta_k;
    sino
            Si  costo(ruta_k) = costo(mejor_camino_iter) y
                cant_vehiculos(ruta_k) < cant_vehiculos(mejor_camino_iter)
                    mejor_camino_iter = ruta_k;
            finSi
    finSi
finPara

//Optimiza la mejor solución encontrada en la iteración
optimizar_ruta( mejor_camino_iter)
```

```
    // Si la ruta hallada tiene mejor costo, o tiene igual costo pero menor cantidad
    // de vehiculos, que la mejor solucion de la iteración hasta el momento,
    // la sustituye.
    Si  costo(mejor_camino_iter) < costo(mejor_camino)
        mejor_camino = mejor_camino_iter;
    sino
        Si  costo(mejor_camino_iter) = costo(mejor_camino) y
            cant_vehiculos(mejor_camino_iter) < cant_vehiculos(mejor_camino)
                mejor_camino = mejor_camino_iter;
        finSi
    finSi

    // Actualiza intensidad de rastros para próxima iteración.
    actualizar_rastros( )
hasta que condición de fin = verdadera.
```

Seudocódigo de las funciones que utiliza el algoritmo

```
completar_grafo (in C:MatrizCostos; out D:MatrizCostos)
// Algoritmo para completar el grafo dado. Debido a que el grafo que representa los
// clientes y los caminos puede no ser completo, lo completamos para poder aplicar el
// formalismo, devuelve el grafo ya completo. Obs: El algoritmo además de agregar
// aristas al grafo, cambia el valor de aristas ya existentes si encuentra un camino mas
// corto.

        For i:= 1 to n do
                For j:= 1 to n do
                        D(i,j):= C(i,j);
                fin
        fin
        For k:=1 to n do
                For i:= 1 to n do
                        For j:= 1 to n do
                                if D(i,k) + D(k,j) < D(i,j) then
                                        D(i,j) := D(i,k) + D(k,j);
                                fin
                        fin
                fin
        fin
```

actualizar_rastro( )

```
// Se escanea cada recorrido de mejor camino y para cada par de clientes visitados se
// actualiza el grafo.
        Para cada arista (i,j) ∈ mejor_camino hacer
                calcular τij:=ρ τij+ (1-ρ) Δτij^best        // Δτij^best = 1/L^best ∀ i,j
        fin
```

guardar_recorrido_en_ruta*(in rec:recorrido, inout: ruta_k)*

```
Si existe un recorrido r1 en ruta_k con el mismo vehiculo que rec
        // Agregar rec al final de r1
        append(r1,rec)
sino
        // Agregar rec a ruta_k como un recorrido nuevo
        insertar(rec, ruta_k)
finsi
```

elegir_prox_cliente*(in v:vehiculo, in i: cod_cliente, in cli_factibles: lista,*
out cliente_elegido: cliente)

// La función *elegir_prox_cliente* elige el próximo cliente a moverse. Depende de la
// carga disponible, las ventanas de tiempo de los clientes, las ventanas de tiempo del
// depósito, y la probabilidad.

```
probabilidad(v,i, cli_factibles, cliente_elegido)
Si cliente_elegido ≠ depósito
        Sacar cliente_elegido de la lista cli_factibles
fin
```

Crear_factibles*(in i:cod_cliente, in v: vehículo, inout cli_factibles: lista,*
out b: lista_tiempos)

```
Para cada c ∈ cli_factibles
        Si y_v - q_c ≥ 0          // si alcanza la carga para abastecer a ese cliente
        ó c = depósito
                buscar_ventana(i, c, v, b_c, nro_ven);
                Si c ≠ depósito
                y (nro_ven = 0 ó b_c + s_c + t_c,0 > l_0,w0) // antes de elegir el cliente
                                                             // me fijo que pueda volver
                                                             // desde allí al depósito
                        sacar c de cli_factibles
                fin
        sino
                sacar c de cli_factibles
        fin
fin
```

buscar_ventana*(in i: cod_cliente, in c: cod_cliente, in v:vehículo, out b_c: tiempo,*
out nro_ven: integer)

```
h:=1
nro_ven = 0
Mientras nro_ven = 0 y h ≤ w_i hacer
    Si  horarios(v) + s_i + t_i,c < l_c,h
        Si e_c,h ≤ horarios(v) + s_i + t_i,c
            b_c = horarios(v) + s_i + t_i,c
        sino
            b_c = e_c,h
        finSi
        nro_ven = h
    sino
        h:=h+1
    finSi
fin
```

visitar_cliente*(in i cliente, inout rec:recorrido, inout no_visitados: lista_clientes)*

```
// Inserta el cliente i en la lista de visitados
insertar(i, rec)
Si i ≠ depósito
    eliminar(i, no_visitados)
finSi
```

actualizar_rastro_local*(in i,j : clientes)*

```
//Actualiza el rastro del tramo entre los clientes i y j
    τij := (1 – ξ) .τij + ξ.τ0
```

elegir_vehículo*(inout veh_factibles: lista_vehículos,*
out v: vehículo, inout cli_factibles: lista)

