Tesis de Licenciatura

correspondiente a la carrera de

Licenciatura en Ciencias de la Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Silvia Noemí Mazzeo

Tema

Un algoritmo de Colonia de Hormigas para el problema de Ruteo de Vehículos (CVRP)

Dirección de Tesis

Lic. Irene Loiseau

Abril del 2003

AGRADECIMIENTOS

Quiero agradecer a mis padres por la ayuda y el apoyo que me brindaron a lo largo de mi carrera y durante el desarrollo de esta tesis, sin los cuales no habría podido llegar a esto.
A mi hijo Manuel, por la paciencia que me tuvo durante todos estos meses, a quien prometo a partir de ahora dedicarle todo mi tiempo.
Quiero agradecer muy especialmente a Irene, mi directora de tesis, quien a pesar de sus innumerables ocupaciones, se "inventó" el tiempo para dirigir mi tesis.

A todos ellos, muchas gracias...

ABSTRACT

Existen muchos problemas de complejidad NP-hard para los cuales no es posible encontrar una solución óptima en un tiempo razonable. Una forma de resolverlos es implementar algoritmos heurísticos que generen buenos resultados en forma más o menos rápida, aunque la solución obtenida no sea la óptima. Estos algoritmos heurísticos pueden ser clasificados en dos grandes grupos: *Heurísticas clásicas* y *Metaheurísticas.* Las metaheurísticas son procedimientos generales para explorar el espacio de soluciones e identificar buenas soluciones y que pueden ser aplicados a diferentes problemas de optimización combinatoria con pocas modificaciones.

Colonia de Hormigas es una técnica metaheurística propuesta por Colorni, Dorigo y Maniezzo, y fue introducida aplicada al clásico problema del Viajante de Comercio (Traveling Salesman Problem o TSP), aunque más tarde fue utilizada para la resolución de otros problemas combinatoriales. El mismo trata de imitar el comportamiento de hormigas que habitan en colonias y van en búsqueda de su alimento.

Un problema de optimización combinatoria muy relacionado a TSP es el problema de Ruteo de Vehículos (en la literatura como VRP). Existen muchas variantes de este problema en la vida real, una de ellas es el problema Ruteo de Vehículos con limitación de capacidades de los vehículos (CVRP). El objetivo de esta tesis fue aplicar la técnica de Colonia de Hormigas a la resolución del problema CVRP.

ABSTRACT

There are many problems of complexity NP-hard for which it is not possible to find an optimal solution in a reasonable time. An option to solve them is to implement heuristics that quickly produce good results, although the solution found is not necessarily the optimal one. This kind of algorithms can be classified into two groups: *classical heuristics* and *metaheuristics.* Metaheuristics are general technique to explore the solution space to identify good solutions and that can be applied to different combinatorial optimization problems with few modifications.

Ant Colony is a metaheuristic technique introduced by Colorni, Dorigo and Maniezzo, and was first used on the Traveling Salesman Problem (TSP), and latter applied to others combinatorial problems. It simulate the behaviour of ant colonies searching for food.

One combinatorial optimization problem related to TSP is the Vehicle Routing Problem (VRP). There are many variants of this problem in real life, one of them is Capacitated Vehicle Routing Problem (CVRP). The purpose of this thesis was to apply Ant Colony technique to solve the CVRP problem.

1. INTRODUCCION

El *Problema del Viajante de Comercio* (*Traveling Salesman Problem* o *TSP*), ha obtenido mucha atención por parte de la comunidad científica por ser un problema fácil de describir y difícil de resolver y además por sus múltiples aplicaciones. El mismo consiste en, dado un conjunto de ciudades, visitar cada una de ellas exactamente una vez, al menor costo, donde dicho costo está dado por la suma de los costos de los arcos entre ciudades. TSP pertenece a la clase de problemas NP-Hard.

Otro problema de aplicación del mundo real, muy relacionado a TSP, es el problema de distribución de mercaderías entre depósitos y usuarios finales o clientes. Estos problemas se conocen con el nombre de *Problemas de Ruteo de Vehículos* (*Vehicle Routing Problem* o *VRP*) [27], y consisten en dado uno o más depósitos, un conjunto de clientes con una determinada demanda de mercadería y un conjunto de camiones con una determinada capacidad, establecer los circuitos de distribución de mercadería para cada camión desde el o los depósitos a cada cliente con el menor costo posible tal que se satisfagan las distintas restricciones planteadas por cada problema: capacidad de los camiones, requerimientos de los clientes, tiempos de entrega. Existen distintas variantes del problema VRP. Ellas son VRP con Capacidades (*Capacited VRP* o *CVRP*), VRP con Restricciones de Distancias (*Distance-Constrained VRP* o *DVRP*) , VRP con Ventanas de Tiempo (*VRP with Time Windows* o *VRPTW*) y VRP con Carga y Entrega (*VRP with Pickup and Delivery* o *VRPPD*). Las características de cada uno de ellos son:

CVRP.

Es la versión básica de VRP y la más estudiada. El problema consiste en entregar a cada cliente la cantidad de mercadería demandada por cada uno de ellos. Los vehículos son idénticos con base en un único depósito. El objetivo es minimizar el costo total del recorrido a realizar para cubrir a todos los clientes, cumpliendo las restricciones de capacidad de cada camión. CVRP pertenece a la clase de problemas NP-hard y generaliza al problema del viajante de comercio (TSP), el cual surge tomando K=1 y C $\geq$ demanda(V), siendo *K* cantidad de camiones, *C* capacidad del Vehículo y *V* conjunto de clientes.

DVRP.

En esta variante de CVRP se reemplaza la restricción de capacidad de cada camión por una restricción de longitud (o tiempo) máxima *T* para cada recorrido. A cada arco se le agrega un valor no negativo representando la longitud del mismo. En el caso de que tanto la restricción de capacidad como de longitud máxima de cada recorrido estén presente el problema se denomina CVRP *con Restricciones de Distancia* (DCVRP).

VRPTW.

En esta versión del problema cada cliente *i* tiene asociado un intervalo de tiempo $[a_i,b_i]$ de recepción de mercadería (llamado ventana de tiempo) y un tiempo de servicio S_i, cada arco (i,j) tiene asociado un tiempo de trayecto $t_{i,j}$. Se agrega la restricción de que el servicio de cada cliente debe comenzar dentro de su ventana de tiempo y debe permanecer en el mismo S_i instantes de tiempo. VRPTW es una generalización de CVRP, el cual surge poniendo $a_i=0$ y $b_i=+\infty$.

VRPPD.

En esta variante cada cliente *i* tiene asociadas dos cantidades d_i y p_i representando las cantidades de entrega y de carga de mercadería respectivamente. Además para cada cliente

i, O_i denota el vértice que es origen de la mercadería a ser entregada y D_i denota el vértice destino de la mercadería a cargar en el nodo *i*. Se agrega la restricción de que los nodos O_i, *i*, D_i deben ser visitados por el mismo vehículo y en ese mismo orden. VRPPD es una generalización de CVRP, el cual surge con $O_i = D_i = 0$ y $p_i = 0$.

Los algoritmos para resolver los problemas descriptos pueden dividirse en dos grandes grupos: *algoritmos Exactos* y *algoritmos Aproximados o Heurísticas*. Cualquier solución factible de un problema combinatorial de minimización, es decir, aquella que satisfaga las restricciones del problema, provee una cota superior del valor óptimo. Para estimar la desviación de una cota superior del valor óptimo es necesario conocer además una cota inferior de este valor. Si ambas cotas coinciden significa que el valor obtenido es el óptimo. Los métodos exactos utilizan algoritmos que calculan dichas cotas. Los algoritmos exactos garantizan encontrar la solución óptima en un número acotado de pasos, sin embargo estos algoritmos requieren mucho tiempo de proceso. Por ejemplo, la solución exacta de un problema TSP de 2392 nodos requirió un tiempo de proceso de 27 horas [16]. El lector interesado en métodos exactos para VRP puede consultar en ([20],[21],[27],[28],[29]).

Los algoritmos que producen soluciones factibles, representando cotas superiores del problema se denominan *Heurísticas*. Las mismas generan soluciones en un tiempo menor, pero sin poder establecerse que tan lejos se halla la solución obtenida del valor óptimo al finalizar el algoritmo. Las heurísticas se clasifican en dos grandes clases: *heurísticas clásicas*, las cuales se desarrollaron en su mayoría entre los años 1960 y 1990 y *metaheurísticas*, que corresponden a la década del 80 en adelante.

A su vez, las heurísticas clásicas de VRP pueden ser clasificadas en tres grupos: *heurísticas constructivas, heurísticas en dos fases* y *heurísticas de mejoramiento* (llamadas también *heurísticas locales*). Las primeras comienzan seleccionando un nodo en forma random del conjunto de nodos y luego continúan agregando nuevos nodos en forma incremental de acuerdo a alguna regla heurística. Las heurísticas en dos fases descomponen el problema en dos componentes: asignación de vértices a recorridos y luego construcción (ordenamiento) de los mismos, con posible feedback entre ambas fases. Las heurísticas de mejoramiento, parten de una solución determinada, intentando reducir el costo de la misma intercambiando ejes de acuerdo a alguna regla heurística, hasta alcanzar un mínimo local, es decir que ya no se pueda mejorar la solución aplicando dicha regla. Explicaremos con más detalle las heurísticas de mejoramiento, las cuales serán utilizadas en este trabajo de tesis. Para una lectura adicional sobre heurísticas clásicas ver ([4],[6],[8],[17],[18],[27]).

Las técnicas metaheurísticas son procedimientos generales para explorar el espacio de soluciones e identificar buenas soluciones y que pueden ser aplicados a diferentes problemas de optimización con pocas modificaciones. Por lo general están inspiradas en procesos naturales. Las mismas ponen énfasis en la ejecución de una exploración profunda de las regiones más prometedoras del espacio de soluciones. Estos métodos combinan reglas de exploración de vecindarios sofisticadas usando conocimiento específico del problema, estructuras de memoria y recombinación de soluciones. La calidad de la soluciones obtenidas por las metaheurísticas es superior a las producidas mediante las heurísticas clásicas pero a un costo de mayor tiempo de procesamiento. Ejemplo de ellas son *Simulated Annealing*, inspirada en el proceso de templado de metales, conocido como *annealing*; *Tabu Search*, que trata de modelar la memoria del ser humano recordando soluciones vistas previamente usando estructuras de datos simples; *Algoritmos Genéticos*, basados en mecanismos de selección natural sobre una población de seres vivos; y *Colonia*

de Hormigas, inspirada en el comportamiento de hormigas que habitan en colonias y van en búsqueda de alimento.

Colonia de Hormigas, propuesto por Colorni, Dorigo y Maniezzo (1996) [12], es una técnica metaheurística utilizada en la resolución de problemas combinatoriales y fue introducido aplicado al problema del viajante, aunque más tarde fue utilizado para la resolución de otros problemas combinatoriales. El objetivo de esta tesis es implementar un algoritmo de Colonia de Hormigas para el problema CVRP.

En la sección siguiente comenzaremos definiendo el problema a resolver (CVRP), en la sección 3 describiremos el algoritmo de Colonia de Hormigas tal cual fue presentado en [12] para luego continuar en la sección 4 explicando como aplicaremos el mismo al problema CVRP. En esta última sección describiremos distintas variantes a implementar, las estructuras de datos utilizadas y un Pseudo Código del algoritmo.

En las sección 5 daremos una breve descripción de la interfase del algoritmo, la cual permite configurar los parámetros del problema. En la sección 6 explicaremos los experimentos realizados y mostraremos los resultados obtenidos.

2. RUTEO DE VEHÍCULOS CON CAPACIDADES

2.1. Definición del Problema

Dado que Colonia de Hormigas fue introducido aplicado a TSP, comenzaremos definiendo el mismo. Dado un conjunto de *n* ciudades, TSP se define como el problema de encontrar el tour de longitud mínima que visite todas las ciudades pasando sólo una vez por cada una de ellas.

TSP

Sea $V = \{v_1,....,v_n\}$ un conjunto de ciudades , $A = \{(r,s) / r,s \in V, r \neq s\}$ el conjunto de arcos y sea $\delta(r,s) = \delta(s,r)$ el costo asociado con cada arco $(r,s) \in A$.

TSP es el problema de encontrar el tour cerrado ("circuito Hamiltoniano") de costo mínimo que visite cada ciudad solo una vez.

ATSP

Si $\delta(r,s) \neq \delta(s,r)$ para al menos un arco (r,s) entonces el problema se convierte en Asymmetric TSP (ATSP).

El problema de Ruteo de Vehículos con Capacidades consiste en dados :

- un conjunto de clientes a los cuales hay que despachar una determinada demanda de mercadería ,
- un conjunto de vehículos con una determinada capacidad los cuales están ubicados en uno o más depósitos,
- y los costos o distancias existentes entre el depósito y los clientes y los clientes entre sí,

establecer los distintos circuitos a cubrir por cada vehículo que comiencen y terminen en el depósito, tal que el costo global del transporte sea mínimo y que el total de demanda de mercadería de cada circuito no exceda la capacidad del vehículo correspondiente.

CVRP

Sea G = (V,A) un grafo completo, V = $\{i_0,....,i_n\}$ es el conjunto de vértices representando a los clientes, donde i_0 corresponde al depósito, A = { (i,j) / i,j $\in$ V, i≠j } es el conjunto de arcos, y sea $\delta(i,j) = \delta(j,i)$ el costo asociado con cada arco (i,j) $\in$ A. Cada vértice *i* está asociado con una demanda $d_i > 0$ y el depósito con una demanda ficticia $d_0 = 0$.
Dado un conjunto *S* de vértices llamemos $d(S) = \sum_{i \in S} d_i$, la demanda total del conjunto.
Existe una cantidad *K* de vehículos, todos con la misma capacidad C_k, tal que $K \geq K_{min}$, donde K_{min} es la mínima cantidad de vehículos necesaria para atender a todos los clientes.

CVRP consiste en encontrar un conjunto de exactamente *K* circuitos, cada uno correspondiente a la ruta de un vehículo, con costo mínimo definido como la suma de los costos de los arcos de cada circuito tal que

(i) cada circuito comience y termine en el depósito.
(ii) cada cliente sea visitado exactamente por un circuito.
(iii) $d(S) \leq C_k$, donde *S* es el conjunto de vértices visitados por el vehículo *k*.

ACVRP

Si $\delta(i,j) \neq \delta(j,i)$ para al menos un arco (i,j) entonces el problema se convierte en Asymmetric CVRP (ACVRP).

2.2. Modelo Matemático

Aunque no será usada en el algoritmo que implementaremos, en esta sección describiremos la formulación de CVRP como problema de Programación Lineal Entera. La misma está basada en los modelos propuestos en [27]; la restricción (8) fue extraída del modelo propuesto por [24].

El modelo aquí descripto es una formulación de flujo de vehículos de dos índices el cual utiliza $O(n^2)$ variables binarias *x* que indican si el vehículo atraviesa un arco que se encuentra en la solución óptima. Además es necesario especificar que vehículo atraviesa cada arco. Para ello se agregan O(n.K) variables binarias $y_{i,k}$, siendo *K* el número de camiones con i $\in$ V, k =1,...., K que toman valor 1 si el nodo *i* es visitado por el camión *k*.

Definamos

V = {0,....,n} conjunto de vértices, donde 0 corresponde al depósito.
A conjunto de arcos en V.
S subconjunto de V ($S \subseteq V$).
$c_{i,j}$ el costo asociado al arco (i,j) $\in$ A.
d_i demanda del nodo i.
$x_{i,j}$ = 1 si el arco (i,j) $\in$ A pertenece a la solución óptima, 0 sino.
$y_{i,k}$ = 1 si el camión k visita el nodo i, 0 sino.
$\tau(S)$ mínima cantidad de camiones necesarias para cubrir los clientes de S.

CVRP $\min \sum_{i \in V} \sum_{j \in V} c_{i,j} \cdot x_{i,j}$

sujeto a

(1) $\sum_{i \in V} x_{i,j} = 1 \qquad \forall j \in V \setminus \{0\}$

(2) $\sum_{j \in V} x_{i,j} = 1 \qquad \forall i \in V \setminus \{0\}$

(3) $\sum_{i \in V} x_{i,o} = K \qquad \forall i \in V \setminus \{0\}$, K cantidad de camiones.

(4) $\sum_{j \in V} x_{o,j} = K \qquad \forall j \in V \setminus \{0\}$, K cantidad de camiones.

(5) $\sum_{i \in V} d_i \cdot y_{i,k} \leq C \qquad \forall k=1 \ldots K$, C capacidad del camión.

(6) $\sum_{k=1}^{K} y_{i,k} = 1 \qquad \forall i \in V \setminus \{0\}$

(7) $\sum_{k=1}^{K} y_{o,k} = K$

(8) $(y_{i,k} - y_{j,k}) \leq 1 - (x_{i,j} + x_{j,i}) \qquad \forall i,j \in V \setminus \{0\},\ k = 1,\ldots,K$

(9) $\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - \tau(S) \qquad \forall S \subseteq V \setminus \{0\},\ S \neq \varnothing$

(10) $x_{i,j} \in \{0,1\} \qquad \forall i,j \in V$

(11) $y_{i,k} \in \{0,1\} \qquad \forall i \in V,\ k = 1,\ldots,K$

La restricción (1) del modelo asegura que la cantidad de arcos entrantes a cada nodo utilizados en la solución sea exactamente uno, lo mismo la (2) para los arcos salientes. Las restricciones (3) y (4) aseguran que los arcos entrantes y salientes al depósito utilizados en la solución sean exactamente igual a la cantidad de camiones.

La restricción (5) asegura que la cantidad de mercaderías entregadas en un viaje no

supera la capacidad del camión, para todo camión *k*.

La restricción (6) garantiza que solamente un camión visite a cada nodo *i* mientras que la (7) que exactamente *K* camiones visiten el depósito.

La restricción (8) dice que si existe un arco que une dos nodos cualesquiera *i* y *j* en la solución, entonces el camión que visita a ambos es el mismo y además que el arco se usa en una única dirección.

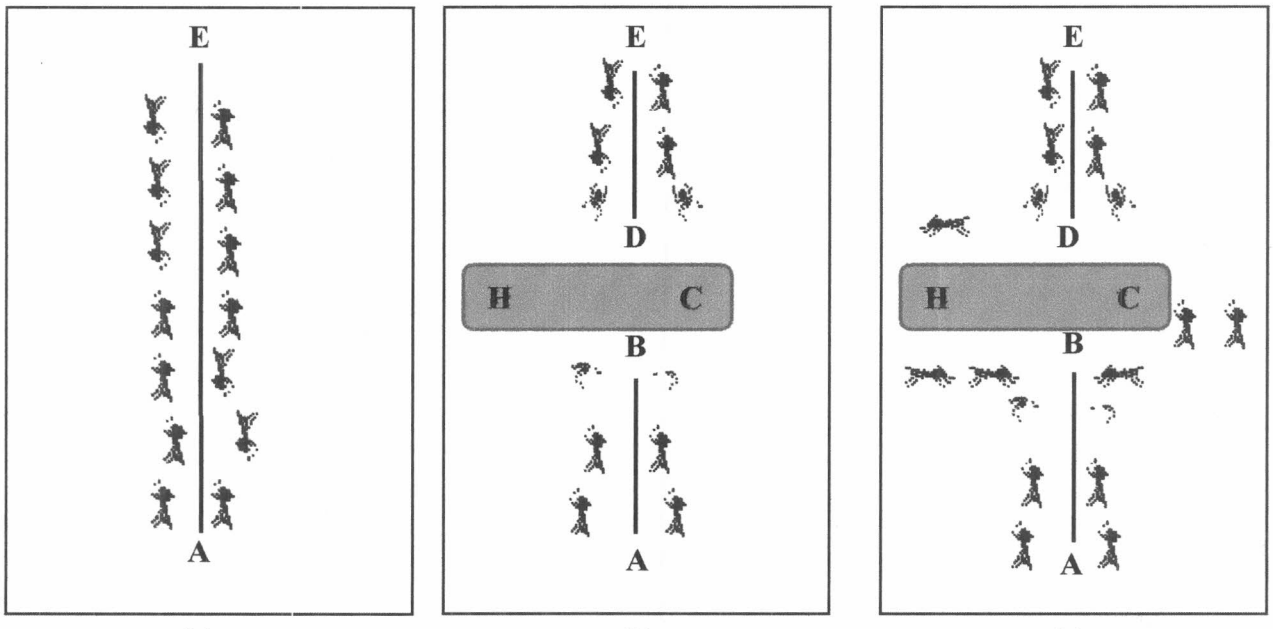
La restricción (9) es una regla de eliminación de subtours que impone que al menos $\tau(S)$ arcos abandonen cada conjunto de nodos *S*. La cantidad de restricciones de este tipo es exponencial.

En la sección 4.5. explicaremos en que forma implementaremos las restricciones aquí descriptas en nuestro algoritmo. El lector interesado en Modelos para VRP puede consultar en [27].

3. DESCRIPCIÓN DE COLONIA DE HORMIGAS

3.1. Introducción

Uno de los problemas estudiados por los entomólogos es entender como las hormigas pueden establecer casi a ciegas la ruta más corta desde sus colonias hacia las fuentes de alimento y viceversa. Se descubrió que la forma utilizada para comunicar información entre individuos y emplearla para decidir por donde seguir es a través de *rastros de pheromona*. A medida que una hormiga se mueve va dejando cantidades variables de pheromona sobre el suelo, marcando de esta forma el camino con rastros de esta sustancia. Mientras una hormiga aislada se mueve esencialmente en forma random, una hormiga que detecta estos rastros dejados previamente por otra hormiga puede decidir con una alta probabilidad de seguir este rastro, reforzando de esta forma el mismo con su propio pheromona. Cuanto más hormigas sigan este rastro más atractivo se vuelve el mismo para que otras hormigas lo sigan. Consideremos el ejemplo mostrado en la Figura 1.