```
encontre=false
Mientras no encontre y veh_factibles no vacía
    sorteo_vehículo(veh_factibles, γ, δ, vs)
    crear_factibles(i, vs, cli_factibles)
    Si cli_factibles ≠ [deposito]
        encontre = true
    sino    // como está en el depósito, quiere decir que no se puede mover
        // a ningún cliente, hay que desechar el vehículo
        elimino vs de la lista veh_factibles
    fin
finMientras
Si encontre
    v:= vs
fin
```

sorteo_vehículo*(in veh_factibles: lista_vehículos, in γ,δ: parámetros, out vs: vehículo)*

Para cada posición j de la lista veh_factibles
v:= veh_factibles (j)
h=$l_{0,w0}$ – horarios(v)
q=y_v
$p(v) = \frac{h^{\gamma} * q^{\delta}}{\sum_{h \in veh\_factibles} h^{\gamma} * q^{\delta}}$
finPara
u:= random ( ) // Variable aleatoria para el sorteo
cont:=1
acum:= p(veh_factibles(1))
Mientras (acum < u) hacer
cont:= cont + 1
acum:= acum + p(veh_factibles(cont))
fin
vs:= veh_factibles(cont)

manejo_sol_no_factible*( )*

// dist_max = $\Sigma_i\Sigma_j d_{ij} + 1$, tiempo_max = $n_v * (l_{0,w0} – e_{0,1}) + 1$ (todos los vehículos
// durante todo el horario del depósito)
dist_ruta$_k$ = dist_max * cant de clientes no visitados
tiempo_ruta$_k$ = tiempo_max * cant de clientes no visitados

Random_Proportional*(in v:vehiculo,in i: cod_cliente, in factibles: lista,*
out cliente_elegido: cliente)

Para cada posición j de la lista factibles
c:= factibles (j)
buscar_ventana (i, c,v, b_c, nro_ven)
delta_tiempo = b_c – horarios(v)
costo := φ. d_{ij} + ψ.delta_tiempo
η(i,c) := 1.0 / costo

$p(i,c) := \frac{\tau(i,c)^{\alpha}\, \eta(i,c)^{\beta}\, \varepsilon_c}{\sum_{h \in factibles} \tau(i,h)^{\alpha}.\eta(i,h)^{\beta}\, \varepsilon_h}$

finPara
u:= random ( ) // Variable aleatoria para el sorteo
cont:=1
acum:= p(i, factibles(1))
Mientras (acum < u) hacer
cont:= cont + 1
acum:= acum + p(i, factibles(cont))
fin
cliente_elegido:= factibles(cont)

Seudo_Random_Proportional*(in v:vehiculo, in r: cliente, in factibles: lista_clientes)*

q:= random ( )
Si $q \leq q_0$
c:= $\max_{u \in \text{factibles}} (\tau(r,u)^{\alpha} . \eta(r,u)^{\beta})$
sino
c:= *Random_proportional (r, factibles)*
finSi

Probabilidad*(in v:vehiculo, in i:cod_cliente, in factibles: lista, out cliente_elegido:cod_cliente)*

Seudo_Random_Proportional(v,i, factibles,cliente_elegido)

Costo_Ruta*(in φ,ψ ,out costo:int)*

costo=φ $dist\_ruta_k$ + ψ $tiempo\_ruta_k$

Or-Opt *(inout rec: recorrido, inout costo: integer)*

```
Para cada ciclo del recorrido
  camino = ciclo
  p = primer cliente visitado del siguiente ciclo // lo usamos para no correr el
                  // tiempo de llegada al depósito si eso implica correr el tiempo de
                  // visitado de este cliente
  for s = 3 downto 1
      t = 1
      Mientras t <= n+1
          sacar_cadena(s, t, camino, camino_temp, sub_cadena)
                          // devuelve sub_cadena que tiene s vertices
                          // consecutivos empezando desde t.
          Para todo vertice u empezando desde t
              simular_insertar(sub_cadena, u, camino_temp, costo,
                                                          mejor)
              // devuelve mejor en true si el costo de insertar sub_cadena
              // entre u y u+1 de camino_temp es menor que costo y los
              // tiempos y carga cumplen las restricciones. Si u+1 es el
              // depósito tener en cuenta p.
              Si mejor
                  insertar(sub_cadena, u, camino_temp, camino)
                                  // devuelve en camino camino_temp
                                  // con sub_cadena insertada en u
                  break           // sale del para todo de u
              fin
          finPara
          Si mejor
              t := 1
          sino
              t := t +1
          fin
```

```
            finMientras
        finFor
    finPara

ajustar_tiempos(inout rec: recorrido)
    hs_libre:=max_tiempo
    i:=first(visitados_k(rec))
    Mientras not end(visitados_k(rec)) and hs_libre > 0
        j:= next(visitadosk(rec))
        hs_libre:=min(hs_libre, l_{i,nro_ven(i)} – b_i)
        espera:=bj – (bi + si + tij)
        Si espera < hs_libre
            hs_libre:= hs_libre – espera
            ajustar:= espera
            espera:= 0
        sino
            espera:= espera – hs_libre
            ajustar:=hs_libre
            hs_libre:=0
        finSi
        actualizo_horas(i, ajustar)   // b_i = b_i + ajustar
                                      // b_{i-1} = b_{i-1} + ajustar
                                      // .....
                                      // b_deposito = b_deposito + ajustar
        i:=j
    finMientras

optimizar_ruta(in mejor_camino: ruta)
    Para cada recorrido rec de mejor_camino
        or_opt(rec, costos)    // optimiza usando OR-opt
        ajustar_tiempos(rec)
    finPara
```

APÉNDICE C: INSTRUCTIVOS DE EJECUCIÓN DE LOS PROGRAMAS

1. Instructivo para Transformación al Formato Ants

Los programas de transformación de formato son:

1) trtsplib – para transformación del formato TSPLIB (ref [32]) al formato Ants.
2) trvrplib – para transformación del formato VRPLIB (ref [31]) al formato Ants.
3) trsolom – para transformación del formato Solomon (ref [30]) al formato Ants.

Brindaremos un instructivo para la ejecución del programa *trsolom*, los demás se ejecutan en forma muy similar.

Se ejecuta en línea de comandos:
`> trsolom <nombre_archivo>`
donde <*nombre_archivo*> indica el nombre del archivo que contiene el problema en formato Solomon. Dicho nombre debe tener como máximo 8 caracteres para el nombre y 3 para la extensión.

Luego el programa solicita la siguiente información:
`Ingresar proporcion entre distancia y tiempo:`
Aquí se ingresa la proporción entre distancia y tiempo. Esto permitirá asignar tiempos de traslado entre cada par de clientes a partir de la distancia, debido a que el formato Solomon no provee esta información pero nuestro formato la requiere.

`Ingresar factor distancia :`
Aquí se ingresa el factor de conversión distancia/unidad de costo. Permite asignar mayor o menor peso en el cálculo del costo a la distancia. Puede ser 0 si para el cálculo del costo sólo interesa el tiempo.

`Ingresar factor tiempo :`
Aquí se ingresa el factor de conversión tiempo/unidad de costo. Permite asignar mayor o menor peso en el cálculo del costo al tiempo. Puede ser 0 si para el cálculo del costo sólo nos interesa la distancia.

En el archivo “datos.dat” se devuelve el problema en formato Ants.

Ejemplo de ejecución

```
t5ants@lulu ~/solomon > trsolom R112.TXT
Ingresar proporcion entre distancia y tiempo: 1
Ingresar factor distancia : 0
Ingresar factor tiempo : 1
```

En el archivo “datos.dat” se devuelve el archivo en formato Ants.

2. Instructivo para la Ejecución del Algoritmo

El programa *vrp_ants* se utiliza para ejecutar el algoritmo Ants.
A continuación se brinda un instructivo para la ejecución de este programa:

Se ejecuta en línea de comandos:
```
> vrp_ants
```

Luego el programa solicita la siguiente información:
```
Ingrese nro de corridas :
```
Aquí se ingresa el número de veces que se desea ejecutar el algoritmo. Para cada corrida se obtendrán distintos resultados debido a que se utilizan distintos torrentes de números aleatorios, por lo tanto puede ser útil correrlo varias veces para obtener un mejor resultado.

```
Desea correr post-optimización? [S/N] :
```
Al ingresar "S", se ejecutará una post-optimización dentro del algoritmo, que permitirá optimizar la solución encontrada por la mejor hormiga de cada iteración, para cada corrida.

A continuación el programa verifica si existe el archivo de parámetros "param.dat".
Si dicho archivo no existe, se crea uno nuevo, solicitando los valores para cada parámetro por pantalla:
```
nro de hormigas :
```
Y así sucesivamente para los demás parámetros.

Si el archivo de parámetros ya existe, se realiza la siguiente pregunta:
```
Desea cambiar algún parametro? [S/N] :
```
Si la respuesta es "S", se piden por pantalla uno a uno los nuevos valores de los parámetros:
```
nro de hormigas [100] :
```
Aquí se puede ingresar un nuevo valor para el número de iteraciones, o contestar "S" para mantener el valor actual (que es 100 en este caso).
Análogamente se ingresan los nuevos valores para los restantes parámetros. Cuando se hayan ingresado todos los parámetros, se sobreescribirá el archivo "param.dat" con los nuevos valores.
Si en cambio la respuesta es "N", se utilizan los parámetros del archivo existente.

```
Ingrese nombre del archivo en formato ants :
```
Aquí se pide el nombre del archivo que contiene el problema a resolver, en formato Ants. El archivo debe existir en el directorio actual y su nombre debe contar como máximo de 8 caracteres para el nombre y 3 caracteres para la extensión.

A continuación el programa comienza a correr, desplegando en pantalla a medida que va trabajando el número de ejecución y luego el número de hormiga que en ese momento está construyendo una ruta.

Al finalizar la ejecución se dejan dos archivos con los resultados obtenidos: "result.dat" y "mejores.dat". En el primer archivo se muestra el costo y la cantidad de vehículos de cada corrida, más el costo promedio y mejor costo entre todas las corridas. En el

segundo se devuelve información más detallada sobre las rutas obtenidas en cada corrida.

Ejemplo de ejecución

1) Corrida