(a) (b) (c)

Figura 1

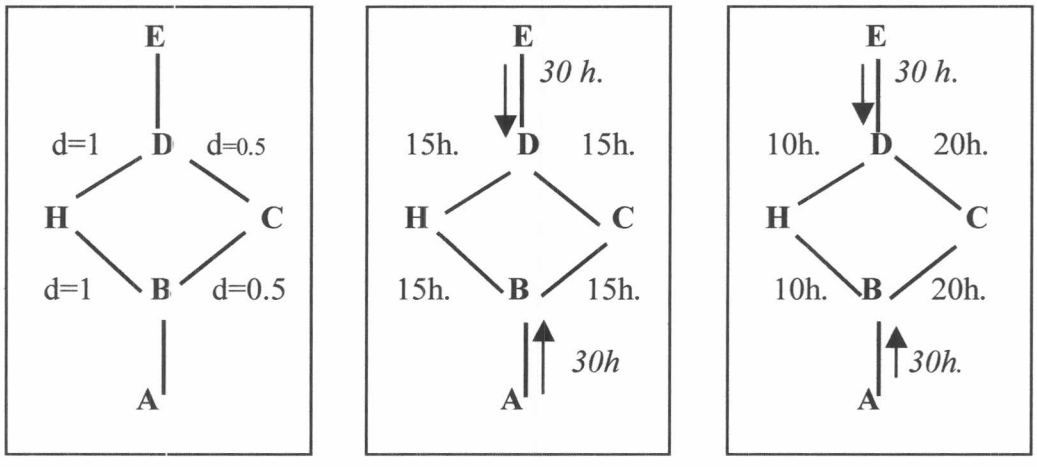
Existe un camino que deben recorrer las hormigas para alcanzar su alimento desde su colonia y volver, es decir de A a E y viceversa (Figura 1a). De repente aparece un obstáculo que corta el camino. Por lo tanto, en el punto B las hormigas que van de A a E (o en la posición D para las hormigas que recorren el camino opuesto) deben decidir si ir a la derecha o izquierda (Figura 1b). La elección es influenciada por el nivel de pheromona dejado por las hormigas precedentes. A nivel más alto de pheromona en el camino hacia la derecha le da a cada hormiga un estímulo mayor y por lo tanto una mayor probabilidad de elegir dicho camino. La primera hormiga en alcanzar el punto B (o D) tiene la misma probabilidad de ir hacia la derecha o izquierda, ya que aún no existe pheromona en ninguno de los dos caminos. Como el camino BCD es más corto que el BHD la primera hormiga que eligió dicho camino llegará antes al punto D que la primera hormiga que eligió el camino BHD. El resultado es que las hormigas que están volviendo de E a A encontrarán mayor nivel de pheromona en el camino BCD que en el BHD provocados por

las hormigas que por azar decidieron volver tomando el camino DCBA y por las que ya llegaron a través de BCD, por lo tanto ellas preferirán con una mayor probabilidad el camino DCB. Como consecuencia el número de hormigas recorriendo el camino BCD por unidad de tiempo será mayor que el número que recorra el camino BHD, causando que la cantidad de pheromona depositado sobre el camino más corto crezca más rápidamente que en el camino largo. El resultado final es que rápidamente todas las hormigas elegirán el camino más corto.

Ahora consideremos el gráfico de la Figura 2, que es una interpretación posible del ejemplo anterior. Supongamos que la distancia ente B-H y entre H-D es 1 y que la distancia B-C-D suma 1 estando C ubicada en la mitad de camino entre B y D (Fig. 2a). Ahora consideremos que es lo que sucede en diferentes intervalos de tiempo: t=0,1,2... . Supongamos que 30 nuevas hormigas llegan al punto B desde A, y 30 al punto D desde E en cada intervalo de tiempo, cada hormiga camina a una velocidad de 1 por unidad de tiempo y que mientras camina una hormiga deposita en el tiempo *t* un rastro de pheromona de intensidad 1 y que el mismo se evapora completamente en el medio de intervalos de tiempo sucesivos (t+1,t+2).

En el intervalo t=0 no hay rastro, pero hay 30 hormigas en B y 30 en D. La decisión de qué camino seguir es completamente al azar. Por lo tanto, desde cada nodo 15 hormigas en promedio irán hacia H y 15 hacia C. (Fig. 2b).

En el intervalo t=1 30 nuevas hormigas que llegan a B desde A encuentran un rastro de intensidad 15 en el camino que va hacia H , depositado por las 15 hormigas que decidieron ir hacia B y una intensidad de 30 en el camino que va hacia C depositado por las 15 hormigas que decidieron ir por C más el rastro de las 15 hormigas que llegaron a B desde D via C. (Fig. 2c). Como la probabilidad de elegir un camino está influenciada por la cantidad de pheromona, entonces la cantidad esperada de hormigas que elijan ir hacia C será el doble que las que elijan H. Lo mismo vale para las hormigas que llegan a D desde E. El proceso continuará hasta que eventualmente todas las hormigas elijan el camino más corto.



(a) (b) (c)

Figura 2

3.2. Algoritmo Básico de Colonia de Hormigas

La idea destacada de esta técnica es la de paralelización en la búsqueda de una solución basada en una estructura de memoria dinámica que incorpora información de la efectividad de los resultados obtenidos con anterioridad.

Una hormiga se define como un agente computacional, la cual en cada iteración *t* construye una solución del problema a resolver. En cada paso σ cada hormiga produce una solución parcial. Cada solución parcial se ve como estados; cada hormiga se mueve de un estado *i* hacia otro *j* correspondiente a una solución parcial más completa. En cada paso σ cada hormiga *k* arma un conjunto A_k de estados de expansión posibles a partir de su estado actual y se moverá hacia algún estado de dicho conjunto de acuerdo a una función de probabilidad basada en los mismos. Para cada hormiga *k* la probabilidad $P_k(i,j)$ de moverse del estado *i* actual hacia algún estado *j* se calcula en base a dos valores:

a) la atractividad η del movimiento, calculada a partir de alguna heurística basada en el conocimiento del problema.
b) El nivel de pheromona τ del movimiento, indicando que tan eficiente fue en el pasado la elección del mismo.

La fórmula que define la probabilidad de transición de cada estado utiliza para cada hormiga *k* una lista tabú, $tabu_k$, indicando el conjunto de movimientos prohibidos. Las probabilidades se computan utilizando la siguiente fórmula:

$$P_k(i,j) = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha} \cdot [\eta(i,j)]^{\beta}}{\sum_{(i,z) \notin Tabu_K} [\tau(i,z)]^{\alpha} \cdot [\eta(i,z)]^{\beta}} & \text{si } (i,j) \notin Tabu_k \\ 0 & \text{si } (i,j) \in Tabu_k \end{cases} \qquad (1)$$

donde α , β son parámetros que determinan la importancia relativa de η versus τ (α >0, β>0).

Luego de cada iteración *t*, los niveles de pheromona se actualizan en base a la siguiente fórmula:

$$\tau(i,j) = \varphi \cdot \tau(i,j) + \Delta\tau(i,j) \qquad (2)$$

donde φ es un coeficiente definido por el usuario que representa el nivel de persistencia de pheromona y $\Delta\tau(i,j)$ representa la suma de las contribuciones de todas las hormigas que utilizaron el movimiento (i,j) en la construcción de su solución.

El procedimiento general de Colonia de Hormigas se describe en el siguiente cuadro:

Inicialización de niveles de pheromona, inicialización de parámetros.

Loop /*Para cada iteración *t* */
    Inicializar hormigas y sus soluciones
    Loop /* Para cada paso σ */
        For cada hormiga *k*
            Armar un conjunto A_k de estados de expansión posibles a partir del estado actual de la hormiga *k*.

            Aplicar la regla de transición de estados sobre el conjunto A_k y elegir un nuevo estado j.
        End
    Until *todas las hormigas hayan generado una solución completa del problema.*
    Actualizar niveles de pheromona.
    Almacenar la mejor solución encontrada.
Until *condicion de finalización*

3.3. Aplicación de Colonia de Hormigas al problema TSP

Hasta aquí hemos descripto las ideas generales de la técnica de Colonia de Hormigas; a continuación explicaremos como se aplican las mismas al Problema del Viajante de Comercio (TSP) para mayor simplicidad. En las secciones siguientes se explicará como se adapta el mismo para resolver el problema de ruteo de vehículos (VRP), que es el objetivo de esta tesis.

Cada hormiga es un agente con las siguientes características:

- En cada paso elige una ciudad a visitar con una probabilidad que es función de la distancia a dicha ciudad y de la cantidad de pheromona presente en el arco que la conecta.

- No puede elegir una ciudad de las ya visitadas hasta completar un tour.

- Cuando completa un tour, cada hormiga deposita una cantidad de pheromona sobre cada arco del tour.

Sea $\tau(i,j)$ la intensidad de pheromona del arco (i,j). Cada hormiga elige en el tiempo *t* la ciudad que visitará en el tiempo t+1. Luego, si llamamos una iteración del algoritmo los *m* movimientos realizados por cada una de las *m* hormigas, entonces luego de *n* iteraciones (a las cuales llamamos un ciclo) cada hormiga habrá completado un tour.
Al final del ciclo se actualiza la cantidad de pheromona de los arcos de acuerdo con la siguiente fórmula:

$$\tau(i,j) = \varphi \cdot \tau(i,j) + \sum_{k=1}^{m} \Delta\tau_k (i,j) \qquad \textbf{(3)}$$

donde φ es el coeficiente de *persistencia de pheromona* ($0<\varphi<1$) , tal que $(1-\varphi)$ representa la *evaporación de pheromona* entre los intervalos t y (t+n) y $\Delta\tau_k$ (i,j) representa la cantidad de pheromona depositado en los arcos por la hormiga *K* entre los intervalos t y (t+n) dada por los siguientes valores :

$$\Delta\tau_k (i,j) = \begin{cases} Q / L_k & \text{si (i,j) pertenece al tour generado por la hormiga K} \\ 0 & \text{sino} \end{cases} \qquad \textbf{(4)}$$

donde *Q* es una constante y L_k es la longitud del tour generado por la hormiga *k*. La intensidad de pheromona en el intervalo t=0 se define como una constante positiva de valor pequeño τ_0 para todos los arcos.

En cada intervalo *t* del algoritmo cada hormiga debe decidir cual es la próxima ciudad a visitar. Para ello se define una regla de transición de estados llamada *regla random-proporcional* que devuelve la probabilidad con la cual una hormiga *k* posicionada en la ciudad *i* elige moverse hacia la ciudad *j*. La misma está basada en el costo (distancia) de cada arco a elegir y el nivel de pheromona del mismo.

Definimos la probabilidad de transición de *i* a *j* para la hormiga *k* como:

$$P_k(i,j) = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha} \cdot [\eta(i,j)]^{\beta}}{\sum_{z \in Vecinos_k(i)} [\tau(i,z)]^{\alpha} \cdot [\eta(i,z)]^{\beta}} & \text{si } j \in Vecinos_k(i) \\ 0 & \text{sino} \end{cases} \qquad \textbf{(5)}$$

donde

τ (i,j) es la cantidad de pheromona del arco.

η(i,j) es una cantidad a la cual llamamos visibilidad y que en este caso representaremos como $1/\delta(i,j)$, siendo δ(i,j) la distancia del arco (i,j).

$Vecinos_k(i)$ es el conjunto de ciudades que restan ser visitadas por la hormiga *k* posicionada en la ciudad *i*.

α , β son parámetros que determinan la importancia relativa de la visibilidad versus pheromona ($\alpha > 0$, $\beta > 0$).

El algoritmo básico consta de las siguientes etapas principales:

- *Etapa de inicialización*: se ubican las hormigas en diferentes ciudades y se inicializan los arcos con un valor inicial de pheromona. Para satisfacer la restricción de que cada hormiga visite las *n* diferentes ciudades, se asocia a cada hormiga una estructura de datos llamada *Lista Tabú*, que almacena las ciudades ya visitadas y prohibe que una hormiga vuelva a elegir alguna ciudad de la lista hasta completar un tour. El primer elemento de cada lista se configura como la ciudad inicial de la hormiga *k*.

- *Etapa de construcción*. En esta etapa cada hormiga genera una nueva solución. Para elegir la siguiente ciudad cada hormiga utiliza la regla de transición definida en **(5)** que es función de la distancia y de la cantidad de pheromona. Haciendo $\alpha=0$, se ignora la cantidad de pheromona del arco, convirtiéndose en un algoritmo goloso con múltiples entradas. Luego de *n* iteraciones cada hormiga completó un tour.

- *Etapa de actualización*. Se calcula para cada hormiga *k* el valor L_k y los valores $\Delta\tau_k(i,j)$ de acuerdo a la regla en **(4)** y se actualizan las cantidades de pheromona de los arcos. Se almacena el tour de longitud mínima obtenido y se vacían las listas Tabú.

Este proceso se repite durante un número de ciclos o hasta que todas las hormigas obtengan el mismo tour. Ante este comportamiento de estancamiento se finaliza el algoritmo.

4. APLICACIÓN DE COLONIA DE HORMIGAS AL PROBLEMA CVRP

En esta sección describiremos como aplicaremos la técnica de Colonia de Hormigas a la resolución de CVRP. Además mencionaremos y compararemos algunas variantes al algoritmo estándar propuestas por varios autores, las cuales serán implementadas en nuestro algoritmo.

4.1. Terminología a Utilizar

Recorrido : Se refiere al conjunto de nodos visitados por un camión. La solución del problema comprenderá de *kMin* recorridos, donde *kMin* representa la mínima cantidad de camiones necesarios para cubrir todos los nodos.

Tour : Se refiere a la solución completa la cual comprende de *kMin* recorridos visitando los *n* nodos clientes. Usaremos indistintamente los términos *tour* o *solución*. Durante el proceso de ejecución del algoritmo se generará un tour por hormiga.

4.2. Descripción General del Algoritmo

En nuestra implementación propuesta el grafo esta formado por *n* nodos clientes y un nodo *s* depósito de partida y llegada. Cada nodo cuenta con un multiarco que indica cual es el camión que lo visita. La capacidad del multiarco es como máximo de 1 para los nodos clientes y de *M* para el nodo *s*, donde *M* es la cantidad de camiones disponibles. Como la única forma de visitar un nodo es a través de un multiarco camión, la restricción (1) del modelo matemático presentado en la sección 2.2. se consigue pidiendo que la cantidad de multiarcos utilizados en cada nodo sea exactamente 1 para todo nodo cliente y a lo sumo *M* para el nodo *s*.

Cada hormiga genera un tour o solución. Los mismos se representan por un vector de *H* elementos (siendo *H* la cantidad de hormigas), donde cada elemento *h* representa la solución de una hormiga.

Proponemos dos métodos para la construcción de cada solución:

Método 1. Camiones consecutivos.
Cada hormiga comienza a armar su recorrido con el primer camión, eligiendo nodos clientes hasta agotar la capacidad del camión. Luego sigue con el siguiente camión hasta visitar todos los nodos. Cada hormiga comienza desde un nodo cliente distinto.

Método 2. Camiones en Paralelo.
Cada hormiga avanza los recorridos de los distintos camiones en paralelo eligiendo en cada paso el arco más prometedor de los *M* recorridos. Cuando un recorrido agota su capacidad, se elige el nodo depósito y se da por terminado el mismo. En cada iteración del algoritmo se agrega sólo un vecino, eligiendo solamente un recorrido. La forma de elegir el siguiente vecino en cada recorrido es utilizando la regla de transición de estados propuesta. Luego, para decidir que recorrido extender, se elige el que obtuvo mejor valor.

4.3. Variantes Propuestas

En la sección 3.3 se describió el algoritmo en su forma estándar. A continuación describiremos algunas variantes que han sido propuestas por distintos autores, que serán implementadas para realizar comparaciones de los resultados obtenidos por las mismas.

La primer variante propuesta tiene que ver con la forma en que se realiza la actualización de pheromona de los arcos. El algoritmo original propone realizar actualización *global* al final de cada ciclo, donde todas las soluciones encontradas contribuyen en la actualización de arcos; en nuestra implementación aplicaremos distintas variantes de actualización global ([2],[12],[13]), como ya explicaremos. Además se experimentará haciendo actualizaciones *locales,* las mismas son realizadas en cada iteración del algoritmo, es decir que cada hormiga deja su rastro en cada paso, sin esperar al final del tour ([13],[15],[11]).

Se propone también una regla de transición alternativa ([13]), se explicará el uso de listas de candidatos para problemas grandes ([2],[13]) y la incorporación de heurísticas de mejoramiento ([18],[19],[25]).

4.3.A. Actualización Global

Se han propuesto tres formas de actualización global: *Todas las soluciones*, *Hormigas elitistas* y *Mejor solución.*

Todas las soluciones.
El primer método de actualización global es el propuesto en la versión original del algoritmo, en la cual se utilizan todas las soluciones obtenidas por las *h* hormigas para actualizar los arcos.

Hormigas elitistas
Llamamos así a las hormigas que obtuvieron las mejores soluciones. Este método propone clasificar las soluciones obtenidas y actualizar los arcos utilizando solo las mejores de la siguiente manera:

$$\tau(i,j) = \varphi \cdot \tau(i,j) + \sum_{\mu=1}^{\sigma-1} \Delta\tau\mu\,(i,j) + \sigma \cdot \Delta\tau^{*}\,(i,j)$$

donde

φ es el factor de persistencia de pheromona, σ es la cantidad de hormigas elites,

$\Delta\tau\mu\,(i,j) = (\sigma - \mu) / L\mu$ si $(i,j) \in$ a la solución de la μ-ésima mejor hormiga, sino 0, siendo $L\mu$ la solución de la μ-ésima hormiga.

$\Delta\tau^{*}\,(i,j) = 1 / L^{*}$ si $(i,j) \in$ a la mejor solución, sino 0, siendo L^{*} la mejor solución.

Es decir que los arcos pertenecientes a la mejor solución, son enfatizados como si σ

hormigas elitistas hubiesen utilizado los mismos en su solución obtenida.

Mejor solución.
Este método utiliza solo la mejor solución obtenida para actualizar los arcos de la siguiente forma:

$$\tau(i,j) = \varphi \cdot \tau(i,j) + \Delta\tau(i,j)$$

donde L* es la mejor solución y

$$\Delta\tau(i,j) = \begin{cases} 1 / L^* & \text{si (i,j) pertenece a la mejor solución.} \\ 0 & \text{sino.} \end{cases}$$

4.3.B. Actualización Local

La actualización local se realiza de la siguiente forma: cuando una hormiga se mueve del arco *i* al *j* en cada paso de construcción de su solución, la cantidad de pheromona del arco recorrido (i,j) se actualiza de acuerdo a la siguiente regla :

$$\tau(i,j) = \varphi \cdot \tau(i,j) + (1-\varphi)\ \Delta\tau(i,j) \qquad \textbf{(6)}$$

Los distintos valores para el término $\Delta\tau(i,j)$ dan origen a los métodos de actualización local que explicaremos a continuación.

Q-learning.
Este método de actualización está inspirado en Q-learning, un algoritmo desarrollado para la resolución de problemas de aprendizaje asistido. Dicho algoritmo cuenta con un agente el cual en cada posible estado en que se encuentre, debe aprender cual es la mejor acción a llevar a cabo usando como única información un número escalar que representa la evaluación del estado alcanzado luego de haber ejecutado cada acción elegida. Como el problema a resolver por las hormigas es similar al problema de aprendizaje asistido (ya que las hormigas deben decidir a que ciudad moverse en función de su ubicación actual) aplicaremos la misma fórmula utilizada por Q-learning para el término $\Delta\tau_k(i,j)$, que corresponde al valor del próximo estado.

Definimos:

$$\Delta\tau_k(i,j) = \gamma \cdot \max_{z \in Vecinos_k(j)} \tau_k(j,z) \qquad (0 \leq \gamma < 1)$$

Es decir, actualiza el rastro de pheromona del arco usando la cantidad de pheromona del arco de mayor valor entre los arcos correspondientes a la ciudad *j*, que es el próximo estado a alcanzar.

Pheromona inicial
En esta segunda variante definimos

$$\Delta\tau(i,j) = \tau_0$$

siendo τ_0 el nivel inicial de pheromona de los arcos del grafo.

Proporcional a la distancia.
Con esta variante queremos lograr que los arcos más cortos sean más deseables para las hormigas que los más largos. Para ello definimos:

$$\Delta\tau(i,j) = \tau_0 / \delta(i,j)$$

Evaporación.
Consiste en evaporar en cada iteración local la cantidad de pheromona de los arcos depositada por la actualización global. Es decir, definiendo

$$\Delta\tau_k(i,j) = 0$$

la fórmula de actualización resultante es

$$\tau(i,j) = \varphi \cdot \tau(i,j)$$

que representa menor persistencia de pheromona en cada iteración.

La aplicación de la regla **(6)** tiene como finalidad disminuir la cantidad de pheromona de los arcos utilizados, haciéndolos menos atractivos. El objetivo de la actualización local es que los arcos visitados se vuelvan menos deseables y por lo tanto sean elegidos con una probabilidad menor por las hormigas restantes en los pasos siguientes del algoritmo. Como consecuencia, las hormigas nunca convergen a una solución común. Con esto se favorece la exploración de arcos aún no visitados y la diversidad en la generación de soluciones. ([13],[15]).

4.3.C. Regla de Transición Pseudo-Random-Proporcional

La regla de transición estándar calcula las probabilidades de cada vecino posible y en forma random elige un vecino de acuerdo a las probabilidades calculadas. En esta sección describiremos una nueva regla, la cual combina selección random con selección de mejor opción. La idea es la siguiente: sean q_0 un valor parámetro tal que $0 \leq q_0 \leq 1$, y q un número random uniformemente distribuido en [0..1]; cada vez que una hormiga ubicada en la ciudad i tiene que elegir una ciudad j , se genera un número random q , $0 \leq q \leq 1$.
Si $q \leq q_0$, entonces se elige el arco de mejor valor (explotación), sino se elige probabilísticamente de acuerdo a la regla de transición **(5)**, descripta en la sección 3.3 (exploración guiada). La regla propuesta de selección de la ciudad j se define como:

$$j = \begin{cases} \underset{u \in Vecinos_k(i)}{\text{Max}} \{ [\tau(i,u)]^{\alpha} \; [\eta(i,u)]^{\beta} \} & \text{si } q \leq q_0 \quad \textit{(explotación)} \\ \mathbf{J} & \text{si } q > q_0 \quad \textit{(exploración)} \end{cases}$$

donde J es una variable random seleccionada de acuerdo a la distribución de probabilidad definida en **(5).** El parámetro q_0 determina la importancia de explotación versus exploración.

4.3.D. Listas de Candidatos

Si el problema es muy grande, la búsqueda del siguiente vecino entre todas las ciudades aún no visitadas es un proceso que puede consumir mucho tiempo. Una práctica común es utilizar una estructura de datos estática llamada *lista de candidatos*. La misma consiste en una lista de ciudades preferidas, es decir, por cada ciudad *i* existe una lista de vecinos de un determinado tamaño (bastante menor al número de ciudades) de las ciudades más cercanas a *i* ordenadas por su distancia en forma creciente. Cada hormiga busca primero en esta lista; solamente si ninguna de estas ciudades puede ser visitada, entonces considera el resto de las ciudades.

4.3.E. Heurísticas de Mejoramiento

Las heurísticas de mejoramiento parten de una solución ya construída e intentan reducir su longitud, intercambiando arcos de acuerdo a alguna regla heurística hasta alcanzar algún mínimo local (es decir, hasta que ninguna mejora sea posible utilizando dicha regla). Las heurísticas de mejoramiento para VRP se aplican a cada recorrido en forma separada o a varios recorridos a la vez. Para el primer caso, las mismas heurísticas de mejora de TSP pueden ser aplicadas.

Las heurísticas de mejoramiento más utilizadas en ruteo de vehículos son los algoritmos de intercambio de ejes, en los cuales un número variable de ellos son intercambiados. Una práctica general es usar una heurística constructiva para generar una solución y luego aplicar una de mejoramiento para optimizarla.

Para aplicar heurísticas de mejoramiento a Colonia de Hormigas, agregaremos una nueva etapa al algoritmo general, posterior a la etapa de construcción. Es decir, luego de que todas las hormigas hayan completado su solución, y antes de realizar la actualización global de pheromona de los arcos, se incorporará una etapa de mejoramiento de las soluciones construídas utilizando alguna de las heurísticas que explicaremos en esta sección.

Ya que en el caso de VRP tenemos múltiples recorridos, distinguiremos entre intercambio de ejes para un recorrido simple (*Intercambio Simple*) o para recorridos múltiples (*Intercambio Múltiple*).

La técnica de mejoramiento que proponemos en esta tesis consiste en realizar varios ciclos de *Intercambio Múltiple* hasta que no se obtenga ninguna mejora en el costo y luego en cada recorrido realizar *Intercambio Simple*.

Intercambio Simple

Los algoritmos de intercambios se definen en base a un tipo de operación (*intercambio o movimiento*) que convierte una solución en otra. Dada una solución inicial, dichos algoritmos repetidamente aplican operaciones de intercambios hasta alcanzar un tour en el cual ya no es posible obtener mejora. Una operación que reemplaza un conjunto de k ejes por otros k nuevos ejes se llama k-intercambio y un tour que ya no puede ser mejorado realizando k-intercambios se dice k-óptimo. La verificación de k-optimalidad requiere $O(n^k)$ operaciones.

El algoritmo básico de *k-optimalidad* (referido como *k-opt*) fue propuesto por *Lin* ([22],[18]). Los más utilizados han sido los algoritmos *2-opt* y *3-opt*. Varias modificaciones al algoritmo básico han sido presentadas. Lin and Kernighan propusieron una variante en la cual el valor de k se modifica dinámicamente durante el transcurso del algoritmo ([23],[17]).

El método que implementaremos en este trabajo es el propuesto por Or ([25],[19]). El mismo consiste en reubicar un conjunto de l nodos consecutivos iterando hasta que no sea posible ninguna mejora. Este tipo de modificación implica el reemplazo de 3 ejes del tour, es decir que los intercambios realizados son un subconjunto de los intercambios generados por el método *k-opt* con k=3. En este caso utilizaremos l=1. Esta operación de intercambio, que explicamos en los gráficos de las Figuras 3 y 4, se denomina *Or-intercambio*. El mismo corresponde a un *Or-intercambio forward* ya que el nodo se reubica en una posición posterior dentro del tour, caso contrario el intercambio se denomina *Or-intercambio backward.*

Sea el siguiente recorrido:

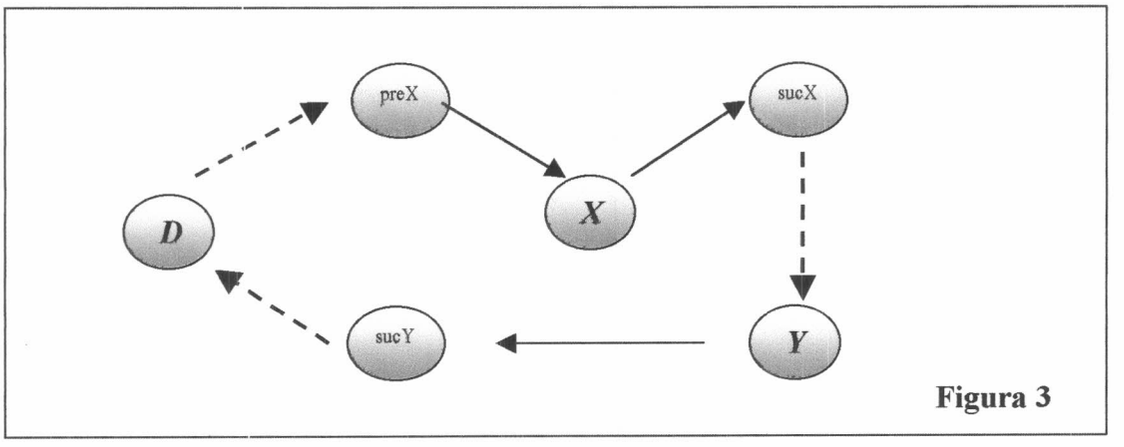


Figura 3

el nuevo recorrido obtenido de reubicar el nodo X entro los nodos Y y Suc_y es el siguiente

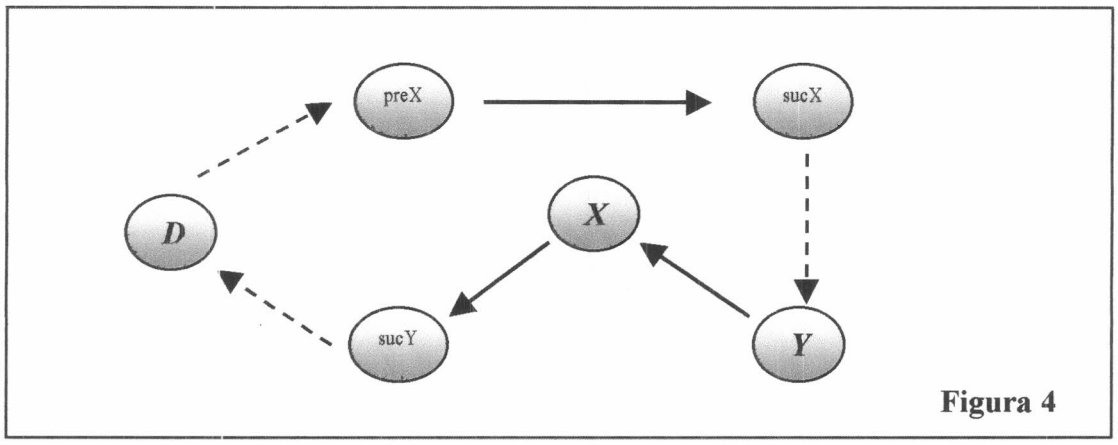


Figura 4

que implica el intercambio de los ejes (Pre_x,X) , (X,Suc_x) y (Y,Suc_y) por los ejes (Pre_x ,Suc_x) , (Y, X) y (X,Suc_y)

Proponemos el siguiente algoritmo de intercambio simple: se elige un nodo a reubicar en forma consecutiva entre todos los nodos del recorrido; para elegir la posición donde reubicar dicho nodo, se recorre el tour desde la posición del nodo elegido hasta encontrar un nodo tal que la ganancia **G** obtenida de intercambiar los mismos sea mayor que 0. Si $\mathbf{G} > 0$ se realiza el intercambio entre ambos nodos.

Definimos **G** mediante la siguiente regla:

$$\mathbf{G} = [\, d(Pre_x,X) + d(X,Suc_x) + d(Y,Suc_y)\,] - [\, d(Pre_x ,Suc_x) + d(Y, X) + d(X,Suc_y)\,]$$

donde d(i,j) indica la distancia o costo del arco (i, j). Este proceso se repite hasta que ya no haya mejora en el costo o hasta alcanzar una cantidad de ciclos de mejora límite (dado como parámetro de entrada). Al final de esta sección exponemos el Pseudo-Código del algoritmo.

Intercambio Múltiple

La construcción de una solución VRP requiere de dos tipos de decisiones: *Asignación,* para determinar que camión prestará servicio a que clientes; *Ruteo,* para determinar en que orden los clientes asignados serán visitados. La técnica de *Intercambio Simple* anteriormente descripta trata de mejorar las decisiones de ruteo, en cambio la de *Intercambio Múltiple* que a continuación describiremos se ocupa de mejorar las asignaciones realizadas [19].

Existen 3 tipos de operaciones de Intercambio Múltiple las cuales se ocupan de reubicar nodos de dos recorridos distintos. Ellas son *Reubicación, Intercambio* y *Cruza.*

Explicaremos brevemente cada una de ellas.

Reubicación. Consiste en reubicar un nodo en otro recorrido destino. Por ejemplo, dados los recorridos

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_x \rightarrow X \rightarrow Suc_x \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Y \rightarrow Suc_y \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

obtenemos luego de reubicar el nodo X

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_x \rightarrow Suc_x \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Y \rightarrow X \rightarrow Suc_y \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

En esta operación se han reemplazado los ejes (Pre_x,x), (x,Suc_x) y (y,Suc_y) por los ejes (Pre_x,Suc_x), (y,x) y (x,Suc_y)

Intercambio. Dos vértices de recorridos distintos son intercambiados simultaneamente. Ejemplo.

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_x \rightarrow X \rightarrow Suc_x \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_y \rightarrow Y \rightarrow Suc_y \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

obtenemos luego de intercambiar los nodos X y Y

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_x \rightarrow Y \rightarrow Suc_x \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Pre_y \rightarrow X \rightarrow Suc_y \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

Esta operación ha reemplazado los ejes (Pre_x,x), (x,Suc_x), (Pre_y,y), (y,Suc_y) por los ejes (Pre_x,y),(y,Suc_x),(Pre_y,x),(x,Suc_y).

Cruza. La última parte de un recorrido se transforma en la última parte del otro recorrido. Ejemplo.

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow X \rightarrow Suc_x \rightarrow m \rightarrow n \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Y \rightarrow Suc_y \rightarrow p \rightarrow q \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

obtenemos luego de cruzar los recorridos a partir de Suc_x y Suc_y

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow X \rightarrow Suc_y \rightarrow p \rightarrow q \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

$$\mathbf{D} \rightarrow \ldots\ldots \rightarrow Y \rightarrow Suc_x \rightarrow m \rightarrow n \rightarrow \ldots\ldots \rightarrow \mathbf{D}$$

Esta operación reemplazó los ejes (x,Suc_x) , (y,Suc_y) por (x,Suc_y), (y,Suc_x)

La operación que implementaremos en nuestro algoritmo es la de *Intercambio.* Para ello proponemos el siguiente algoritmo: tomamos los recorridos de a pares, eligiendo todos los pares posibles. Luego, para cada nodo del primer recorrido, buscamos en el segundo recorrido un nodo para hacer el intercambio tal que la ganancia G obtenida en el costo sea positiva. Si se cumple esta condición, entonces verificamos que los nuevos recorridos cumplan con las restricciones de capacidad de cada camión. Para ello definimos las siguientes reglas:

Dados los recorridos

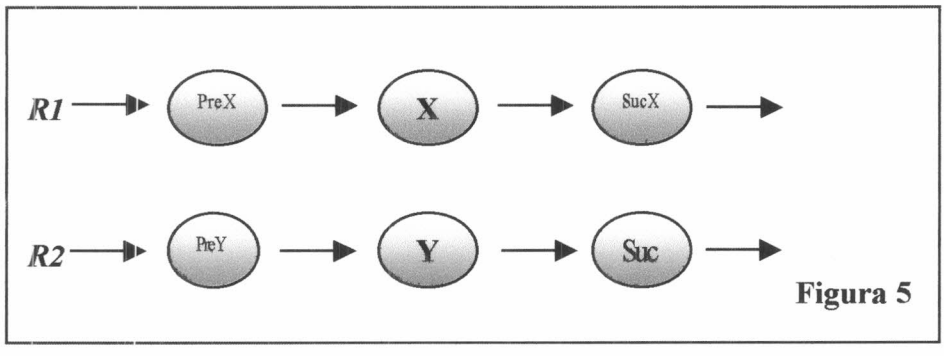


Figura 5

definimos la ganancia de intercambiar *x* e *y* como:

$$\mathbf{G} = d(Pre_x,x) + d(x,Suc_x) - [\, d(Pre_x,y) + d(y,Suc_x) \,] + d(Pre_y,y) + d(y,Suc_y) - [\, d(Pre_y,x) + d(x,Suc_y) \,]$$

La restricción de capacidad de los recorridos se verifica mediante las siguientes condiciones:

$$\text{Demanda}(\mathbf{R1}) - \text{Demanda}(x) + \text{Demanda}(y) > \mathbf{Q1}$$

$$\text{Demanda}(\mathbf{R2}) - \text{Demanda}(y) + \text{Demanda}(x) > \mathbf{Q2}$$

donde *R1* y *R2* son los recorridos entre los que se realizará el intercambio, *Q1* y *Q2* son las capacidades de los camiones asignados a los recorridos R1 y R2 respectivamente, y Demanda(R_i) es la demanda acumulada de cada recorrido *i* antes de realizar el intercambio.

Si **G**>0 y se cumplen las condiciones de capacidad definidas entonces se realizará el Intercambio.

El siguiente cuadro describe el Pseudo-Código del algoritmo que proponemos para implementar la heurística de mejoramiento descripta.

```
HeuristicaLocal( tour )
{       costoNuevo = Costo(tour);
        ciclo = 0;
        do
        {       costoAnterior = costoNuevo;
                for  r1=1 to M-1
                        for r2=(r1+1) to M
                                MultipleExchange(r1,r2);
                costoNuevo = Costo(tour);
                ciclo++;
        } (mientras costoNuevo<costoAnterior and
          (ciclo<MaxCiclosMejora  or  MaxCiclosMejora=0));

        for  r1=1 to M
                SingleExchange(r1);
}
```

```
MultipleExchange(r1,r2)
{       for x=1 to r1.CantidadNodos()
        for  y=1 to r2.CantidadNodos()
                if  VerificarIntercambio(r1,r2,Nodo(x),Nodo(y),Demanda(r1),Demanda(r2))
                    Intercambio(r1,r2, Nodo(x),Nodo(y))
}

SingleExchange(r1)
{       costoNuevo = Costo(r1);
        ciclo=0;
        do
        {       nodoX = Primero(r1);
                costoAnterior = costoNuevo;
                Mientras  no finalice el recorrido
                {       nodoX = nodoX.Siguiente() ; /* La primera vez saltear el depósito*/
                        nodoY = nodoX;
                        hayIntercambio = False;
                        Mientras NO recorrido finalizado  y  NO hayIntercambio
                        {       NodoY = nodoY.Siguiente();
                                if  VerificarIntercambioSimple(nodoX,nodoY)
                                {       nodoIntercambio = nodoX.Anterior();
                                        IntercambioSimple(nodoX,nodoY);
                                        nodoX = nodoInterambio;
                                        hayIntercambio=True;
                                };
                        };
                };
                costoNuevo = Costo(r1);
                ciclo++;
        }       (mientras costoNuevo<costoAnterior and
                (ciclo<MaxCiclosMejora  or  MaxCiclosMejora=0));
};
```

4.4. Implementación

4.4.A. Estructuras de datos a utilizar

En esta sección describiremos la implementación del problema, las principales estructuras de datos y clases utilizadas para representarlo y el pseudo código del algoritmo. El lenguaje elegido fue Microsoft Visual C++ v6.0 y se realizó utilizando un diseño orientado a objetos.

Representamos el problema mediante un grafo completo (clase *Grafo*) al cual le agregamos un nodo como depósito. El *Grafo* contiene una lista de nodos clientes (clase *nodo_Cliente*) y una matriz *m_Pheromone* donde cada entrada *(i,j)* representa el nivel de pheromona del arco que une los nodos *i,j*. El grafo está diseñado para representar instancias del problema *Simétricas* o *Asimétricas*. La matriz *m_Pheromone* es en realidad un vector que almacena las filas de la matriz. Para la variante simétrica la misma se representa como una triangular inferior sin diagonal, almacenada por filas, en cambio para la asimétrica utilizamos una matriz cuadrada. Se diseñó un método para acceder a la entrada en forma correcta dependiendo de la simetría del problema.

Cada *nodo_Cliente* tiene un vector *m_Costo* que representa el costo de los arcos a cada vecino. Otra alternativa de implementación consistiría en utilizar una matriz de costos en el grafo, de la misma manera que se realizó con la matriz de Pheromona. Esta última ahorraría espacio en el caso simétrico, mientras que para el caso asimétrico utilizaría la misma cantidad de memoria que la implementación elegida. Se tomó la decisión de mantener los costos en la clase *nodo_Cliente* porque de esta forma se puede implementar muy fácilmente las listas de candidatos ordenando la lista de vecinos en función de sus distancias, sin necesidad de estructuras auxiliares. Además de esta forma cada cliente tiene toda la información necesaria en su propia clase, sin tener que acceder al grafo. De todas formas, al ser el diseño orientado a objetos, se puede pasar de una representación a la otra fácilmente.

Para indicar que un nodo ha sido asignado a algún recorrido (camión) se agregó en la clase *nodo_Cliente* una propiedad *m_MultiarcosUsados* que indica el número de camiones que pasan por el mismo. Si este valor es 0 entonces el nodo aún no se asignó a ningún recorrido. Esta propiedad es un vector de tamaño *H* (cantidad de hormigas) para poder representar las asignaciones de cada hormiga.

La propiedad *FlujoMaximo* corresponde a la cantidad máxima de camiones que pueden pasar por el nodo, con $0 \leq FlujoMaximo \leq 1$ para los nodos clientes y $0 \leq FlujoMaximo \leq M$ (con *M* cantidad de camiones) para el nodo depósito. Es decir, que *FlujoMaximo* determina el máximo valor posible de *m_MultiarcosUsados*. Con esta representación evitamos tener que realizar búsquedas en una lista de nodos asignados (*Lista Tabú*) para saber si el mismo ya fue incluído en algún recorrido.

El tour solución se representa mediante la clase *Tour* que es un conjunto de nodos de tipo tour (clase *nodo_Tour*).

Las clase *Flota* representa al conjunto de camiones y la clase *Camion* describe las propiedades de cada camión. Si bien en la versión básica de CVRP todos los camiones poseen la misma capacidad, nuestra implementación de la clase *Camion* nos permitirá correr el algoritmo con instancias del problema donde los camiones tengan distintas capacidades. La cantidad de camiones *M* necesaria para satisfacer el problema es calculada en base a la capacidad de los camiones (que es un dato del problema) y la suma total de demanda de los clientes.

A continuación describiremos detalladamente las clases mencionadas, más algunas estructuras auxiliares utilizadas en la implementación del algoritmo. Las mismas están expresadas en C++ que es el lenguaje utilizado en la resolución del problema.

Sean las siguientes variables:

H : número de Hormigas
M : número de Camiones
N : número de Clientes.

La cantidad de nodos del grafo será de *N*+1 (se incluye el nodo depósito).

Representación del Grafo

Las propiedades de la clase **Grafo** son :

Cantidad de nodos.
Cantidad de nodos del grafo, que será N+1, ya que incluye al nodo depósito.

Simetria.
Indica si se está utilizando la variante Simétrica o Asimétrica del problema.

Lista de Clientes.
Lista de punteros a nodos clientes (clase *nodo_Cliente*)

Matriz de Pheromona.
Lista de floats que representa la matriz de pheromona de los arcos.
Cada posición *i,j* de la matriz representa el nivel de pheromona depositado por cada hormiga sobre el arco que une al cliente *i* con su vecino *j*. Inicialmente vale τ_0 (Parámetro de entrada).

```
class Grafo
{private:
        int m_CantNodos;
        char m_Simetria;
        nodo_Cliente* Cliente;
        float* m_Pheromone;
};
```

Representación de los Clientes

La clase **nodo_Cliente** tiene la siguiente estructura:

Identificador
Rótulo del Nodo.

Orden
Es un número interno que representa el desplazamiento del nodo dentro del grafo (desplazamiento dentro de la *Lista de Clientes* del grafo). Se corresponde con los subíndices de la matriz de pheromona, es decir que se utiliza este índice para localizar su entrada en ella.

Tipo de nodo
'D' depósito, 'C' Cliente.

Cantidad de vecinos
Cantidad de vecinos del nodo. Representa la longitud de la lista de vecinos.

Demanda
Demanda de mercadería del nodo.

Vecinos
Lista de punteros a nodo_Cliente de vecinos del nodo. La longitud de la lista está determinada por la propiedad *Cantidad de vecinos.*

Costo
Lista de enteros de longitud "Cantidad de vecinos" . Cada posición *j* de la lista representa el costo del arco que va desde el *nodo_Cliente* al nodo apuntado por *Vecinos(j).*

Flujo Máximo
Representa la máxima cantidad de flujo de camiones que pueden pasar por el nodo, valiendo 1 para los nodos clientes y *M* para el depósito.

Multiarcos Usados
Lista de longitud *H* (cantidad de hormigas) de enteros que indica cuantos multiarcos (camiones asignados) se están usando en el nodo . Si dicho valor es igual a *FlujoMáximo*, el nodo ya completó su capacidad de flujo y no podrá volver a elegirse en la solución *H* actual. Esta propiedad se usa para controlar que cada nodo (salvo el depósito) no se utilice más de una vez en cada solución.

Hormiga
Número de la hormiga que inicia su tour en el nodo. Sirve para representar la ciudad inicial de cada hormiga, ya que en la implementación propuesta no habrá más de una hormiga por ciudad.

```
class nodo_Cliente
{ private:
        int m_Id ;
        int m_Orden;
        char m_TipoNodo;
        int m_CantVecinos;
        int m_Demanda;
        nodo_Cliente** m_Vecinos;
        int* m_Costo;
        int m_FlujoMaximo;
        int* m_MultiarcosUsados;
        int m_Hormiga;
};
```

Representación del Tour solución

La clase Tour representa la solución de una hormiga. Cada solución se compone de *M* recorridos y cada recorrido es una lista de punteros a *nodo_Tour* (los cuales apuntan a los clientes). Cada recorrido *m* representa el trayecto de un camión. Utilizaremos una estructura de datos llamada **tour** que representa las soluciones de las *H* hormigas y que es una lista de longitud *H* de punteros a la clase *Tour* que definiremos a continuación.

Las propiedades de la clase **Tour** son

Cantidad de Recorridos
Cantidad de recorridos del tour, que será igual a la cantidad de camiones.

Cantidad de Clientes
Contador de la cantidad de clientes que posee la solución. Cuando esta cantidad es igual a *N,* significa que se completó la misma.

Costo
Una vez que se completó la solución, en esta propiedad se calcula el costo de la misma.

Primero
Lista de longitud *Cantidad de Recorridos* de punteros al primer nodo de cada recorrido.

Ultimo
Lista de longitud *Cantidad de Recorridos* de punteros al último nodo de cada recorrido.

```
class Tour
{private:
        int m_CantRecorridos;
        int m_CantClientes;
        int m_Costo;
        nodo_Tour** m_Primero;
        nodo_Tour** m_Ultimo;
};
```

la estructura de la clase **nodo_Tour** es la siguiente :

Cliente
De tipo puntero a *nodo_Cliente*, apunta a un nodo cliente del grafo.

Siguiente
De tipo puntero a *nodo_Tour*, apunta al siguiente nodo en el tour (o Null si es el último nodo).

Anterior
De tipo puntero a *nodo_Tour*, apunta al nodo anterior en el tour (o Null si es el primer nodo).