```
t5ants@lulu ~/pruebas > vrp_ants
Ingrese nro de corridas de prueba : 1
Desea correr post-optimización? [S/N] :N
Desea cambiar algun parametro? [S/N] :S
nro de hormigas [100] :20
nro de iteraciones [200] :S
rho [0.7] :S
alpha [1.3] :S
beta [5] :S
gamma [1] :S
delta [1] :S
omega [20] :0
q0 [0.5] :S
xi [0.7] :S
tau [8.54e-06] :3.12e-05
Ingrese nombre del archivo en formato ants :datos-20.dat
```

2) Resultados

result.dat

```
problema   :  datos-20.dat
 costo    veh
 887     3
 --------------
 887   3

Mejor costo encontrado para el problema 887
//////////////////////////////////////
```

mejores.dat

```
Corrida nro. 1
hormiga      :   12
distancia    :   312.000000
tiempo       :   575
vehiculos usados      :   3
clientes no visitados :   0
vehiculo     hora salida   hora llegada       recorrido
0              0               205         cliente   hora atencion    ventana
                                           0              0               0
                                           12              15              0
                                           4               73              0
                                           3               108             0
                                           9               133             0
                                           20              154             0
                                           1               180             0
                                           0               205             0
                                          carga total       86
                                          Distancia 113
4              0               187          cliente   hora atencion    ventana
                                           0              0               0
                                           2               18             0
                                           15              41             0
                                           14              67             0
```

```
                                    16              88              0
                                    17              109              0
                                    5               129             0
                                    6               149             0
                                    13              166              0
                                    0               187             0
                                   carga total       108
                                   Distancia 107
1               0               183          cliente   hora atencion    ventana
                                    0              0               0
                                    18              47              0
                                    8              67              0
                                    7              89              0
                                    19              110              0
                                    11              127              0
                                    10              148              0
                                    0              183              0
                                   carga total       71
                                   Distancia 92
Tiempo de CPU: 124.589996
```

APÉNDICE D: Estudio de la paralelización del código

Introducción a la programación paralela

El procesamiento en paralelo es básicamente el método por el cual pequeñas tareas o módulos resuelven un problema mayor.
En los pasados años creció la aceptación del procesamiento en paralelo tanto para problemas científicos como para aplicaciones de propósito general, debido a la demanda de mejor performance, menores costos y mayor productividad. El desarrollo de MPPs (massively parallel processors) y la computación distribuida dieron lugar a la aceptación del procesamiento en paralelo. La computación distribuida se vio favorecida por el avance en tecnologías de red (Ethernet, FDDI ,HiPPI, SONET, ATM), según muestra la figura.

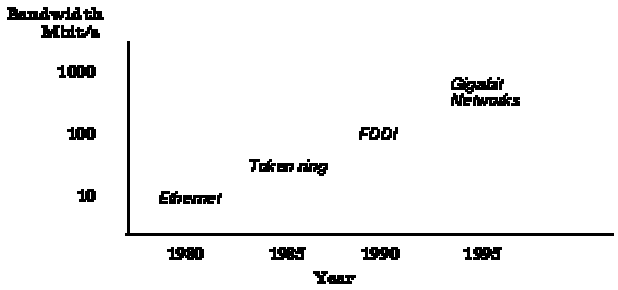


Figura: velocidades de red

El objetivo de paralelizar un problema es lograr tiempos de ejecución más rápidos que en una máquina serial; o resolver problemas de gran tamaño. Una aplicación que corra en paralelo puede hacerlo en dos tipos de arquitecturas diferentes: un multiprocesador o un sistema distribuido. Dentro de éste último, un cluster constituye un sistema en el que diferentes topologías soportan decenas de miles de procesadores.
Los clusters se caracterizan por dar resultados más rápidos que una LAN (Local Area Network) común, y sus protocolos de comunicación tienen latencias más bajas. Un cluster, al contrario de un SMP (Shell Memory Processor) es un sistema de memoria distribuida. Por ejemplo, los SP son varios RS 6000 conectados a través de un switch que trabajan cooperativamente como un único recurso de cómputo.

En un sistema distribuido los factores ancho de banda de la red y latencia del canal (establecimiento de la comunicación), influyen directamente en el tiempo de ejecución debido a que no se trata de compartir un área común de datos como en un multiprocesador sino de transmitir mensajes para compartir los datos globales.
Adicionalmente, se deben tener en cuenta temas de seguridad si la aplicación corre en un sistema distribuido. Básicamente, si una aplicación corre en paralelo es necesario controlar el overhead de sincronización y comunicación para elevar el grado de performance.
Para evaluar la performance final se puede considerar el “speed up algorítmico” que se define como el cociente entre el tiempo de ejecución usando un solo procesador y el tiempo usando n procesadores. Igualmente el “speed up” se puede definir como el cociente entre el tiempo de ejecución del algoritmo más rápido que resuelve el problema

sobre el tiempo de ejecución para n procesadores. Para mejorar el “speed up” una opción es aumentar el número de procesadores. A la habilidad de agregar más procesadores para obtener un mejor “speed up” se le llama escalabilidad.

Un factor relevante sobre el desempeño de aplicaciones distribuidas ejecutando en una red, es el balance de cargas. La idea es evitar que la performance global del sistema se degrade a causa de la demora en tareas individuales. Las técnicas de balance de carga son estáticas, dinámicas o adaptativas, con el principal criterio de mantener los procesadores la mayor parte del tiempo ocupados y disminuir la comunicación entre procesos.
En las técnicas de despacho estáticas, la asignación inicial de tareas se mantiene, sin tener en cuenta las fluctuaciones de carga de la red, por el contrario el despacho dinámico asigna las tareas al momento de su creación, aprovechando de los tiempos ociosos de los procesadores que conforman la red. El modelo adaptativo utiliza técnicas de migración de procesos de forma aprovechar completamente las fluctuaciones de carga de la red.

Aplicación de los conceptos a nuestro sistema

Para llevar nuestra aplicación a un sistema paralelo, nuestra elección sería ejecutar el mismo código de ruteo en cada procesador distribuyendo la carga de procesamiento y logrando así simular un sistema de hormigas en el que éstas buscan el camino óptimo en forma cooperativa.
Es decir que en nuestro caso de estudio habría que contar con múltiples procesadores ejecutando las mismas instrucciones y utilizando un área (lógica) común de datos.
Estos datos comunes son los datos de los clientes que no varían (distancia, demanda, ventanas de tiempo y tiempo de servicio), y los mismos se manipulan según la arquitectura elegida tal como se mencionó más arriba.