Multiarco
De tipo entero con valor menor o igual a *M* que representa por que multiarco del nodo pasa el recorrido (es decir que camión presta el servicio). Es equivalente al número de recorrido en el cual se encuentra el nodo dentro de la solución.

```
class nodo_Tour
{private:
nodo_Cliente* m_Cliente;
nodo_Tour* m_Siguiente;
nodo_Tour* m_Anterior;
int m_Multiarco
 };
```

Representación de los Camiones

La clase Flota representa al conjunto de Camiones.

Las propiedades de la clase **Flota** son:

Cantidad de Camiones
Representa la cantidad de Camiones de la Flota

Camiones
Lista de longitud *Cantidad de Camiones* de punteros a clase *Camion*.

```
class Flota
{ private:
        Camion* m_Camiones;
        int m_CantCamiones;
};
```

Las propiedades de la clase **Camion** son las siguientes:

Capacidad
Capacidad del Camión.

Estado
Lista de longitud *H* (*H* cantidad de hormigas) que representa los estados por los que va pasando el camión en la solución construída por la hormiga *h*.
Los valores posibles son 0 Inactivo, 1 Activo, 2 Recorrido Finalizado. Cuando se use el método de asignación de camiones consecutivos habrá solo un camión activo a la vez por hormiga. En cambio al utilizar el método de asignación de camiones en paralelo se activan todos los camiones y se inactivan a medida que completan su capacidad.

```
class Camion
{ private:
        int  m_Capacidad;
        int* m_Estado;
};
```

Estructuras adicionales

Vecinos_Permitidos[M,N]
Vector de 2 dimensiones que define, para cada recorrido activo, la lista de posibles vecinos a elegir en cada paso de construcción de la solución. Cada fila representa el recorrido *m* de la solucion que se está procesando, y cada columna representa a los vecinos posibles de visitar en ese recorrido *m*, considerando como nodo actual al último nodo de la solución que se está procesando. Se consideran en el proceso solo los recorridos tal que el estado del camión *m* sea Activo. Para mayor eficiencia en la implementación, se apunta a los vecinos utilizando su desplazamiento en la lista *Vecinos()* del nodo actual. Se inicializa para cada hormiga, en cada iteración del algoritmo.

Probabilidad[M,N]
Vector de 2 dimensiones que almacena las probabilidades calculadas para cada posible vecino (en *Vecinos_Permitidos*) según la regla de transición de estados **(5)** definida en la sección 3.3. Esta estructura es paralela a la anterior.

MejoresVecinos[M]
Vector de enteros con el vecino elegido para cada recorrido según la regla de transición de estados. En caso de utilizar el método de asignación de camiones en paralelo, en el que se avanzan todos los recorridos a la vez, se selecciona el vecino que tenga mayor probabilidad en este vector.

4.4.B. Implementación de Selección Probabilística

Una vez calculados los valores de probabilidad de transición correspondiente a cada vecino aplicando la función de transición, se elige un vecino en forma probabilística. Para ello se implementó un *operador de selección* que simula una ruleta. Se despliegan los

valores de transición de cada vecino sobre un disco con trazos proporcionales a dichos valores. Se arroja una bolilla sobre la ruleta y se elige el vecino sobre el cual cayó la bolilla. Tendrá mayor probabilidad de ser elegido el vecino de mayor valor. Las probabilidades sobre el disco se despliegan ordenadas en forma ascendente. La siguiente figura muestra como quedaría nuestra ruleta, si tenemos 3 vecinos disponibles con probabilidades 0.20, 0.30 y 0.50

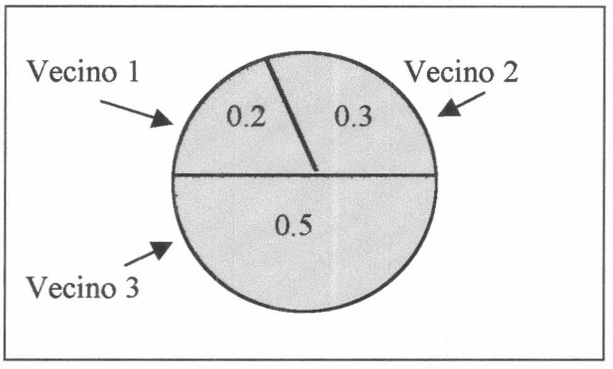


Figura 6. Simulación de selección probabilística.

Para realizar esta simulación se genera un numero random *R* en el rango [0,1] y se suman los valores en el disco (valores de transición de cada vecino) hasta que la suma acumulada alcance a *R*. En ese momento se selecciona el vecino correspondiente al último tramo sumado del disco.

Si la regla de transición utilizada es la *Pseudo Random-Proporcional* se utiliza este operador de selección en el caso de exploración y se elige el vecino correspondiente al mayor valor en caso de explotación.

4.4.C. Implementación de las Listas de Candidatos

Para ello no se agregaron estructuras adicionales, sino que se utilizó la misma lista de punteros a vecinos. En este caso, y como primer paso del proceso, se ordena la lista de vecinos y de costos de cada nodo en función al valor del costo en orden creciente. No se ordena la lista entera, sólo se ordenan *l* nodos, siendo *l* la longitud de la lista de candidatos. Dicha longitud se ingresa como parámetro y como un porcentaje de la cantidad de nodos del problema.

En cada paso del algoritmo cada hormiga, ubicada en un nodo cualquiera, debe armar una lista de vecinos posibles, donde la búsqueda se realiza en todo el vecindario. En el caso de utilizar listas de candidatos la búsqueda se realizará primero en dicha lista, solo si no se encuentra ningún vecino que satisfaga las restricciones entonces se busca en el vecindario entero.

4.4.D. Pseudo Código del Algoritmo

El algoritmo comienza determinando cual es la mínima cantidad de camiones necesarias para resolver el problema e inicializando los arcos con el valor de inicial de pheromona τ_0. Luego, se ejecuta el proceso de generación de soluciones durante una determinada cantidad de ciclos (parámetro *MaxCiclos*) o hasta que se cumpla la condición

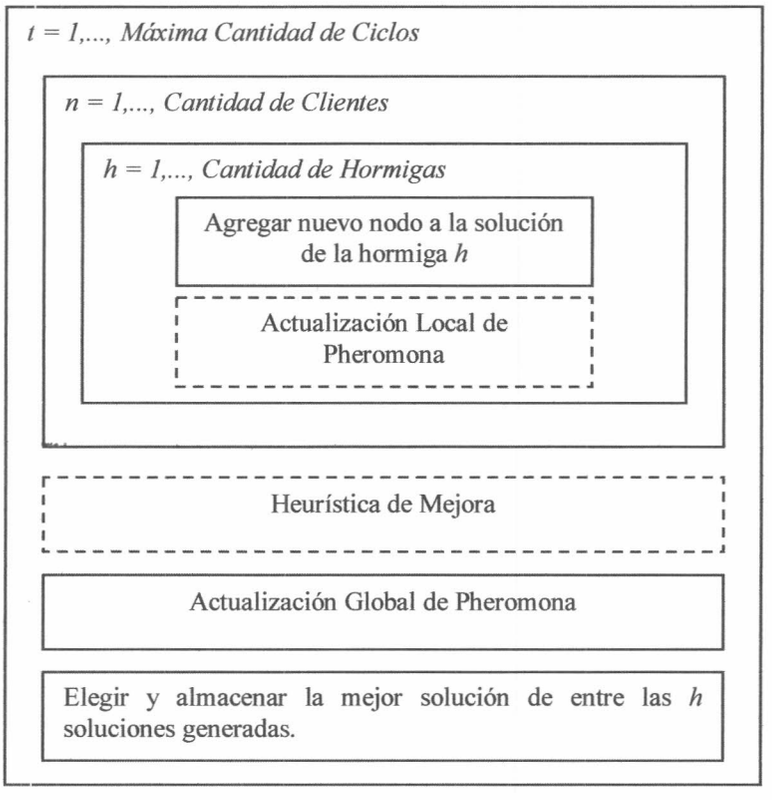
de finalización, la cual consiste en que no se obtenga mejora durante una cierta cantidad de ciclos (también parámetro de entrada). El proceso de generación de soluciones se divide en tres pasos principales:

Paso 1
Inicializar la solución de cada hormiga y la flota de camiones según la cantidad de camiones mínima calculada. Determinar la ciudad inicial de cada hormiga para el caso de asignación de camiones consecutivos.

Paso 2
Cada hormiga construye su solución, obteniendo el siguiente vecino probabilísticamente según la regla de transición de estados elegida, basándose en la cantidad de pheromona de los arcos y el costo de los mismos. En caso de utilizar listas de candidatos los vecinos se seleccionan primero de esta lista, sino se realiza la búsqueda en todo el vecindario. En caso de utilizar actualización local de arcos, luego de elegir el siguiente vecino se actualiza la cantidad de pheromona del arco agregado.

Paso 3
Luego de que todas las hormigas hayan obtenido una solución, y en caso de utilizar heurística de mejora, se mejoran todas las soluciones generadas, almacenándose en *lBest* la mejor solución obtenida. En este paso se procede a realizar la actualización global de arcos según el tipo de actualización global elegido.

A continuación se muestra un esquema del algoritmo. Las distintas variantes implementadas, que no corresponden al algoritmo básico, se indican con línea punteada.



4.5. Implementación de las Restricciones del Modelo

En la sección 2.2 definimos el modelo matemático del problema y las restricciones del mismo. En esta sección explicaremos como se incluyeron las mismas en el marco de nuestro algoritmo.

Las restricciones (1) y (2) del modelo que exigen que la cantidad de arcos entrantes y salientes a cada nodo utilizados en cada solución sea exactamente uno se satisfacen al exigir que cada nodo sea asignado solamente una vez a alguna solución. Cuando cada hormiga debe seleccionar el siguiente vecino para agregarlo a su lista de vecinos permitidos, solo se elige entre aquellos que cumplan la siguiente condición:

(nodo.MultiarcosUsados() < nodo.FlujoMaximo())

donde *FlujoMaximo()* vale 1 para todo nodo cliente y *MultiarcosUsados()* se inicializa en 0 para cada nodo y se incrementa en 1 cada vez que un nodo es asignado a alguna solución.

Las restricciones (3) y (4) aseguran que los arcos entrantes y salientes al depósito respectivamente sean exactamente igual a la cantidad de camiones. Para ello en el nodo depósito el valor de *FlujoMaximo*() es igual a K (K cantidad de camiones). En realidad, esta condición se da por construcción de los recorridos, ya que no se permite elegir nodo depósito mientras haya aún nodos clientes no elegidos que no violen las condiciones de capacidad del camión.

La restricción (5) exige que la cantidad de mercaderías entregadas en un viaje no supera la capacidad del camión, para todo camión k. Esta condición se cumple al no permitir asignar un nodo a algún recorrido si la demanda del mismo excede la capacidad disponible del camión. Nuevamente, cuando cada hormiga debe seleccionar el siguiente vecino para agregarlo a su lista de vecinos permitidos, se verifica además que el nodo elegido cumpla la siguiente condición:

((nodo.Demanda()+demandaAcumulada) <= capacidad)

donde *demandaAcumulada* representa la demanda de mercadería acumulada del recorrido que se está construyendo y *capacidad* representa la capacidad del camión correspondiente a dicho recorrido.

La restricción (6) garantiza que solamente un camión visite a cada nodo i, esta condición se da por construcción de los recorridos al no permitir que un nodo sea asignado a más de un recorrido (al aplicar la condición descripta para las restricciones (1) y (2)). Igualmente para la restricción (7) que exige que exactamente K camiones visiten el depósito (al aplicar la condición descripta para (3) y (4)).

La restricción (8) que exige que si existe un arco que une dos nodos cualesquiera (excepto el depósito), entonces esos nodos deben ser visitados por el mismo camión k también se cumple por construcción de los recorridos, ya que cada nodo asignado al tour se asigna al recorrido (camión) actual, y cuando el mismo agota su capacidad, se elige otro camión como actual.

La restricción (9) que es una regla de eliminación de subtours, se garantiza exigiendo que toda solución válida tenga exactamente n nodos (n cantidad de clientes), y además por construcción todo nodo asignado a una solución está conectado al nodo anterior en dicha solución.

4.6. Complejidad del algoritmo

Siendo h cantidad de hormigas, m cantidad de camiones, n cantidad de nodos clientes y NC número de ciclos a ejecutar, calcularemos la complejidad del algoritmo en las distintas etapas.

Etapa de Inicialización de Camiones.
Esta etapa corresponde a la Activación e Inactivación de los camiones correspondientes según el método de construcción elegido, donde cada hormiga debe inicializar su flota de camiones. El orden de la misma es:

$$O(m \cdot h \cdot NC)$$

Etapa de Inicialización de Tours.
Se inicializa un tour solución por hormiga, agregando a cada recorrido de cada solución el nodo depósito. Si el método de construcción elegido es el de asignación de camiones consecutivos, a cada tour solución se le agrega la ciudad inicial correspondiente a la hormiga, debiendo recorrer por cada hormiga los n nodos (a lo sumo) hasta encontrar la ciudad correspondiente. Por lo tanto el orden de esta etapa es:

Método 1 : $O(m \cdot h \cdot NC) + O(h \cdot n \cdot NC)$ **(1)**

Método 2 : $O(m \cdot h \cdot NC)$ **(2)**

Dado que en nuestra implementación, definimos una hormiga por cliente nos queda en

$$O(m \cdot h \cdot NC) + O(h^2 \cdot NC)$$

Etapa de Construcción.
Se elige del vecindario (de tamaño n-1) el siguiente nodo a agregar al tour. Para ello se calcula la probabilidad de cada vecino según la función de transición. Esta operación es realizada por cada hormiga, y por cada cliente del grafo hasta completar todos los tours. Debemos diferenciar según el método de construcción que estemos utilizando. En caso de estar utilizando el método de construcción de asignación de camiones en paralelo, dicha operación se realiza por cada camión, en cambio con el método de asignación de camiones consecutivos sólo se realiza para el camión activo.
Si bien en cada paso de construcción, el tamaño del vecindario que cumple con las restricciones del problema se va decrementando, como en nuestra implementación debemos recorrer todo el vecindario para seleccionar los que son factibles, consideraremos el tamaño del vecindario de (n-1) para cada paso.

Método 1 : $O(n \cdot h \cdot (n\text{-}1) \cdot NC)$

Método 2 : $O(n \cdot h \cdot m \cdot (n\text{-}1) \cdot NC)$

Etapa de Actualización de Pheromona.
Supongamos para el cálculo el peor caso, que es cuando se utilizan todas las soluciones encontradas para actualizar el nivel de pheromona de los arcos. Siendo la cantidad de arcos de un tour solución n+1 (considerando los arcos a los 2 depósitos) la cantidad de operaciones es:

$$O\ (h \cdot (n+1)\ )$$

5. DESCRIPCION DE LA INTERFASE

Se implementó una interfase muy simple que permite ingresar los parámetros de configuración del problema. Los datos del problema se ingresan mediante un archivo de texto que responde al formato propuesto en las librerías de ejemplos TSPLIB. Las mismas se pueden encontrar en: http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95 y en http://www.branchandcut.org/VRP/data.

A continuación mencionaremos los parámetros de configuración.

Versión del problema.

Según se explicó en la sección 4.4.A. el algoritmo está diseñado para resolver tanto instancias simétricas como asimétricas del problema. La interfase permite ingresar que versión del problema se desea resolver siendo los valores posibles:

1. Simétrico
2. Asimétrico

Método de asignación de camiones.

Este parámetro permite seleccionar el método de asignación de camiones a utilizar según los métodos que propusimos en la sección 4.2., siendo los valores posibles:

1. Camiones Consecutivos
2. Camiones en Paralelo

Tipo de actualización global.

El mismo define el tipo de actualización global de arcos que se utilizará según se explicó en la sección 4.3.A. Los posibles valores son:

1. Mejor solución
2. Hormigas Elitistas
3. Todas las soluciones

Si el tipo de actualización global elegida es *Hormigas Elitistas* se solicita al operador ingresar la cantidad de hormigas elitistas a utilizar.

Usa actualización local.

Mediante este parámetro se habilita o deshabilita la actualización local de pheromona de arcos, según se explicó en la sección 4.3.B. En caso de utilizar actualización local, la interfase solicita además los siguientes parámetros:

Tipo de actualización local : siendo los valores posibles:

1. Pheromona inicial
2. Proporcional a la distancia
3. Evaporación
4. Q-Learning

En caso de seleccionar Q-Learning, se solicita el parámetro de ajuste γ.

Factor de persistencia de pheromona para actualización local : Corresponde al parámetro φ de la regla **(6)** explicada en la sección 4.3.B.

Tipo de regla de transición.

Este parámetro permite seleccionar la regla de transición de estados a utilizar siendo los posibles valores:

1. Regla Random-Proporcional
2. Regla Pseudo-Random-Proporcional

En caso de utilizar la regla Pseudo-Random Proporcional la interfase solicita además el parámetro q_0 que determina la importancia de explotación versus exploración.

Valor inicial de pheromona en los arcos

Corresponde al parámetro τ_0 de nivel inicial de pheromona de los arcos. Es el mismo valor que se utiliza para el tipo de actualización local *Pheromona Inicial*.

Factor de persistencia de pheromona

Corresponde al parámetro φ de factor de persistencia de pheromona utilizado en la fórmula de actualización global de arcos.

Parámetros de ajuste para la regla de transición

La interfase solicita los siguientes parámetros que definen la regla de transición de estados:

- Importancia relativa de pheromona (Parámetro α).
- Importancia relativa de la distancia (Parámetro β).

Utiliza heurística local (de Mejoramiento)

Este parámetro habilita o no el uso de heurística de mejora. En este caso solicita ingresar la máxima cantidad de ciclos de mejora a desarrollar sobre cada solución. En caso de ingresar 0 el proceso se ejecuta hasta que ya no se produzcan mejoras sobre la solución.

Utiliza listas de candidatos.

El mismo habilita el uso de listas de candidatos según se describió en la sección 4.3.D. En este caso se solicita el tamaño de las listas. El mismo se ingresa como porcentaje de la cantidad de nodos del problema.

Cantidad de ciclos a ejecutar.

Indica la cantidad de ciclos que se repetirá el proceso de construcción de tours.

Cantidad de ciclos de condición de finalización.

Si luego de la cantidad de ciclos indicada por este parámetro no se obtiene mejora en el costo, se finaliza el proceso. Si el valor de este parámetro es 0, el proceso finaliza al alcanzar la cantidad de ciclos indicada por el parámetro descripto anteriormente.

Elige cantidad de hormigas.

En caso afirmativo, se solicita al usuario la cantidad de hormigas a utilizar, sino se

define una hormiga por cliente.

La selección de los parámetros óptimos se realizó mediante experimentación, la siguiente sección describe los mismos.

6. RESULTADOS COMPUTACIONALES

En esta sección describiremos los resultados obtenidos, y los tipos de experimentos que se realizaron. Los problemas utilizados para testear correponden a las librerías TSPLIB y se pueden encontrar en:

http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95 y en
http://www.branchandcut.org/VRP/data.

Se realizaron experimentos sobre dos conjuntos de problemas. El primer conjunto tuvo como finalidad encontrar los parámetros óptimos para cada situación y la configuración que genere los mejores resultados. Se consideraron nueve problemas de distintos tamaños, calculando el Error Relativo Promedio del conjunto para cada configuración posible. La cantidad de hormigas fue definida en todos los casos igual a la cantidad de clientes y todos los testeos se realizaron sobre 100 ciclos. Para cada configuración se consideraron los dos métodos de asignación de camiones: *Camiones en Paralelo* y *Camiones Consecutivos* produciendo cada uno de ellos resultados distintos, como ya describiremos.

El segundo conjunto de problemas tiene como finalidad testear el comportamiento del algoritmo, utilizando la mejor configuración del mismo encontrada en el paso anterior, y comparar resultados con otras heurísticas y metaheurísticas. Para ello se utilizó un conjunto de problemas más amplio, agregándose problemas de mayor cantidad de nodos. En esta segunda parte de los experimentos, se hicieron algunas pruebas utilizando menor cantidad de hormigas.

Los problemas utilizados para la primera y segunda parte de los experimentos se encuentran en las Tablas I (a) y (b) respectivamente. Las mismas describen para cada problema la cantidad de clientes, cantidad mínima de camiones necesarios, la cota superior conocida del mismo y el *gap* correspondiente a la cota inferior. En los casos en que el gap es 0 significa que el valor correspondiente a *Cota Superior* es el óptimo. Los casos en que no se conoce la cota inferior los indicaremos con un signo de interrogación (?) en la columna *Gap*. El esquema de nombres utilizado para los problemas de testeo describe sus características, donde la primer letra corresponde al autor, la *n* se antepone a la cantidad de nodos del problema y la *k* a la cantidad de camiones.

Las letras correspondientes al autor significan:

A. Augerat. Conjunto A. [1]

B. Augerat. Conjunto B.

E. Christofides y Eilon. [7]

F. Fisher. [14]

M. Christofides, Mingozzi y Toth. [9]

Problema	#Nodos	#Camiones	Cota Superior	Gap
A-n32-k5	32	5	784	0
A-n33-k5	33	5	661	0
A-n65-k9	65	9	1174	0
A-n80-k10	80	10	1764	3.62
B-n31-k5	31	5	672	0
E-n13-k4	13	4	247	0
E-n22-k4	22	4	375	0
E-n76-k7	76	7	682	0
E-n101-k14	101	14	1077	?

Tabla I (a) . Primer set de problemas de testeo.

Problema	#Nodos	#Camiones	Cota Superior	Gap
E-n101-k8	101	8	815	?
F-n135-k7	135	7	1162	0
M-n101-k10	101	10	820	0
M-n151-k12	151	12	1028.42	?

Tabla I (b). Segundo set de problemas de testeo.

Como ya comentamos, los parámetros óptimos fueron encontrados por experimentación. Una idea muy interesante para determinar los mismos en una forma más sistemática fue propuesta por Botee y Bonabeau. Ellos proponen usar un *Algoritmo Genético (GA)* para evolucionar los parámetros de Colonia de Hormigas, donde cada individuo está caracterizado por un conjunto de parámetros y donde la función de aptitud está definida para medir que tan bien se comporta el algoritmo con dichos parámetros, y con que rapidez converge a una solución. La aptitud de cada individuo es evaluada ejecutando Colonia de Hormigas. Para una descripción más detallada ver [5].

6.1. Determinación de los parámetros de importancia de Pheromona y Visibilidad.

El primer experimento realizado consistió en la comparación de distintos pares de valores para obtener el mejor valor de los parámetros de importancia relativa de pheromona α y visibilidad β, para ambos métodos de asignación de camiones, consecutivos y en paralelo.

Los pares de valores testeados se muestran en la Tabla II a continuación :

Caso	α	β
1	0	1
2	0,5	1
3	1	0
4	1	0,5
5	1	1
6	1	2
7	1	5
8	2	1
9	2	2
10	2	3
11	2	5
12	2	8
13	3	2
14	5	2
15	5	5
16	8	2

Tabla II. Parámetros considerados para los valores de importancia relativa de Pheromona (α) y Visibilidad (β).