Si la opción fuera la de un sistema multiprocesador, lo que se tendría es varios procesadores asincrónicos cuya comunicación sería a través de la memoria global compartida. En estos casos, el bus de datos limita la escalabilidad a decenas de procesadores (además de la limitaciones físicas). También existe la complejidad adicional de mantener la coherencia entre los cachés de cada procesador.

La segunda opción sería un cluster en el que varías máquinas se comunican a través de mensajes explícitos, extrayendo los datos de una base de datos, por ejemplo. También sería factible que aquellos se replicaran en cada nodo para disminuir congestionamiento, aunque en este caso habría que considerar la posibilidad de eventuales cambios en los datos, además de los recursos de almacenamiento.

Adicionalmente al área de datos mencionada antes, tendríamos un área local de datos a cada procesador donde se almacene información sobre los clientes visitados hasta el momento, la ruta que se está construyendo junto con la información de los vehículos elegidos.

Luego tendríamos un área de datos común y dinámica, que cumple la función autocatalítica del método. La información que comparten las hormigas es la actualización local y global de rastros que se utiliza para construir las probabilidades de elección de los clientes, junto con la mejor solución hallada hasta el momento.

La actualización global de rastros consiste en que si una hormiga obtiene una solución mejor que la mejor hallada hasta el momento, actualiza los rastros de las aristas del camino encontrado para hacer más probable su elección por otras hormigas, de manera de aumentar la probabilidad de elección de ese camino.
Mediante la actualización local se logra disminuir el atractivo de caminos ya recorridos para incrementar la exploración y evitar que se encuentren siempre a partir de un punto las mismas soluciones.

Con respecto a la actualización de rastros, serían aplicables los métodos Ant Density y Ant Quantity, pero empobrecerían la performance ya que requieren de intercambio de información en cada paso. El método Ant Cycle es igualmente aplicable y tiene la ventaja de economizar más el flujo de información, además de ser el que escogimos para la implementación de la versión secuencial.

En cualquier arquitectura de sistema paralelo se tiene conceptualmente un grafo dirigido acíclico cuyos nodos representan las tareas a ejecutarse y sus aristas las dependencias entre las mismas. Estas últimas imponen limitaciones al paralelismo, es decir, no es posible comenzar una tarea si la o las tareas previas no han culminado.

En la paralelización de nuestro algoritmo, la idea original sería que cada procesador ejecutara una réplica idéntica del mismo código dividiendo las hormigas según la cantidad de procesadores. De esta manera lo que se logra es distribuir la carga de procesamiento, teniendo en cuenta la latencia debida al intercambio de información de rastros actualizados.
En la versión serial de nuestro algoritmo, el camino de la mejor solución encontrada hasta el momento actualiza sus rastros en cada iteración. En el modelo paralelo, mantener este criterio significa que, al finalizar la iteración, el flujo de ejecución de todos los procesadores se sincronice en un único nodo para comparar la mejor de las rutas hallada y actualizar el rastro de ese camino únicamente. Esto implica que luego de finalizada una iteración (k hormigas realizan su recorrido) se debe enviar a una tarea “maestro” el costo del camino hallado y las aristas involucradas, de manera de que éste decida si el costo es mejor que el mejor hallado hasta el momento, y envíe a las demás tareas los rastros actualizados correspondientes al mejor camino.
Debido al intenso tráfico en la red que generaría esta solución (degradando la performance) se podría en forma alternativa comunicar las tareas cada x cantidad de iteraciones. Esta opción no generaría los mismos resultados que la original, pero mantiene el efecto cooperativo del método y logra un mejor aprovechamiento de los recursos de la red.

La paralelización lleva a que ya no exista un flujo de ejecución del algoritmo, sino diferentes estados posibles que dependen de múltiples factores. Por ejemplo, la velocidad de un procesador puede hacer que la actualización local de rastros sea más rápida y eso influir en la decisión de las demás hormigas para la elección de sus caminos.
También el tiempo de ejecución va a depender de la demanda de los procesadores en ese instante y si son dedicados o no.

Según la ley de Amdhal, la parte secuencial de un programa determina una cota inferior al tiempo de ejecución. En nuestro caso, nuestra propuesta es simular la construcción de rutas por diferentes hormigas en paralelo, así que esto determinaría el tiempo inferior de ejecución. Sin embargo, también existe la alternativa de hacer una granularización más fina del programa para sacar provecho de las secciones paralelizables dentro del cuerpo mismo del algoritmo.
Otra propiedad de la ley antes mencionada permite deducir que al aumentar el tamaño del problema implica que aumentar la cantidad de procesadores ayuda a mejorar la performance, ya que la parte paralelizable del problema crece con su tamaño (si aumenta la cantidad de clientes aumenta en igual proporción la cantidad de hormigas).

En nuestro caso se requiere una máquina MIMD –Multiple Instruction Multiple Data- debido a que se ejecutan diferentes instrucciones sobre diferentes datos, basado en un modelo SPMD –Single Program Multi Data-, un mismo programa corriendo en diferentes nodos (en instrucciones diferentes) actualizando diferentes datos, y a su vez un área común de datos que es la mejor solución obtenida.