A continuación describiremos los resultados obtenidos para los distintos métodos de asignación de camiones.

A. Método de asignación de camiones consecutivos

La Tabla III muestra, para cada caso considerado de α y β, el Error Relativo Promedio (E.R.P.) obtenido por el conjunto de problemas, el cual surge de comparar cada solución alcanzada contra la cota superior de cada problema (informada en la Tabla I), ordenados en forma ascendente por dicho valor de error.

α	β	E.R.P.
2	5	0,147524792
1	5	0,167071675
2	8	0,16927809
2	3	0,172029548
5	5	0,173238052
8	2	0,200853668
2	2	0,205663525
1	2	0,213311395
5	2	0,223539434
3	2	0,248367294
2	1	0,334887358
1	1	0,367041685
0,5	1	0,579094065
1	0,5	0,724834852
0	1	0,770858651
1	0	1,096239531

Tabla III. E.R.P. obtenidos para los distintos valores de α y β. Método de asignación de camiones consecutivos .

El gráfico de la Figura 7 presentado a continuación muestra los resultados expresados en la Tabla III, donde el tamaño de cada burbuja es inversamente proporcional al E.R.P. Para poder comparar ambos métodos de asignación se dividieron los resultados en tres zonas de acuerdo al E.R.P. obtenido, donde cada zona está representada en el gráfico con un color diferente. La primer zona está dada por los valores de error menores de 0.20, la segunda corresponde a los errores entre 0.20 y 0.50, y la tercer zona por los errores mayores a 0.50.

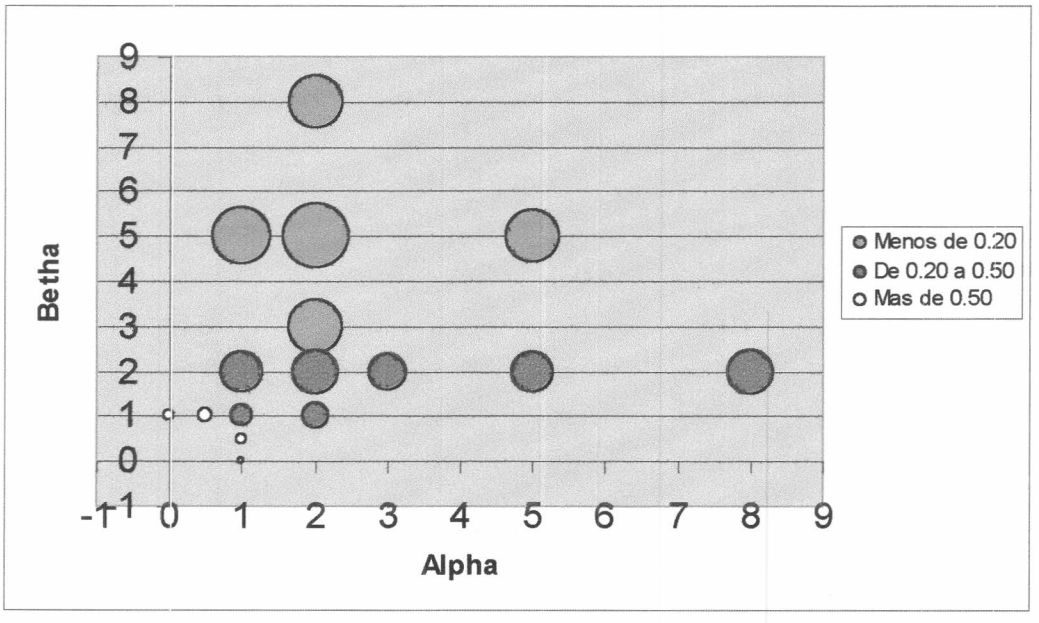


Figura 7. E.R.P. obtenido para los distintos valores de α y β. Método de asignación de camiones consecutivos.

Como podemos observar, los peores resultados se obtuvieron con $\alpha = 0$ o $\beta = 0$. El primero representa el caso en que las hormigas no tienen en cuenta el valor de pheromona de los arcos, guiándose solamente por la distancia de los arcos. El resultado de este experimento demuestra la importancia de la cooperación entre hormigas mediante el intercambio de información a través de pheromona. El segundo experimento representa el caso en que no se considera la distancia de los arcos.

Los mejores resultados se obtuvieron en general en los casos en que la importancia relativa de la distancia (β) es mayor a la importancia relativa de pheromona (α).

B. Método de asignación de camiones en paralelo

Los resultados obtenidos por el método de asignación de camiones en paralelo se detallan en la Tabla IV y en el gráfico de la Figura 8.

α	β	E.R.P.
1	1	0,157426485
1	2	0,163750149
3	2	0,16559894
2	2	0,165608305
0,5	1	0,167546469
1	5	0,170392854
2	3	0,173160318
2	5	0,173204484
5	2	0,177070876
5	5	0,187031471
8	2	0,191659592
2	1	0,205717879
2	8	0,217299788
0	1	0,23883697
1	0,5	0,24108561
1	0	1,200240531

Tabla IV. E.R.P. obtenidos para los distintos valores de α y β. Método de asignación de camiones en paralelo .

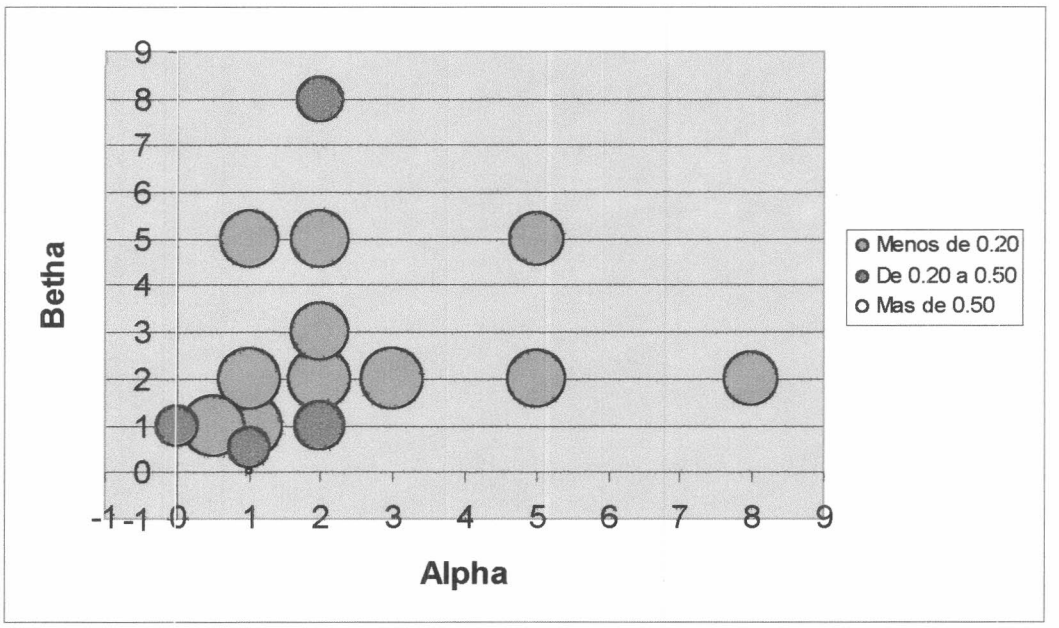


Figura 8. E.R.P. obtenido para los distintos valores de α y β. Método de asignación de camiones en paralelo.

Al igual que para el método de asignación de camiones consecutivos, los casos con $\alpha = 0$ o $\beta = 0$ están dentro del grupo de los peores resultados. Sin embargo, el caso $\alpha = 0$ produjo un error bastante aceptable con respecto al método de asignación consecutivo.

Además vemos que la variación de los errores obtenidos para cada caso por este método es menor que en el método consecutivo, produciéndose en general mejores resultados (obsérvese que la cantidad de burbujas azules es mayor que en el caso anterior).

Los resultados obtenidos responden a lo que se esperaría intuitivamente, ya que el método de asignación en paralelo considera el espacio de arcos en forma más global que el método consecutivo. En el momento de elegir el siguiente vecino (o arco de salida) a incluir en el recorrido, el método de asignación de camiones consecutivos solo considera los arcos de salida correspondientes al último nodo del recorrido actual, en cambio en el método de asignación de camiones en paralelo se consideran los arcos de salidas de cada nodo correspondiente a cada recorrido.

En los experimentos sucesivos utilizaremos los mejores pares obtenidos para cada método; $\alpha=2$, $\beta=5$ para el método de asignación consecutivo y $\alpha=1$, $\beta=1$ para el método de asignación paralelo.

6.2. Actualización global de arcos

Las pruebas que describiremos en esta sección corresponden a la etapa de actualización global de pheromona en los arcos, realizada luego de que cada hormiga obtuvo una solución. Según fue explicado en la sección 4.3. los tipos de actualización global considerados son: *todas las soluciones*, la *mejor solución* y *soluciones elitistas*. La cantidad de hormigas elites es un dato de entrada que se ingresa como porcentaje de la cantidad de nodos del problema. Los porcentajes considerados fueron 10 % y 50 %.

Para ambos métodos de asignación de camiones el mejor resultado, tomando el E.R.P. de los casos de prueba, se obtuvo utilizando *mejor solución* mientras que el peor fue para el tipo de actualización *todas las soluciones.* Sin embargo los resultados obtenidos en todos los casos son muy cercanos.

Los resultados para ambos métodos de asignación de camiones son presentados en los gráficos de las Figuras 9 y 10. Los mismos presentan los errores relativos obtenidos por cada tipo de actualización global para cada uno de los problemas de testeo. Las Tablas V y VI muestran un resumen del gráfico, promediando los errores relativos obtenidos.

A. Método de asignación de camiones consecutivos

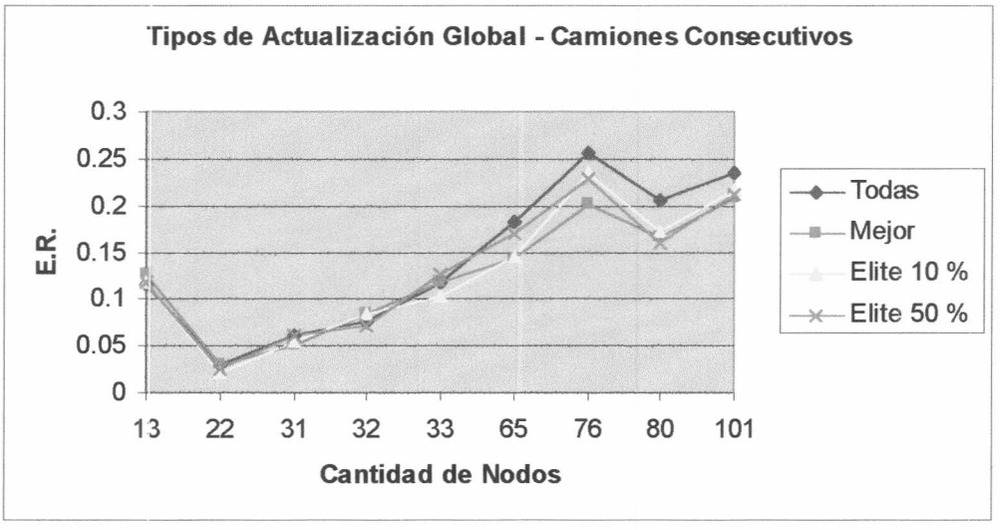


Figura 9. Actualización global de pheromona. Método de asignación de camiones consecutivos.

Tipo de actualización Global	E.R.P.
Mejor	0.12491108
Elite 10%	0.12757862
Elite 50 %	0.13023113
Todas	0.14201301

Tabla V. Actualización global de pheromona. Método de asignación de camiones consecutivos.

B. Método de asignación de camiones en paralelo

Como podemos observar en el gráfico, los errores relativos obtenidos con este método de asignación según el tipo de actualización global se diferencian más entre sí que para el método de asignación de camiones consecutivos, especialmente para el tipo de actualización *mejor solución* con respecto a los demás tipos.

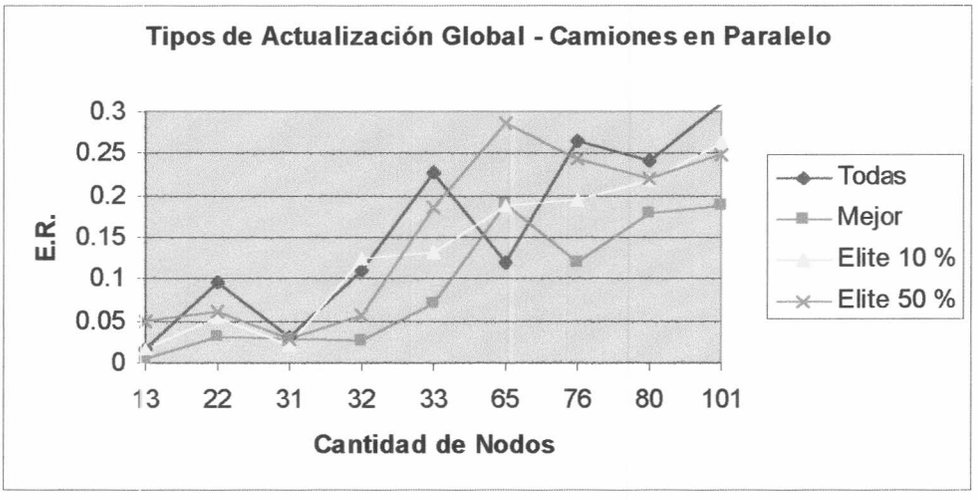


Figura 10. Actualización global de pheromona. Método de asignación de camiones en paralelo.

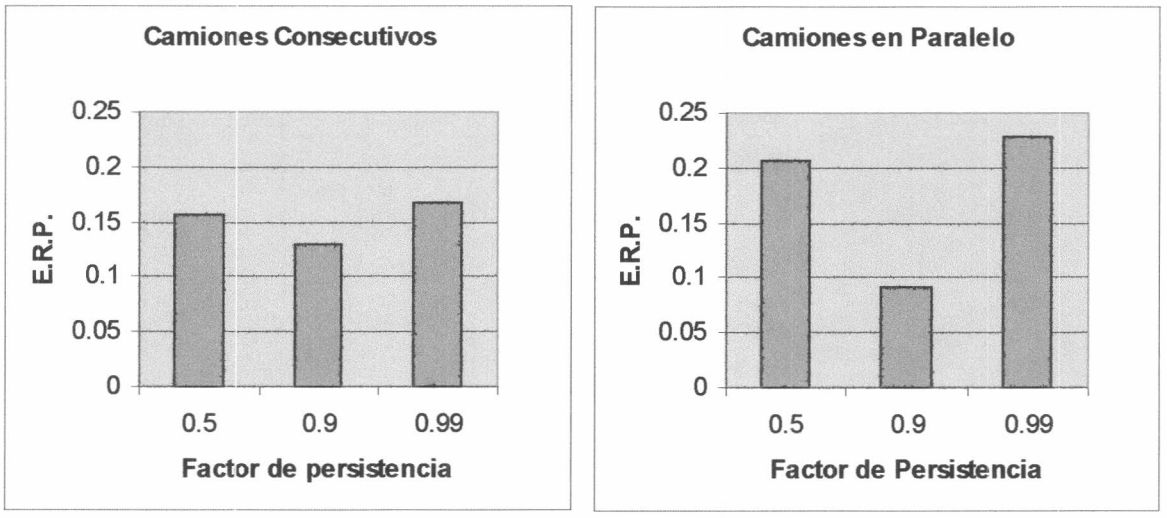
Tipo de actualización global	E.R.P.
Mejor	0.09240768
Elite 10 %	0.13421333
Elite 50 %	0.15316412
Todas	0.15742649

Tabla VI. Actualización global de pheromona. Método de asignación de camiones en paralelo.

6.3. Factor de persistencia de pheromona

Como ya fue descripto en la sección 3.2, el factor de persistencia de pheromona φ, indica que cantidad de pheromona se mantiene en los arcos en cada ciclo de ejecución del

algoritmo, tal que (1- φ) representa la evaporación del arco. Se experimentaron con tres valores distintos para φ : 0.50, 0.90 y 0.99. El gráfico de la Figura 11 y la Tabla VII muestran los resultados obtenidos. En los mismos se detalla el error relativo promedio obtenido sobre los problemas de testeo para cada valor de persistencia.



(a) (b)

Figura 11. Factor de persistencia de pheromona. Gráfico comparativo.
(a) Camiones consecutivos. (b) Camiones en paralelo.

Camiones Consecutivos	
φ	E.R.P.
0.5	0.157203686
0.9	0.130231324
0.99	0.166502219

Camiones en Paralelo	
φ	E.R.P.
0.5	0.206408647
0.9	0.092407683
0.99	0.227621374

Tabla VII. Comparativo valores de persistencia de pheromona. Error relativo promedio.

6.4. Tipos de regla de transición

En esta sección se comparan los resultados obtenidos de aplicar las distintas variantes de regla de transición de estados: regla *random–proporcional* y la variante *pseudo-random-proporcional* propuesta en la sección 4.3.C. En la aplicación de esta última se consideraron los valores $q_0 = 0.5$ y $q_0=0.9$.

Los mejores resultados se obtuvieron con la aplicación de la regla random-proporcional y pseudo-random-proporcional con $q_0=0.5$, obteniéndose errores relativos promedios muy similares. La aplicación de la variante pseudo-random-proporcional con $q_0=0.9$ produjo el peor resultado, y esto se debe a que al elegir siempre (o mejor dicho un 90 % de las veces) el mejor valor de transición calculado, no se favorece la exploración de nuevos arcos.

En los gráficos de las Figuras 12 y 13 mostramos los errores relativos obtenidos con cada problema de testeo para las variantes de reglas de transición aplicadas. Las Tablas

VIII y IX presentan los datos resumidos, mostrando el error relativo promedio del conjunto testeado.

A. Método de asignación de camiones consecutivos

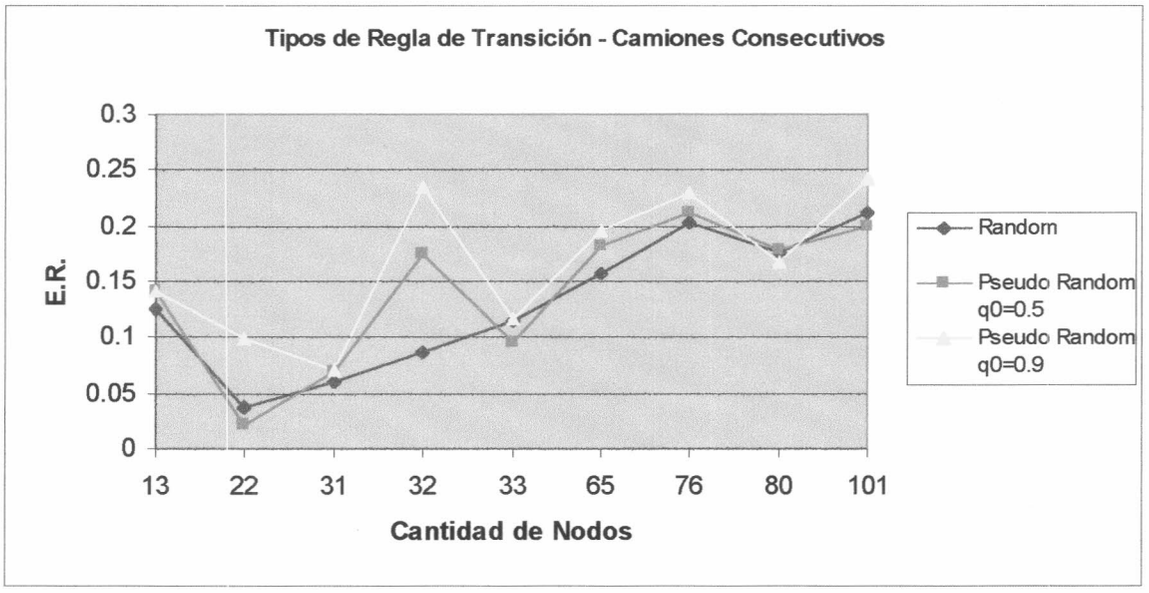


Figura 12. Error relativo obtenido con las distintas variantes de regla transición de estados. Camiones consecutivos.

Tipo de regla de transición	E.R.P.
Random-proporcional	0.13023132
Pseudo-random-proporcional $q_0 = 0.5$	0.141625994
Pseudo-random-proporcional $q_0 = 0.9$	0.166315765

Tabla VIII. Tipos de regla de transición de estado. Error relativo promedio obtenido. Camiones consecutivos.

B. Método de asignación de camiones en paralelo

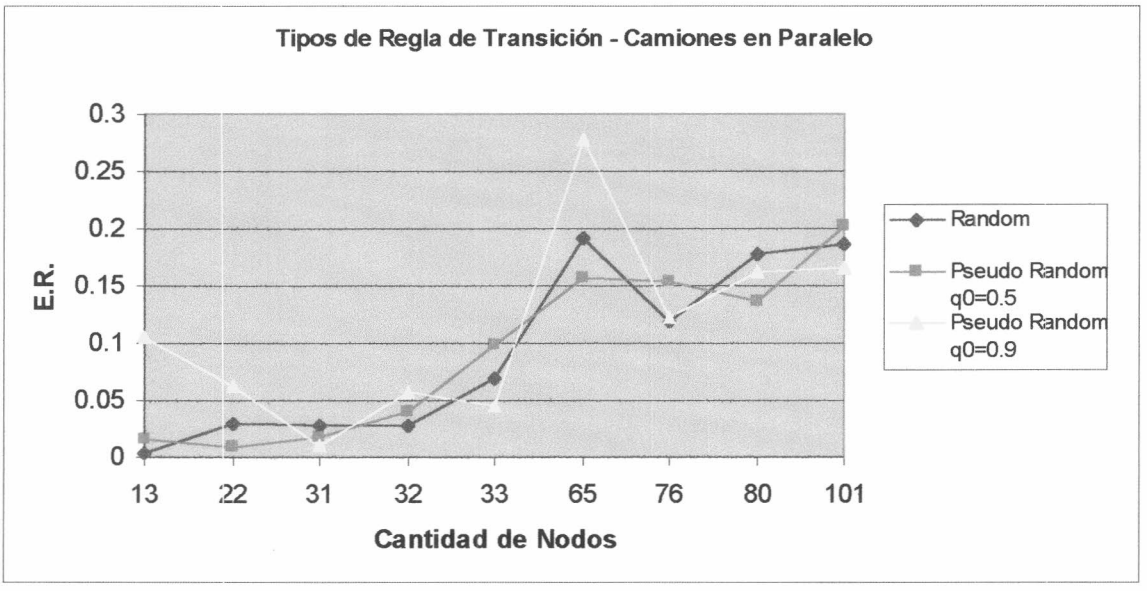


Figura 13. Error relativo obtenido con las distintas variantes de regla transición de estados. Camiones en paralelo.