Elección de software

Existen varios paquetes de software para asistir al programador en un sistema distribuido, entre ellos: P4, Express, MPI, Linda, PVM.
PVM permite que un conjunto de computadores heterogéneos sean vistos como una máquina virtual, de ahí su nombre. Está hecho para explotar la heterogeneidad de un sistema distribuido (diferentes arquitecturas, formatos de datos, velocidad computacional, etc), maneja en forma transparente la transmisión de mensajes y la transformación de datos. Cuenta con una interfase de programación intuitiva en la que el usuario escribe la aplicación como un conjunto de tareas cooperativas. Estas tareas se sincronizan y comunican haciendo uso de las rutinas provistas por las librerías de PVM. En cualquier punto de la ejecución una tarea puede detener o lanzar a otra, y agregar o eliminar una computadora de la máquina virtual, esto permite mantener una estructura de dependencias entre las diferentes tareas. Debido a todas estas razones PVM ha ganado aceptación en el desarrollo de la programación paralela, y sería nuestra elección a la hora de hacer la versión paralela de nuestro sistema.

La programación paralela usando un sistema como PVM se puede realizar desde tres puntos de vista, el llamado “crowd computing”, “tree computing” e híbridos. En el modelo de “crowd computing” una colección de procesos relacionados (eventualmente que ejecutan el mismo código) realizan computaciones en diferentes porciones del área de datos, involucrando el intercambio periódico de resultados. Este modelo se puede dividir en dos categorías:

- *master-slave* (o host-node) : modelo en el que un programa de control (el maestro) es responsable de la asignación de procesos, inicialización, recolección y despliegue de resultados. Los esclavos ejecutan el programa propiamente dicho, y la carga de trabajo es asignada estática o dinámicamente por el maestro o por ellos mismos.

- *node-only* : modelo donde múltiples instancias de un mismo programa realizan por igual todas las tareas.

El modelo “tree” consiste en que los procesos se lanzan dinámicamente a medida que progresa la ejecución del programa, a manera de árbol. Es un modelo natural para casos en los que no se conoce a priori la carga de trabajo, por ejemplo algoritmos “divide and conquer” (ordenamiento de listas usando “merge” es un ejemplo).
En ambos modelos todos los procesos se comunican y sincronizan entre ellos.

La elección de uno u otro modelo es dependiente de la aplicación y en nuestro caso nos parece más aplicable el modelo “crowd” ya que la carga de trabajo se conocerá de antemano, y un sistema maestro-esclavo en el que el maestro se encargue de la inicialización de la aplicación y el despacho de tareas.

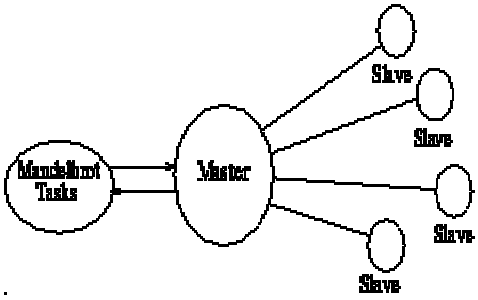


Figura: paradigma master-slave (maestro-esclavo)

Existen a su vez diferentes metodologías en cuanto a la distribución de carga de trabajo entre los diferentes procesos. *Data decomposition*, asume que el problema en su totalidad involucra aplicar transformaciones iguales a ciertas estructuras de datos que pueden ser divididas para operar sobre ellas. *Function decomposition*, divide el trabajo basado en diferentes operaciones y funciones.

Posiblemente en nuestro caso la paralelización sería un caso particular de descomposición funcional en el que cada tarea esclavo realiza los mismos cálculos, debido a que los datos que se actualizan no se dividen entre las diferentes tareas (no sería aplicable descomposición de datos).

En conclusión, para utilizar el sistema PVM sería necesario en primer lugar desarrollar la versión paralela de la aplicación, tomando decisiones de estructura y eficiencia. Con respecto a la primera, eligiríamos un modelo crowd maestro-esclavo.
Con respecto a la eficiencia cuando se trata de sistemas con memoria distribuida es importante reducir la frecuencia y el volumen de comunicaciones. En estos sistemas aumentar la granularidad (cantidad de trabajo que realiza cada nodo) significa mejor performance.
Los lenguajes de programación aceptables son C ,C++, y Fortran 77 para PVM, nuestra aplicación se implementó en C++.

El software de PVM se consigue a través de un servicio de distribución de software, netlib, a través de ftp.netlib.org, http://www.netlib.org/pvm3/index.html, o por email a netlib@netlib.org.

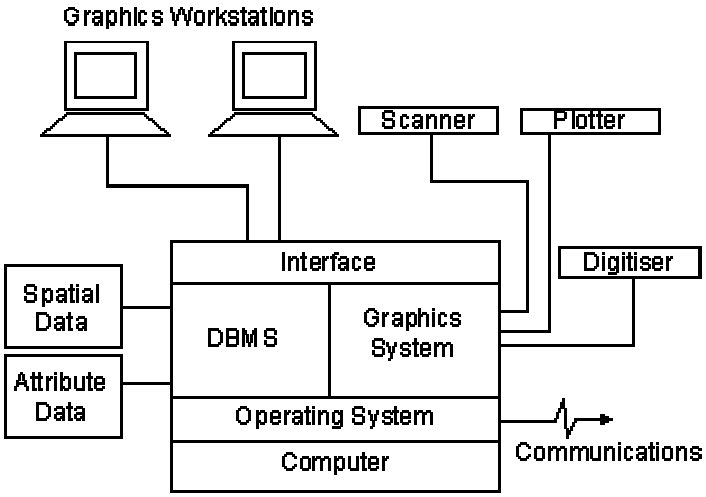
APÉNDICE E: Estudio de la inclusión de una posible interfase que utilice Sistemas de Información Geográfica

Un Sistema de Información Geográfica ("Geographic Information System" o GIS) es un sistema que se usa para capturar, almacenar, verificar, integrar, administrar, analizar y desplegar datos relacionados a posiciones en una superficie. Básicamente un GIS es usado para manipular mapas de cualquier tipo.

El mapa se representa como un conjunto de diferentes capas en las que cada una contiene información de una característica determinada. Cada característica se vincula a una posición en la imagen gráfica del mapa.

Estas capas de datos se organizan para ser estudiadas y realizar análisis estadísticos. Sus usos son entre otros: gubernamentales, planeamiento territorial, ambiental, ingeniería, administración de recursos, distribución, etc.

La estructura básica de un Sistema de Información Geográfica es como se muestra en la figura.



The physical components of a GIS

La ubicación de las variables dentro del mapa geográfico se determinan por las tres coordenadas dimensionales x, y, z; cualquier variable definida de esta manera puede ser introducida en un GIS. Existen bases de datos que pueden ser directamente introducidas en estos sistemas; diferentes tipos de datos en formato de "mapa" pueden alimentar al sistema.