Tipo de regla de transición	E.R.P.
Pseudo-random proporcional $q_0 = 0.5$	0.09205754
Random	0.09240768
Pseudo-random proporcional $q_0 = 0.9$	0.11190591

Tabla IX. Tipos de regla de transición de estado. Error relativo promedio obtenido. Camiones en paralelo.

6.5. Actualización local de arcos

En esta sección comparamos los resultados obtenidos con la aplicación de actualización local de arcos. Los casos considerados fueron: *sin actualización local*, *pheromona inicial*, *Q-learning*, *proporcional a la distancia* y *evaporación* según fue explicado en la sección 4.3.B. El factor de persistencia utilizado en la regla de actualización local fue el mismo que para la actualización global de arcos, utilizando el propuesto en la sección 6.3, $\varphi = 0.9$. El valor inicial de pheromona de los arcos fue el mismo que en las pruebas anteriores siendo $\tau_0 = 0.5$.

La aplicación de actualización local en el método de asignación de camiones en paralelo no produjo buenos resultados, mejorando los mismos para el método de asignación de camiones consecutivos. En ambos casos el mejor resultado se obtuvo sin la aplicación de actualización local de arcos, y los peores resultados fueron para los tipos de actualización *proporcional a la distancia* y *evaporación.*

Los gráficos de las Figuras 14 y 15 y las Tablas X y XI comparan los errores relativos promedios obtenidos con la aplicación de cada tipo de actualización local para ambos métodos de asignación de camiones.

A. Método de asignación de camiones consecutivos

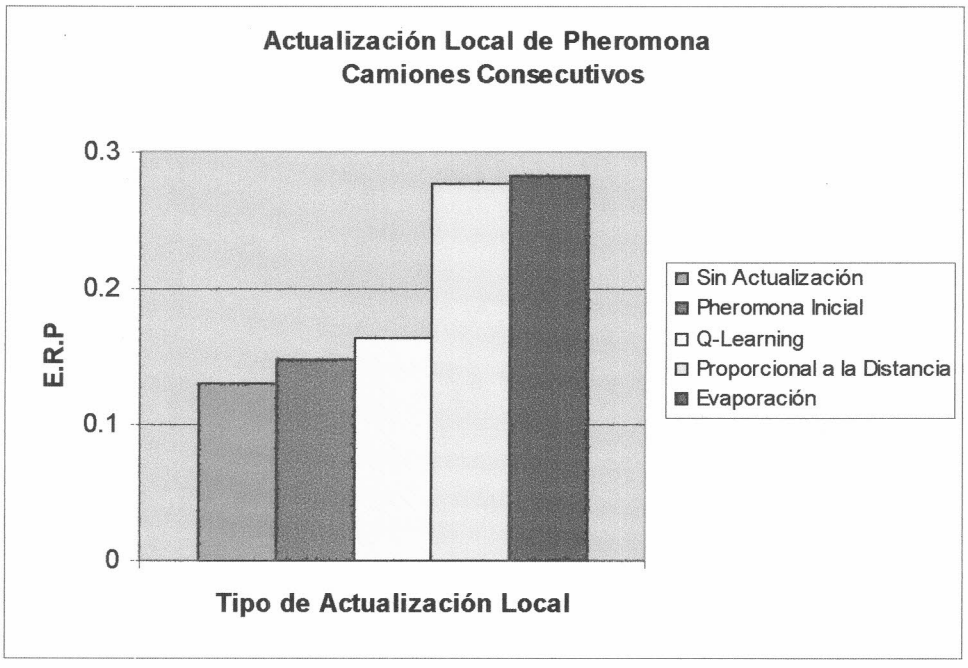


Figura 14. Comparativo de reglas de actualización local de arcos. Error relativo promedio obtenido. Camiones consecutivos.

Tipo de actualización local	E.R.P.
Sin actualización	0.130231324
Pheromona inicial	0.14715596
Q-Learning	0.163144763
Proporcional a la distancia	0.276650546
Evaporación	0.282914012

Tabla X. Tipos de actualización local de arcos. Error relativo promedio. Camiones consecutivos.

B. Método de asignación de camiones en paralelo

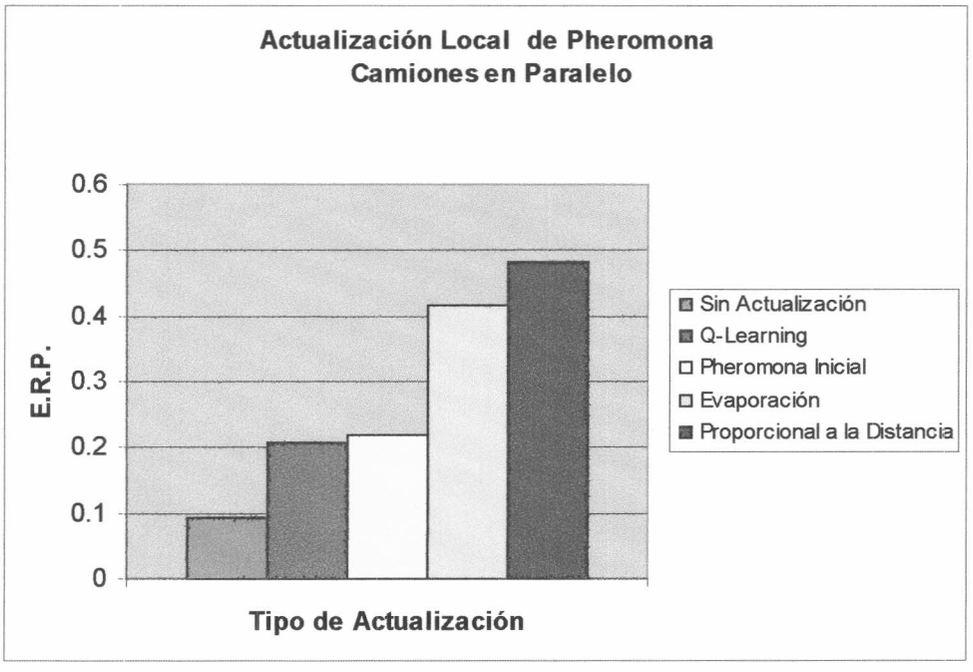


Figura 15. Comparativo de reglas de actualización local de arcos. Error relativo promedio obtenido. Camiones en paralelo.

Tipo de actualización local	E.R.P.
Sin Actualización	0.092407683
Q-Learning	0.20724768
Pheromona inicial	0.217657008
Evaporación	0.416103905
Proporcional a la distancia	0.4835696

Tabla XI. Tipos de actualización local de arcos. Error relativo promedio. Camiones en paralelo.

En la sección 4.3.B. mencionamos que la finalidad de aplicar actualización local es hacer que los arcos recientemente visitados sean menos deseables por las siguientes hormigas, al disminuir el nivel de pheromona de los mismos y lograr como consecuencia mayor diversidad en las soluciones encontradas. Esto se cumple, dependiendo de los valores elegidos para φ y del valor inicial de pheromona de los arcos τ_0. Se realizaron distintas pruebas para verificar dicho comportamiento, descubriéndose que tomando como factor de persistencia local φ=0.99 y valor inicial de pheromona de los arcos $\tau_0 = 0.05$ la aplicación de la regla de actualización local disminuye el nivel de pheromona, no siendo así para los valores que veníamos usando de φ=0.90 y $\tau_0 = 0.50$. A pesar de ello, con esta última combinación de valores obtuvimos en general mejores resultados, tanto para

actualización local como global, por lo tanto utilizaremos dichos valores en el resto de los experimentos. En la Tabla XII (a) y (b) exponemos los resultados. Los resultados de actualización local corresponden al tipo *Pheromona Inicial.*

Valores Considerados	E.R.P
φ=0.90 y τ_0 = 0.50. Actualización Local.	0.14715596
φ=0.99 y τ_0 = 0.05. Actualización Local.	0.158727538
φ=0.90 y τ_0 = 0.50. Actualización Global.	0.130231324
φ=0.99 y τ_0 = 0.05. Actualización Global.	0.150038742

Tabla XII (a). Parámetros considerados para el método de asignación de camiones consecutivos.

Valores Considerados	E.R.P
φ=0.90 y τ_0 = 0.50. Actualización Local.	0.232757411
φ=0.99 y τ_0 = 0.05. Actualización Local.	0.200965488
φ=0.90 y τ_0 = 0.50. Actualización Global.	0.092407683
φ=0.99 y τ_0 = 0.05. Actualización Global.	0.18352499

Tabla XII (b). Parámetros considerados para el método de asignación de camiones en paralelo.

Realizando un seguimiento de la mejor solución encontrada en cada ciclo con y sin actualización local de arcos, se observó que la aplicación de actualización local provoca mayor diversidad en las soluciones encontradas, para cualquier combinación de parámetros considerada, sin importar si disminuye o no el nivel de pheromona del arco. En cambio sin la utilización de actualización local a partir de un determinado ciclo las soluciones encontradas son muy similares, convergiendo todas las hormigas a la mejor solución. Además, se realizaron pruebas utilizando como factor de persistencia local φ = 0.99. Si bien no produjo gran variación en la mejor solución final encontrada, notamos que no se produce tanta variación en las soluciones halladas en cada ciclo.

A continuación expondremos lo anteriormente explicado en forma gráfica. Los gráficos detallan la mejor solución encontrada en cada ciclo, sobre 100 ciclos de ejecución del algoritmo. Los problemas del conjunto de testeo presentados son E-n13-k4 (Figuras 16, 17 y 18) y E-n101-k14 (Figuras 19, 20 y 21). Mostraremos como evoluciona el algoritmo en los siguientes tres casos:

actualización global: utilizando el método *Mejor Solución*, sin actualización local, factor de persistencia φ = 0.90 y τ_0 = 0.50.

actualización local: utilizando el método *Pheromona Inicial*, factor de persistencia local φ = 0.90 y τ_0 = 0.50.

actualización local: utilizando el método *Pheromona Inicial*, factor de persistencia local φ = 0.99 y τ_0 = 0.50.

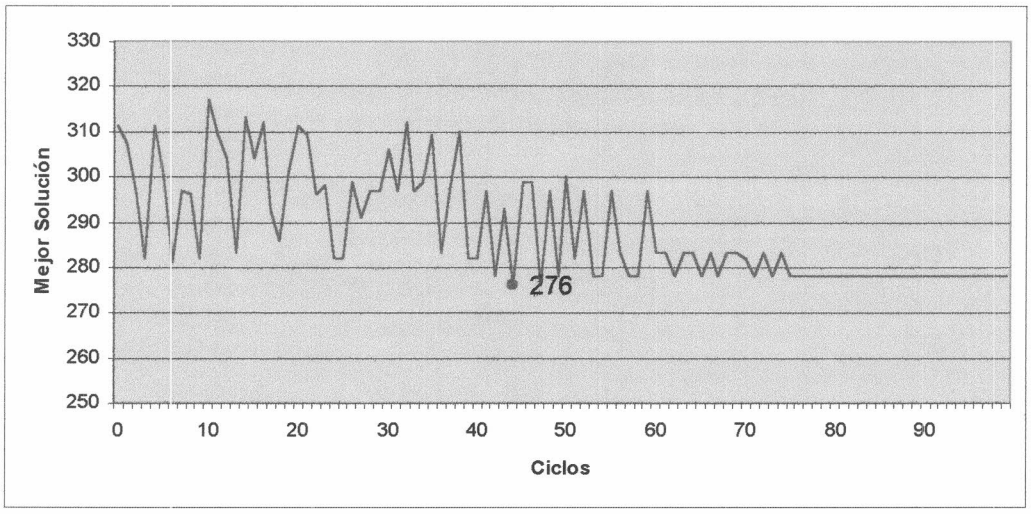


Figura 16. Actualización global de arcos. Mejor solución. Problema E-n13-k4.

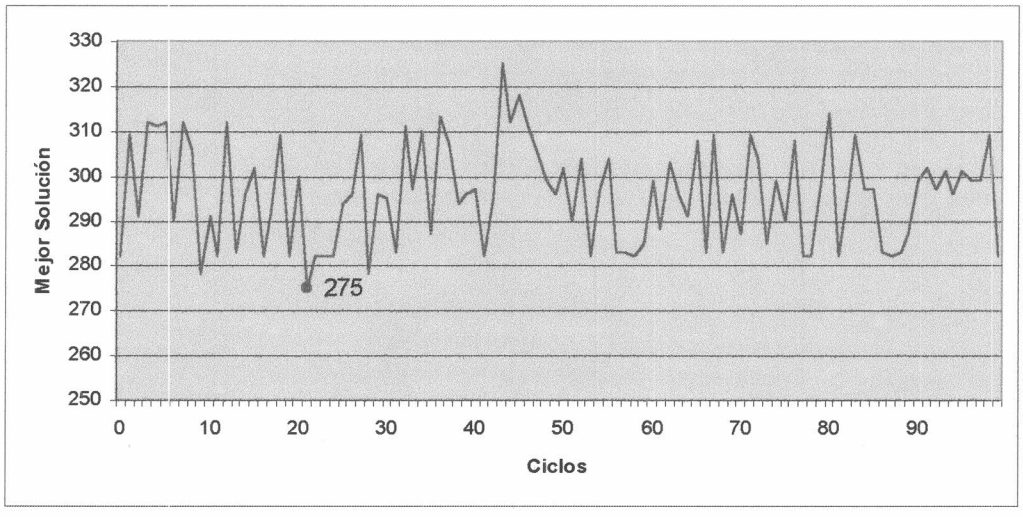


Figura 17. Actualización local de arcos. Pheromona Inicial. φ=0.90. Problema E-n13-k4.

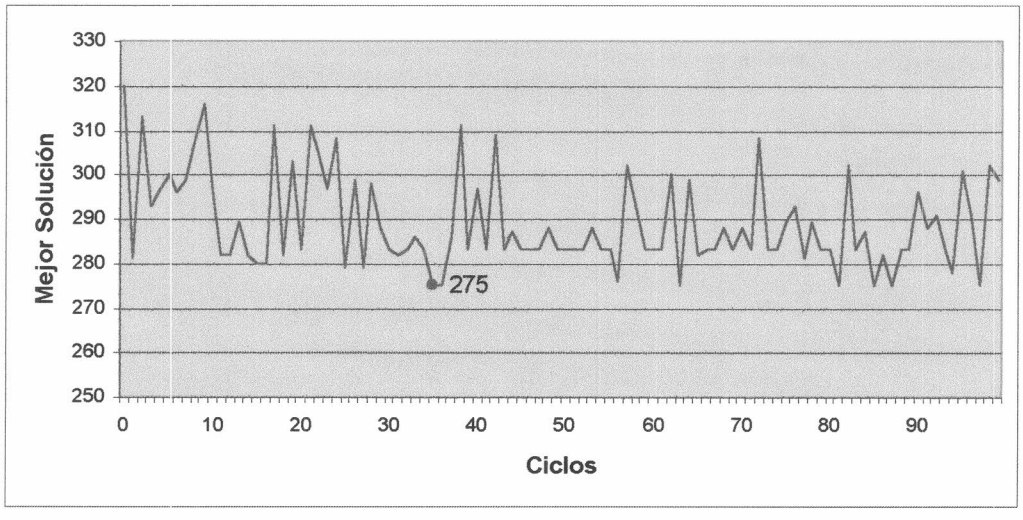


Figura 18. Actualización local de arcos. Pheromona Inicial. φ=0.99. Problema E-n13-k4.

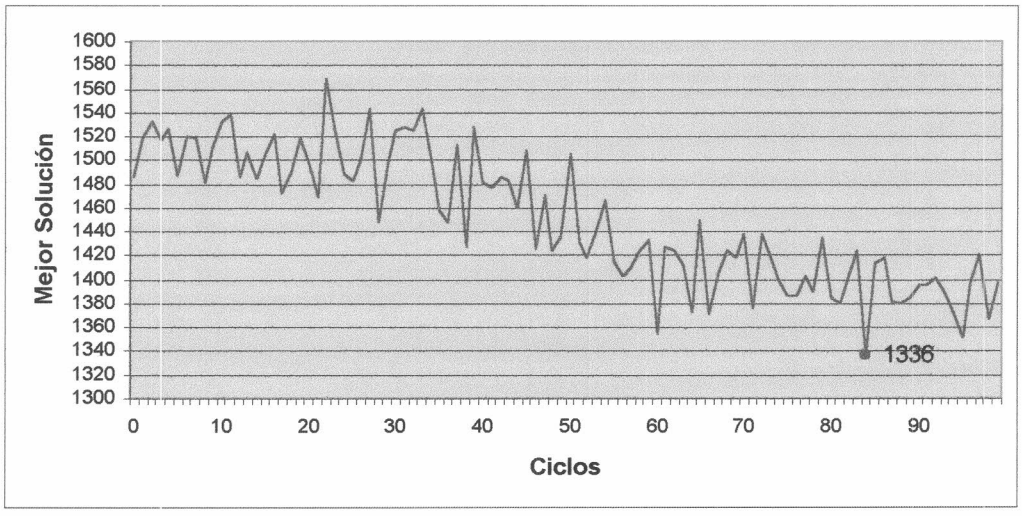


Figura 19. Actualización global de arcos. Mejor solución. Problema E-n101-k14.

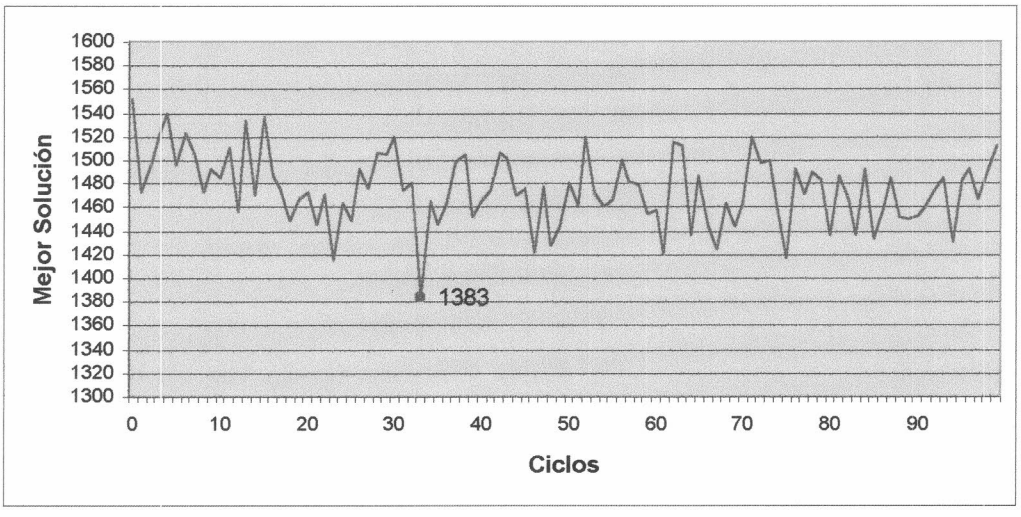


Figura 20. Actualización local de arcos. Pheromona Inicial. φ=0.90. Problema E-n101-k14.

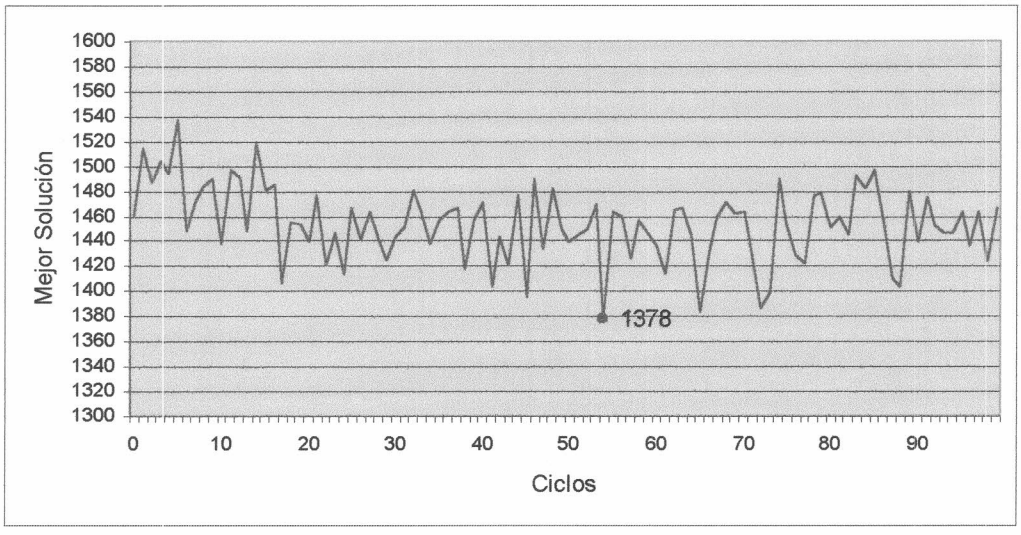


Figura 21. Actualización local de arcos. Pheromona Inicial. φ=0.99. Problema E-n101-k14.

La explicación de este comportamiento la podemos encontrar analizando la distribución de pheromona en los arcos. Aplicando sólo actualización global, se hace cada vez más notoria la diferencia de pheromona entre los arcos pertenecientes a la mejor solución y el resto de los arcos, tendiendo a que más hormigas elijan estos arcos. En cambio con la aplicación de actualización local, el nivel de pheromona se distribuye en forma más pareja en todos los arcos. Veamos esto gráficamente, tomando para ello el problema E-n13-k4 y analizando la distribución de pheromona en los arcos luego de 100 ciclos. No consideraremos los arcos que salen del depósito. Los gráficos de las Figuras 22, 23 y 24 muestran el nivel de pheromona de cada arco del grafo, para los tres casos planteados anteriormente.

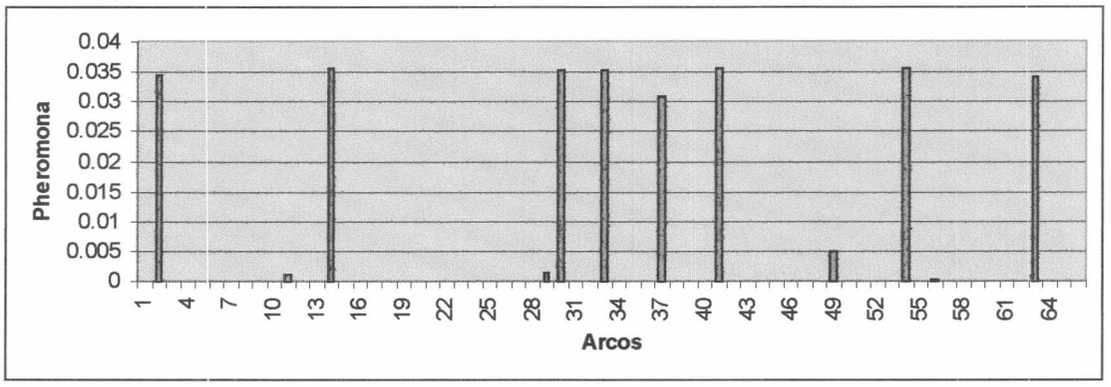


Figura 22. Actualización global arcos. Nivel de pheromona luego de 100 ciclos. Problema E-n13-k4

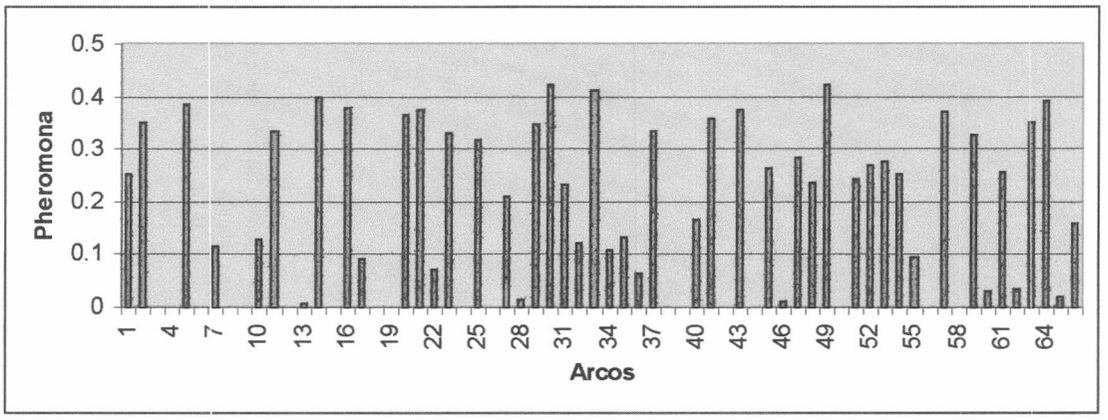


Figura 23. Actualización local de arcos. $\varphi = 0.90$. Nivel de pheromona luego de 100 ciclos. Problema E-n13-k4.

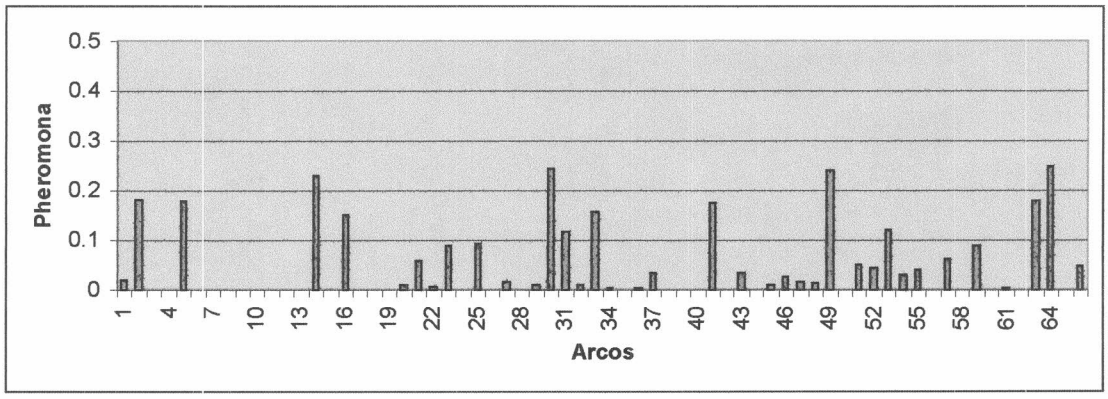


Figura 24. Actualización local de arcos. φ=0.99. Nivel de pheromona luego de 100 ciclos. Problema E-n13-k4.

Para mostrar la influencia que ejerce la relación Costo-Pheromona sobre la elección de un arco agregamos a nuestro algoritmo un *Ranking de arcos* (solo a fines estadísticos). Calculamos dos valores de ranking para cada arco: el primero indica cuantas veces el arco ha sido elegido en alguna solución de alguna hormiga, el segundo indica cuantas veces el mismo fue elegido en la mejor solución de alguna hormiga. Tomaremos el problema E-n33-k5 y analizaremos el grafo luego de 100 ciclos usando actualización local de arcos y actualización global de arcos solamente. Mostraremos la influencia que ejercen sobre la elección de un arco los siguientes valores del arco: *Visibilidad* (inversa del costo), *Pheromona* y la relación *Pheromona-Visibilidad* exponiendo los dos tipos de ranking descriptos. Denotaremos al primer tipo de ranking simplemente como *Ranking* y al segundo como *Ranking Mejor Solución.*

Los gráficos de las Figuras 25, 26 y 27 muestran el *ranking* con utilización de actualización global de arcos y las Figuras 28, 29 y 30 el *ranking mejor solución* también con actualización global. Los gráficos de las Figuras 31, 32 y 33 muestran el *ranking* para el caso de uso de actualización local de arcos y las Figuras 34, 35 y 36 el *ranking mejor solución* con actualización local. En los gráficos se observa claramente como aumenta el ranking del arco a medida que aumentan sus valores de visibilidad y/o pheromona. Nuevamente se observa que con utilización de actualización local de arcos el crecimiento es más moderado que en el caso en que solo se utilice actualización global.

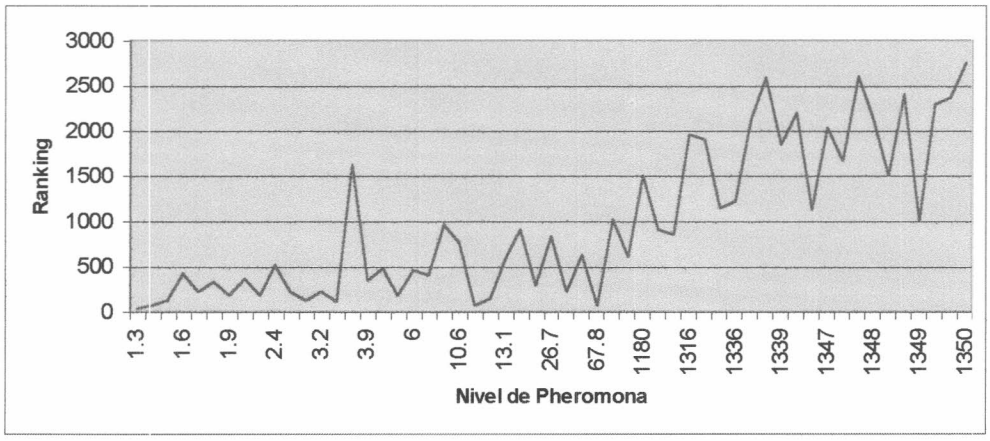


Figura 25 . Ranking de arcos. Influencia del Pheromona sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

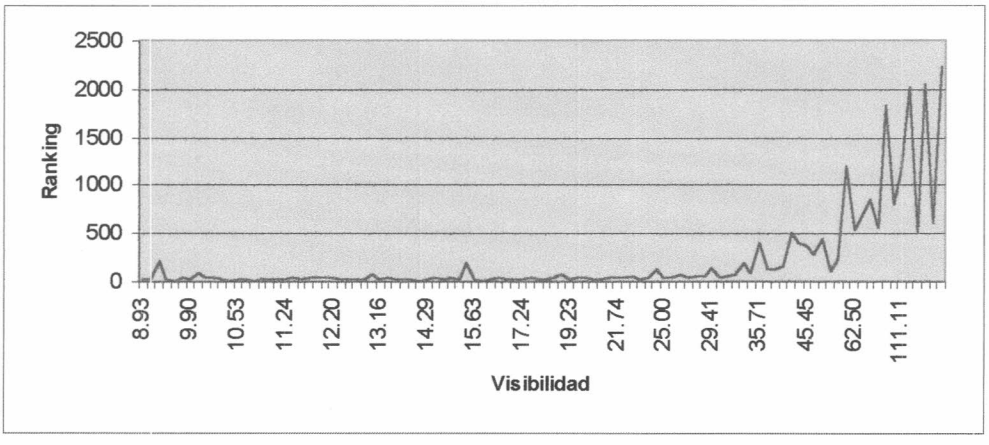


Figura 26 . Ranking de arcos. Influencia del Costo sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

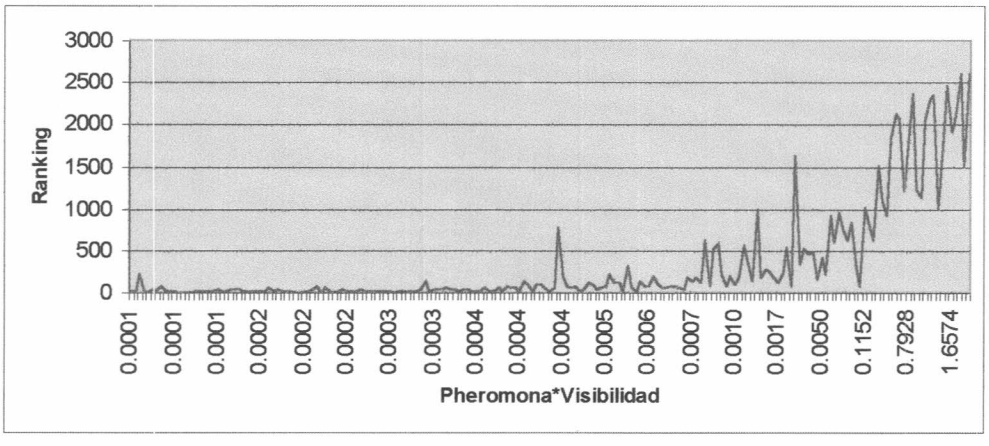


Figura 27 . Ranking de arcos. Influencia de la relación Pheromona-Visibilidad sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

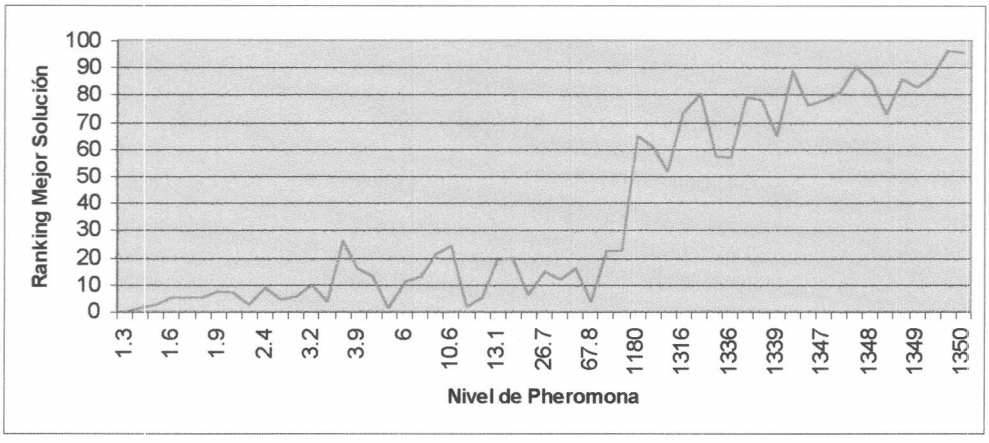


Figura 28 . Ranking Mejor Solución de arcos. Influencia del pheromona sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

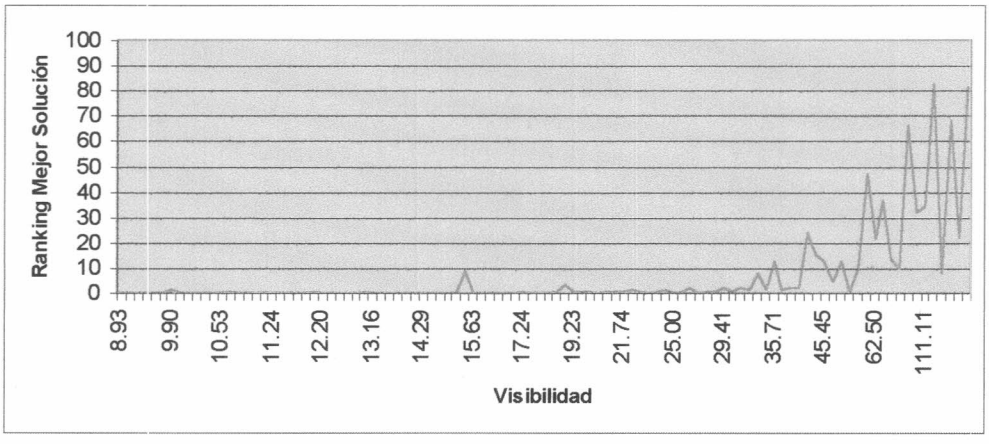


Figura 29. Ranking Mejor Solución de arcos. Influencia del Costo sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

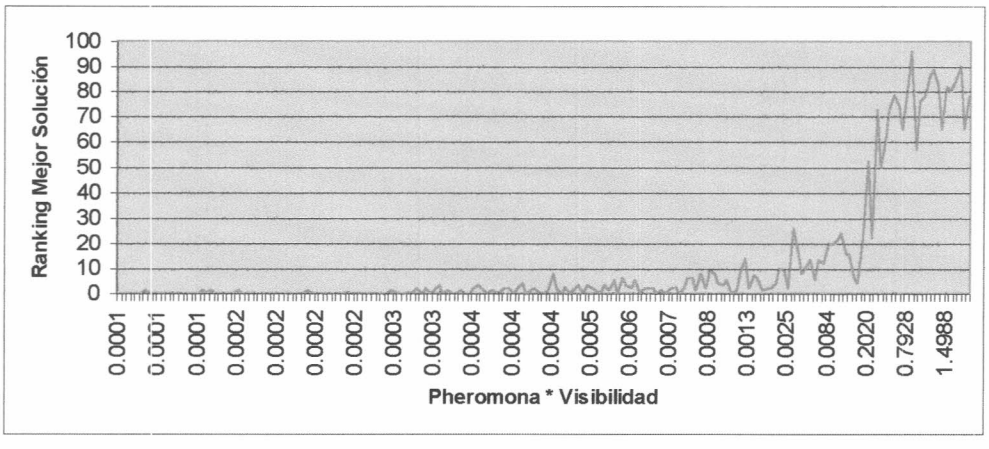


Figura 30. Ranking Mejor Solución de arcos. Influencia de la relación Pheromona-Visibilidad sobre la elección del arco con actualización Global de arcos. Problema E-n33-k5.

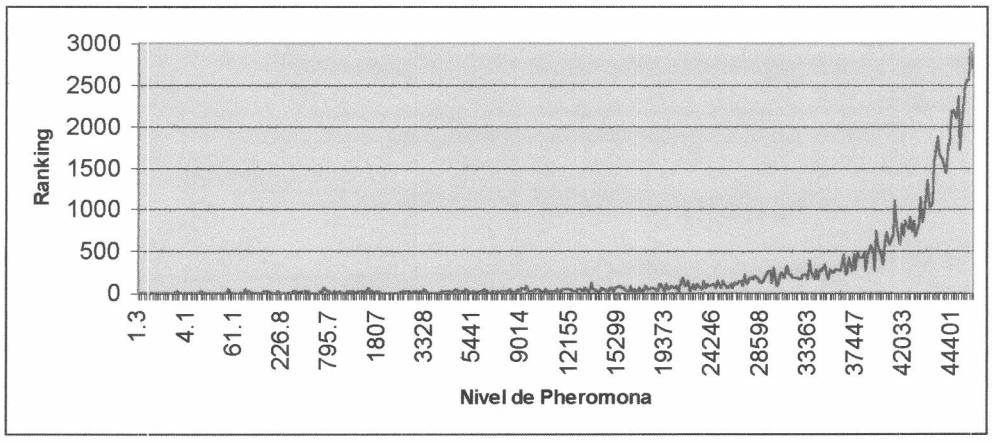


Figura 31. Ranking de arcos. Influencia del Pheromona sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

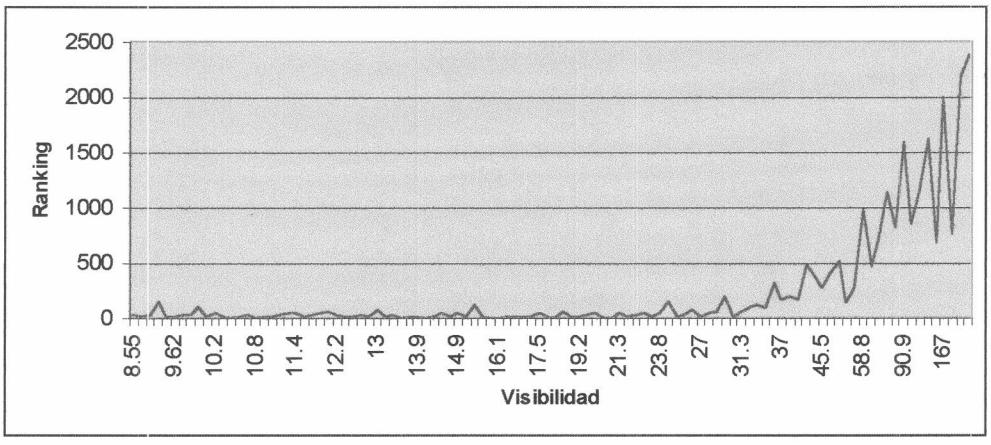


Figura 32. Ranking de arcos. Influencia del Costo sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

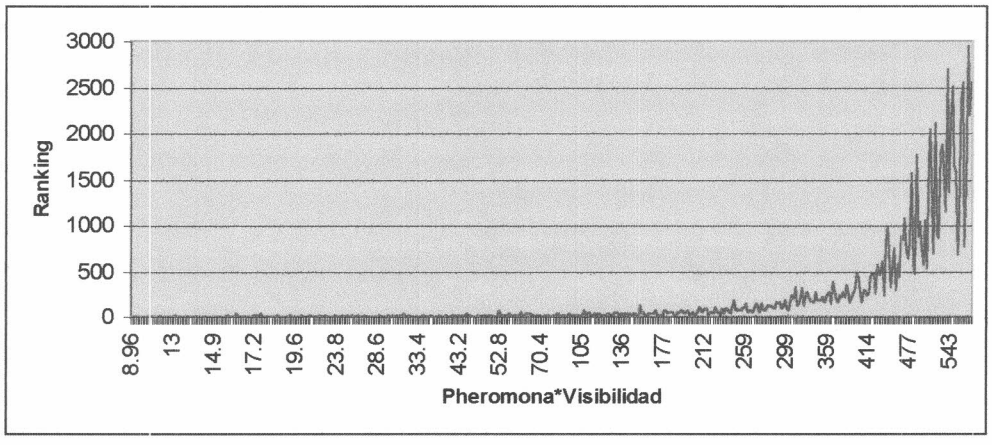


Figura 33. Ranking de arcos. Influencia de la relación Pheromona-Visibilidad sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

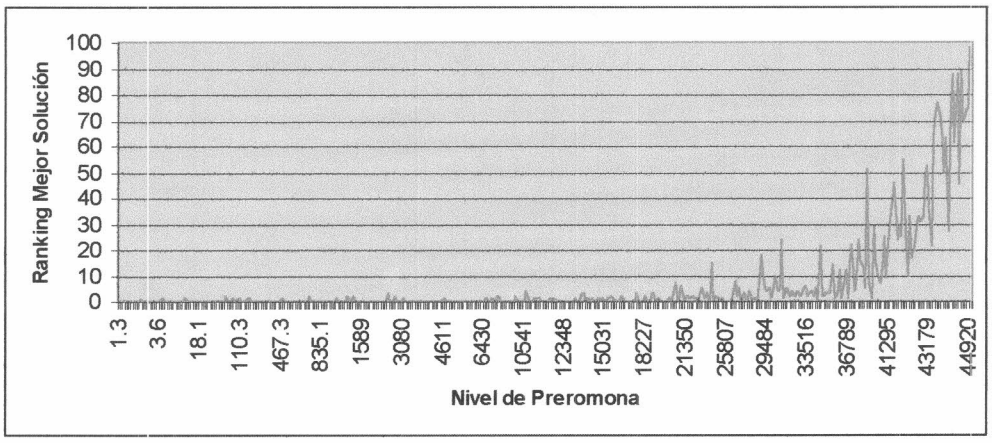


Figura 34. Ranking Mejor Solución de arcos. Influencia del Pheromona sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

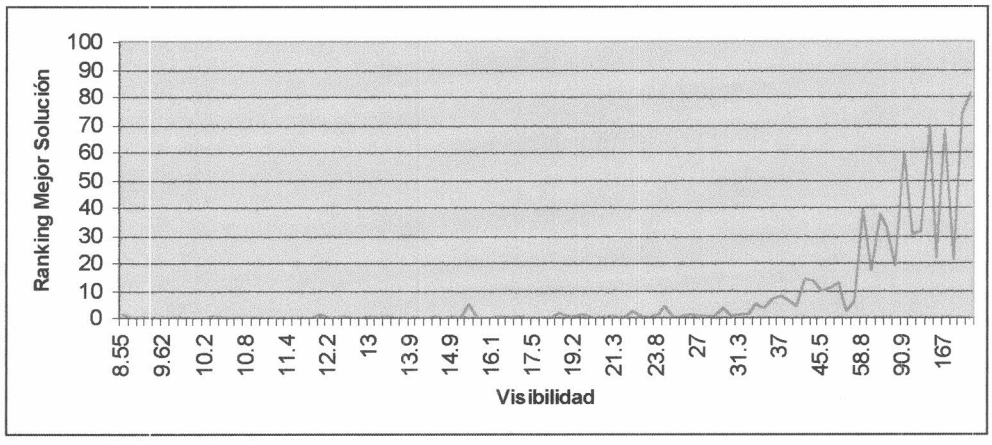


Figura 35. Ranking Mejor Solución de arcos. Influencia del Costo sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

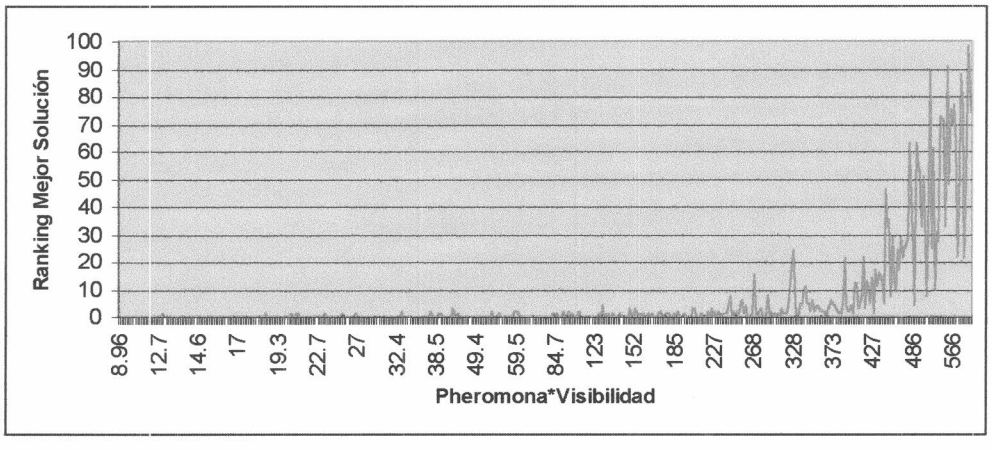


Figura 36. Ranking Mejor Solución de arcos. Influencia de la relación Pheromona-Visibilidad sobre la elección del arco con actualización Local de arcos. Problema E-n33-k5.