La información digital que no se encuentre en el formato antes dicho es convertida por el sistema de manera de obtenerla en forma reconocible para él. Por ejemplo, imágenes satelitales pueden ser analizadas para llevarlas a un formato geográfico. También es posible usar un scanner para capturar los datos e ingresarlos al sistema.

Imágenes satelitales interpretadas por una computadora para producir un mapa, podrán ser leídos en el GIS en lo que se llama formato "Raster". Estos archivos de datos consisten en filas de celdas uniformes codificadas de acuerdo a los valores de los datos. Raster es un método de almacenamiento, procesamiento y visualización de datos espaciales. Básicamente, el área se divide en filas y columnas formando una grilla. Cada celda contiene ciertos atributos y coordenadas, que corresponden a la distribución de la matriz.

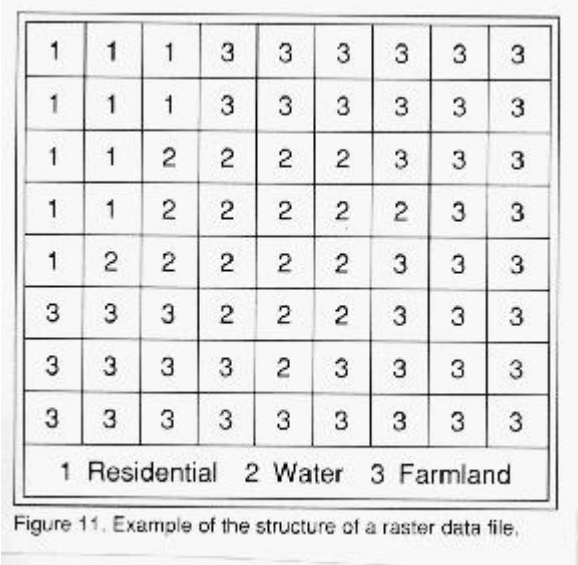


Figure 11. Example of the structure of a raster data file.

Los archivos de datos Raster son procesados fácilmente por una computadora, sin embargo, son menos detallados y no tan visualmente agradables como lo son los archivos vectoriales, que se aproximan a lo que es un mapa dibujado. Los datos digitales de un vector son capturados como puntos, líneas (una serie de coordenadas) y áreas (determinadas por líneas).

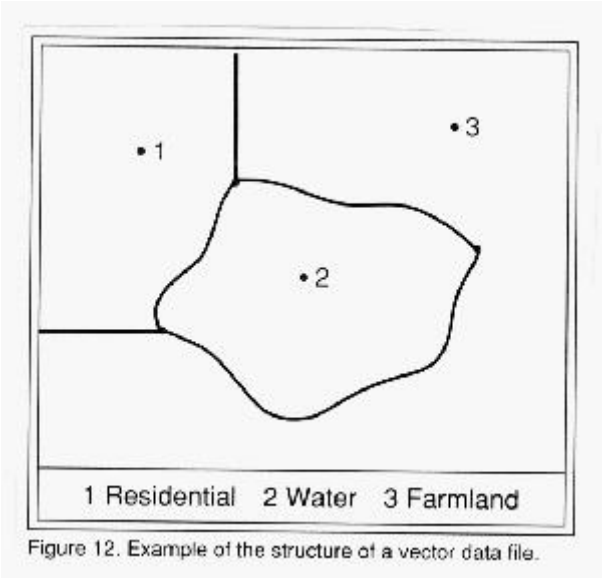


Figure 12. Example of the structure of a vector data file.

Por ejemplo, un GIS puede ser usado para convertir la imagen de un satélite en una estructura vectorial generando líneas alrededor de las celdas de igual característica, y determinando relaciones espaciales entre ellas como adyacencia e inclusión.
Un GIS hace posible integrar información que resulta difícil de asociar a través de otros métodos. Utiliza la combinación de diferentes variables (información con la que ya se cuenta) para construir y analizar nuevas variables.

Utilizando fotos aéreas scanneadas como una guía visual, el GIS proveerá de información acerca de la geología e hidrología del área.

Entre sus utilidades se encuentra el permitir el cálculo de tiempos de respuesta en situaciones de emergencia, por ejemplo en el caso de desastres naturales.
Otra utilidad que se le da a estos sistemas es capturar las áreas de cultivos de una región que se deben proteger de la polución.

Un GIS puede reconocer y analizar relaciones espaciales entre los fenómenos que se encuentran registrados en el mapa; condiciones de adyacencia, inclusión, y proximidad pueden ser determinados por el GIS.
Si todas las fábricas cercanas a un área arrojaran desperdicios químicos en un río al mismo tiempo, ¿cuánto tardaría la polución en comenzar a hacer sus efectos en aquellas tierras? El GIS realiza una simulación de la ruta de los materiales a lo largo de una red lineal. Para esto se asignan valores como dirección y velocidad a la corriente y así se "moverán" los contaminantes a lo largo de la corriente digital.
Para lograrlo se utilizarán diferentes mapas de tierras húmedas, corrientes, utilización de tierras, y elevaciones. El sistema produce un nuevo mapa a partir de los anteriores que ranquea las tierras húmedas de acuerdo a su sensibilidad debido a la cercanía a las fábricas e industrias.

Se pueden generar gráficos a partir de los resultados obtenidos de la simulación para una mejor visualización de los mismos.
En conclusión, un GIS provee al usuario de las utilidades de un mapa, es decir extraer información de diferente tipo (calles, vegetación, etc), adicionando a esto el hecho de que los datos están almacenados en una computadora, lo que hace posible el análisis y la modelación. Por ejemplo, seleccionando dos puntos geográficos extraer conclusiones acerca de ellos (la mejor ruta que los une).
Uno de los tantos usos que se le da a las redes de calles involucra el modelado del servicio de distribución dentro de ciudades. Este modelo se usará como una herramienta de decisión para ahorrar tiempo y dinero y proveer de un mejor servicio de atención a los clientes. Es de esta manera que nuestro sistema de ruteo de vehículos aplicable a clientes con múltiples ventanas de tiempo se puede adaptar a un GIS en el que éste último actúe como interface con el usuario.

Aplicación a nuestro sistema

Para hacer esto posible sería necesario contar una arquitectura en tres capas que consta básicamente de:
- el Sistema VRP-mTW diseñado en este taller
- el Sistema de Información Geográfica, y
- una base de datos donde almacenar la información de la que hacen uso ambos sistemas.

Muchos cálculos relacionados a las diferentes variables de nuestro sistema (distancia entre clientes por ejemplo) podrían estar centralizados en el Sistema de Información Geográfica.
Entonces el cambio a implementar en nuestro sistema sería brindar la posibilidad de conexión a una base de datos y derivar nuestras estructuras de datos (que cargamos en memoria) a información en un medio de almacenamiento secundario. De esta forma se contará con tablas en una base de datos donde estará almacenada la información que sirve como entrada al algoritmo de ruteo por nosotros implementado.

El GIS le presentará al usuario una interfase visual en la que él podrá construir interactivamente el mapa de clientes y sería posible agregar también la posibilidad de ingresar los demás parámetros del sistema en forma visual.
Utilizando el archivo de salida de nuestro algoritmo “mejores.dat” que contiene el detalle de los recorridos que componen la solución obtenida, éstos se podrían mostrar en el mapa de forma conveniente (por ejemplo: marcando con distintos colores los trayectos realizados por distintos vehículos).