6.6. Listas de Candidatos

Las listas de candidatos se incorporaron con el fin de reducir los tiempos de proceso, los cuales pueden ser considerables para problemas grandes, especialmente utilizando el método de asignación de camiones en paralelo.

En esta sección describiremos los valores obtenidos utilizando las mismas. Para ambos métodos de asignación de camiones los resultados fueron muy satisfactorios, debido a que los tiempos de procesamiento fueron notoriamente inferiores a los alcanzados sin el uso de listas de candidatos sin deteriorar los errores relativos obtenidos para cada problema, los cuales fueron en algunos caso levemente inferior. El tamaño de las listas se ingresa como parámetro del problema y como porcentaje de la cantidad de nodos. Nuestras pruebas fueron realizadas con un tamaño de lista del 25%.

A continuación graficaremos los resultados obtenidos para ambos métodos de asignación de camiones, comparando los errores relativos (Figuras 37 y 39), y los tiempos alcanzados por cada problema (Figuras 38 y 40), con y sin utilización de listas de candidatos. Las pruebas fueron realizadas en una PC con procesador Intel Celeron de 333 Mhz, con 64 MB de RAM.

A partir de esta sección se han agregado como casos de testeo problemas de mayor tamaño, correspondientes a los descriptos en la Tabla I (b).

A. Método de asignación de camiones consecutivos

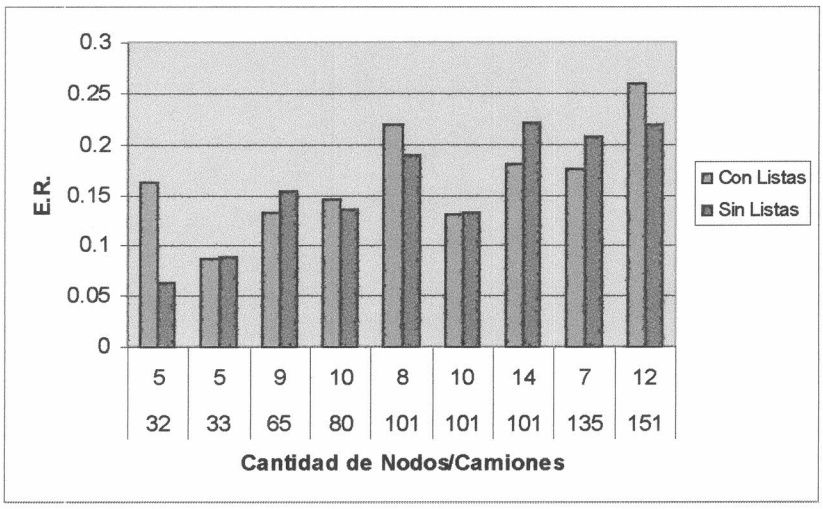


Figura 37. Utilización de listas de candidatos. Comparativo de error relativo. Camiones consecutivos.

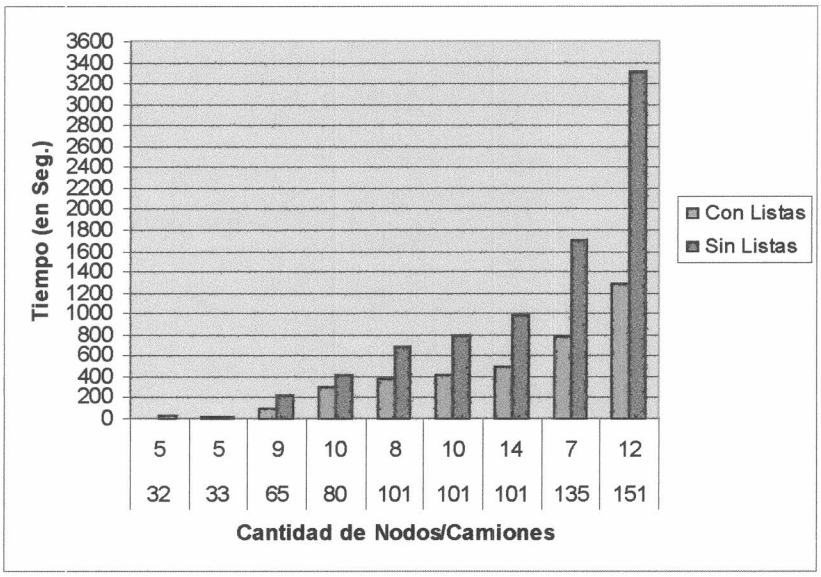


Figura 38. Utilización de listas de candidatos. Comparativo de tiempos. Camiones consecutivos. Tiempos en segundos correspondendientes a un procesador Intel Celeron de 333 Mhz.

B. Método de asignación de camiones en paralelo

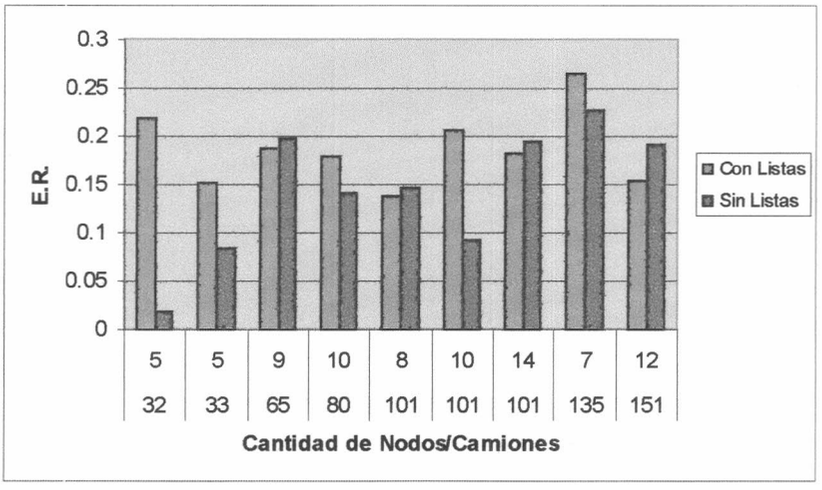


Figura 39. Utilización de listas de candidatos. Error relativo. Camiones en Paralelo.

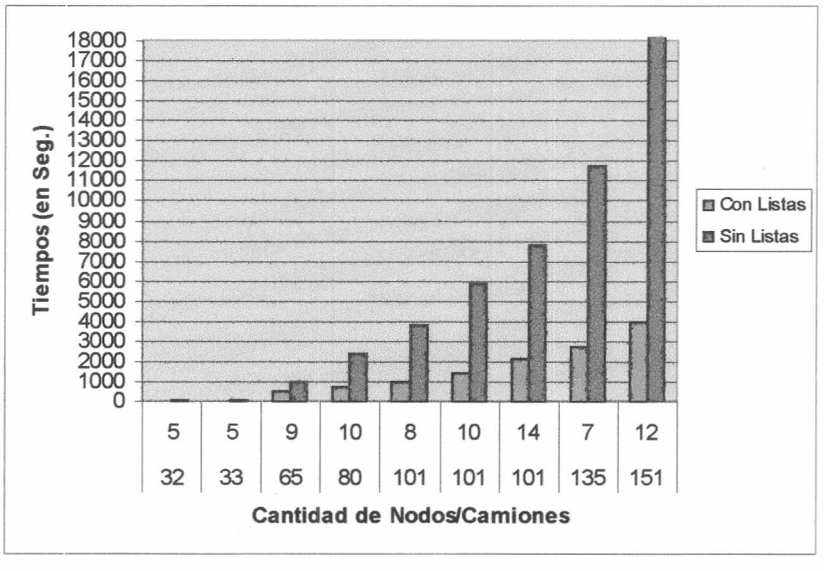


Figura 40. Utilización de listas de candidatos. Comparativo de tiempos. Camiones en paralelo. Tiempos en segundos correspondendientes a un procesador Intel Celeron de 333 Mhz.

Observemos que en algunos casos el menor error se obtuvo utilizando listas de candidatos. Si tomamos el error relativo promedio de todos los casos de prueba vemos que los mismos son muy cercanos. Las Tablas XIII y XIV resumen lo dicho.

Utilización de Listas	E.R.P
Sin listas de candidatos	0.166090863
Con listas de candidatos	0.185522404

Tabla XIII. Comparativo de utilización de listas de candidatos. Error relativo promedio. Camiones consecutivos.

Utilización de Listas	E.R.P.
Sin listas de candidatos	0.182277424
Con listas de candidatos	0.201719999

Tabla XIV. Comparativo de utilización de listas de candidatos. Error relativo promedio. Camiones en paralelo.

Resulta llamativo la diferencia obtenida con la utilización de listas de candidatos para el problema A-n32-k5 con ambos métodos de asignación de camiones. Evidentemente la limitación de selección del siguiente nodo entre los vecinos más cercanos en este problema no resulta beneficiosa. Se hicieron experimentos para este problema en particular utilizando listas de candidatos de tamaño 50 % de la cantidad de nodos, obteniéndose mejor resultado. Las Figura 41 (a) y (b) y las Tablas XV y XVI muestran los resultados.

Figura 41. Utilización de listas de candidatos para el problema A-n32-k5.
(a) Camiones consecutivos. (b) Camiones en paralelo.

Utilización de Listas	Error Relativo
Sin listas	0.06377551
Listas 50%	0.06505102
Listas 25 %	0.161989796

Tabla XV. Utilización de listas de candidatos para el problema A-n32-k5. Camiones consecutivos.

Utilización de Listas	Error Relativo
Sin listas	0.019132653
Listas 50 %	0.114795918
Listas 25 %	0.218112245

Tabla XVI. Utilización de listas de candidatos para el problema A-n32-k5. Camiones en paralelo.

6.7. Heurística de Mejora

Los experimentos realizados demuestran que la incorporación de heurística de mejora resulta altamente beneficiosa, llegándose en algunos casos (y en problemas de menor cantidad de nodos) al valor óptimo. Nuevamente, el método de asignación de camiones en paralelo produjo los mejores resultados.

Se procedió de la siguiente forma: a cada solución obtenida por cada hormiga, se le aplicó heurística de mejora durante una cantidad de ciclos, previo a la actualización global de arcos, tomando la solución mejorada para realizar dicha actualización de arcos. La última solución se mejoró sin límite de ciclos, es decir hasta ya no obtener mejora en el costo. Se realizaron varias corridas por cada caso, aplicándose 3 ciclos de mejora, tomándose el mejor resultado. Para ahorrar tiempo de proceso en los casos de actualización global de arcos correspondientes a *mejor solución* o *soluciones elites,* se intentó mejorar solamente las soluciones a utilizar en la actualización de arcos, es decir la mejor solución para el primer caso y las σ mejores soluciones para el segundo, en lugar de mejorar todas las soluciones. Las pruebas realizadas demostraron que con esta metodología los resultados obtenidos empeoraron bastante. Es evidente que al ser mejoradas, algunas soluciones de hormigas *no elites* producen menor costo que las soluciones elegidas como *soluciones elites,* las cuales quedarían descartadas utilizando la metodología descripta. Debido a esto, se procedió mejorando todas las soluciones, eligiendo la mejor (o las σ mejores, según el caso) luego de aplicada la heurística de mejoramiento.

Con la aplicación de heurística de mejora los tiempos de proceso se incrementaron. Para mejorar estos tiempos, se agregó al algoritmo la posibilidad de elegir la cantidad de hormigas a utilizar. La ciudad inicial de cada hormiga se determina en forma random. Realizamos nuevas pruebas, reduciendo la cantidad de hormigas aproximadamente al 25% de la cantidad de nodos del problema. Para los problemas de menor cantidad de nodos prácticamente no se encontró diferencia en la calidad de la solución hallada, mientras que para los problemas de mayor tamaño se obtuvo mejor resultado utilizando una hormiga por cliente. Sin embargo los tiempos de proceso se redujeron casi a un tercio. En comparación con la técnica de uso de *listas de candidatos,* con la cual utilizamos una hormiga por cliente, se obtuvo por lo general mejores resultados con el uso de listas, si bien en algunos casos puntuales el menor costo fue obtenido reduciendo la cantidad de hormigas. Nuevamente los tiempos conseguidos con la reducción en la cantidad de hormigas resultaron menores que los correspondientes al uso de listas de candidatos.

Las Tablas XVII y XIX comparan los resultados obtenidos para ambos métodos de asignación de camiones, exponiéndose los siguientes casos: *heurística de mejora sin listas de candidatos* (con *n* hormigas), *heurística de mejora con listas de candidatos* (con *n* hormigas), *heurística de mejora sin listas de candidatos* (25% de hormigas) y *sin heurística de mejora*. Mostramos en rojo los casos que alcanzaron el valor óptimo y en negrita el menor valor alcanzado distinto del óptimo. Las Tablas XVIII y XX comparan los

tiempos en segundos alcanzados por cada método; los mismos corresponden a una PC con procesador AMD XP de 1800 Mhz con 512 MB de RAM. Nuevamente indicamos en negrita el menor tiempo alcanzado.

A. Método de asignación de camiones consecutivos

Problema	Cota Superior	Mejora Sin Lista	Mejora Con Lista	Mejora 25%Hormigas	Sin Mejora
A-n32-k5	784	**810**	811	816	842
A-n33-k5	661	**674**	**674**	709	719
A-n65-k9	1174	1270	**1228**	1289	1332
A-n80-k10	1764	1943	**1937**	1944	2057
B-n31-k5	672	**703**	706	712	713
E-n13-k4	247	247	247	247	277
E-n22-k4	375	**376**	**376**	**376**	383
E-n76-k7	682	**747**	759	793	818
E-n101-k8	815	946	**913**	943	984
E-n101-k14	1077	1208	**1193**	1220	1313
M-n151-k12	1028.42	**1183**	1214	1195	1305

Tabla XVII. Uso de heurística de mejora. Cuadro comparativo de costos obtenidos. Camiones consecutivos

Problema	Mejora Sin Lista	Mejora Con Lista	Mejora 25% Hormigas	Sin Mejora
A-n32-k5	55	20	**15**	2
A-n33-k5	51	31	**18**	2
A-n65-k9	1002	501	**335**	33
A-n80-k10	**756**	1357	774	66
B-n31-k5	39	14	**8**	3
E-n13-k4	0	0	0	0
E-n22-k4	7	5	**2**	0
E-n76-k7	1607	1176	**613**	60
E-n101-k8	7142	**1724**	2538	334
E-n101-k14	10773	1987	**332**	330
M-n151-k12	35883	21278	**6398**	565

Tabla XVIII. Uso de heurística de mejora. Cuadro comparativo de tiempos. Camiones consecutivos. Tiempos en Segundos correspondientes a un procesador AMD XP de 1800 Mhz.

B. Método de asignación de camiones en paralelo

Problema	Cota Superior	Mejora Sin Lista	Mejora Con Lista	Mejora 25% Hormigas	Sin Mejora
A-n32-k5	784	784	810	**785**	799
A-n33-k5	661	661	661	**665**	731
A-n65-k9	1174	**1233**	1245	1280	1376
A-n80-k10	1764	**1898**	1929	1921	2028
B-n31-k5	672	672	684	**676**	687
E-n13-k4	247	247	247	247	258
E-n22-k4	375	375	375	375	378
E-n76-k7	682	720	**710**	719	766
E-n101-k8	815	**845**	891	884	930
E-n101-k14	1077	**1188**	1198	1219	1286
M-n151-k12	1028.42	**1105**	1142	1163	1248

Tabla XIX. Uso de heurística de mejora. Cuadro comparativo de costos obtenidos. Camiones en Paralelo.

Problema	Mejora Sin Lista	Mejora Con Lista	Mejora 25% Hormigas	Sin Mejora
A-n32-k5	53	47	**10**	11
A-n33-k5	57	**46**	50	12
A-n65-k9	1472	**261**	540	534
A-n80-k10	3833	2104	**630**	1031
B-n31-k5	25	22	**10**	25
E-n13-k4	0	0	0	0
E-n22-k4	6	3	**1**	4
E-n76-k7	1258	1659	**716**	567
E-n101-k8	7944	4239	**3079**	1902
E-n101-k14	20274	16630	**4476**	3187
M-n151-k12	42225	24175	**11466**	5884

Tabla XX. Uso de heurística de mejora. Cuadro comparativo de tiempos. Camiones en Paralelo. Tiempos en Segundos correspondientes a un procesador AMD XP de 1800 Mhz.

6.8. Comparación con otras heurísticas

Compararemos los resultados obtenidos por nuestro algoritmo con los valores obtenidos utilizando heurísticas clásicas y otras metaheurísticas. Para ello seleccionamos un conjunto de problemas, que son los descriptos en la Tabla XXI, donde indicamos la mejor cota superior conocida. Los valores que presentaremos, reportados por diversos autores, han sido extraídos de [27], eligiéndose una heurística constructiva, una heurística de mejora y otras dos metaheurísticas.

Como heurística constructiva usaremos los resultados de la heurística de *Clarke* y

Wright (versión en paralelo) [10], y *3-opt* como heurística de mejora, la cual ha sido aplicada al algoritmo de Clarke y Wright. Se han propuesto dos versiones para la heurística de mejora [27]. La primera, llamada *First Improving* (FI) ejecuta el primer intercambio que mejore el costo mientras que la segunda, *Best Improving* (BI), explora todo el vecindario para identificar la mejor operación. En nuestra comparación usaremos la versión *FI*, que es la forma en que procedemos con la heurística de mejora implementada en nuestro algoritmo. Las metaheurísticas a comparar son las implementaciones propuestas por Osman de *Simulated Annealing* y *Tabu Search* [26].

Para nuestra metaheurística de Colonia de Hormigas utilizamos la versión de asignación de camiones en paralelo, sin listas de candidatos con una hormiga por nodo y con incorporación de heurística de mejora, realizando como máximo 3 ciclos de mejora .

Problema	#Nodos	#Camiones	Cota Superior	Gap
E-n51-k5	51	5	521	0
E-n76-k10	76	10	832	?
E-n101-k8	101	8	815	?
M-n101-k10	101	10	820	0
M-n121-k7	121	7	1042.11	?
M-n151-k12	151	12	1028.42	?
M-n200-k17	200	17	1291.45	?

Tabla XXI. Set de problemas a comparar.

Problema	Clarke and Wright (1)	Clarke and Wright 3-Opt (2)	Simulated Annealing (3)	Tabu Search (4)	Colonia de Hormigas	Cota Superior	Gap
E-n51-k5	584.64	578.56	528	524.61	**521**	**521**	0
E-n76-k10	900.26	888.04	**838.62**	844	877	832	?
E-n101-k8	886.83	878.70	**829.18**	835	845	815	?
M-n101-k10	833.51	824.42	826	**819.59**	838	**820**	0
M-n121-k7	1071.07	1049.43	1176	**1042.11**	1189	1042.11	?
M-n151-k12	1133.43	1128.24	1058	**1052**	1105	1028.42	?
M-n200-k17	1395.74	1386.84	1378	**1354**	1606	1291.45	?

Tabla XXII. Comparación computacional con otras heurísticas.

(1) Clarke and Wright Savings Algorithm. Versión Paralela. [10].
(2) Clarke and Wright con 3-opt. First Improving (FI).
(3) Simulated Annealing. Osman. [26].
(4) Tabu Search. Osman. [26].

Los resultados presentados corresponden al mejor valor obtenido realizando entre dos y tres corridas del algoritmo. Para mostrar el hecho de que la técnica de Colonia de Hormigas es totalmente probabilística, variando la mejor solución obtenida en cada corrida, se tomó el problema E-n76-k10 y se realizaron doce corridas del mismo. La Tabla

XXIII muestra las soluciones obtenidas y el ciclo de ejecución en que se obtuvo la misma.

Corrida Nº	Costo Obtenido	Ciclo
1	921	42
2	907	86
3	887	83
4	891	70
5	905	80
6	892	67
7	882	89
8	905	80
9	900	94
10	881	84
11	905	61
12	**877**	**68**

Tabla XXIII. Problema E-n76-K10 con cota superior 832 . Resultados obtenidos en doce corridas del algoritmo.

6.9. Conclusiones

En esta tesis hemos aplicado la técnica metaheurística de Colonia de Hormigas al problema VRP. Presentamos dos formas de asignación de camiones: *camiones consecutivos* y *camiones en paralelo*, obteniéndose mejores resultados con el método de asignación de camiones en paralelo.

Los experimentos realizados demostraron que el método produce excelentes resultados para problemas de menor tamaño (de hasta 50 nodos), y siendo combinado con una heurística de mejora. Para todos los casos testeados de hasta 50 nodos el algoritmo alcanzó el valor óptimo en un corto tiempo. En problemas de mayor cantidad de nodos no se obtuvo tan buen resultado, aumentando bastante los tiempos de procesamiento, especialmente para el método de camiones en paralelo. Comparándolo con otras heurísticas clásicas y metaheurísticas nuestro método fue el que produjo mejor resultado para los problemas de menor tamaño.

Se utilizaron dos técnicas para disminuir los tiempos de procesamiento: la primera consiste en el uso de listas de candidatos, la cual concentra la búsqueda del siguiente vecino entre los nodos de la lista en vez de buscar en todo el vecindario; y la segunda en utilizar menor cantidad de hormigas ubicadas al azar en distintos nodos. La cantidad de hormigas se ingresó como parámetro del algoritmo. Ambas técnicas produjeron una notable disminución de tiempos, con resultados aceptables. Con respecto a las soluciones obtenidas la técnica de incorporación de listas de candidatos produjo mejores resultados que la técnica de reducción de cantidad de hormigas, sin embargo con esta última se consiguieron mejores tiempos.

Se realizaron pruebas con distintas variantes de actualización global de arcos: *todas las soluciones*, *mejor solución* y *hormigas elites*, obteniéndose mejor resultado con la técnica de *mejor solución*.

Se experimentó la incorporación de actualización local de pheromona en los arcos, en cada paso de construcción de la solución, además de la actualización global. Si bien, como sugirieron otros autores el uso de esta técnica produjo mayor variación en las

soluciones obtenidas por cada hormiga, dicha variación provocó que se disminuyera la atracción hacia los arcos correspondientes a la mejor solución, debido a la poca diferenciación de rastros de pheromona entre los arcos de la mejor solución y los del resto del grafo. Por lo tanto, se descartó el uso de esta técnica, utilizándose solamente actualización global de arcos.

Además de utilizar la regla de transición *random-proporcional* se experimentó con el uso de la regla *pseudo-random-proporcional* la cual combina selección random con selección de mejor opción en base a un parámetro q_0. Se comprobó que la selección random produce mejores resultados, obteniéndose resultados similares con la utilización de la regla random-proporcional y pseudo-random-proporcional con q_0=0.5, ya que como se espera intuitivamente con este valor de q_0 esta última debería comportarse en forma similar a la variante random.

Los distintos parámetros de ajuste del algoritmo se obtuvieron por experimentación, realizándose pruebas con distintos valores de los mismos. No se realizó un análisis detallado con respecto a los valores óptimos de dichos parámetros; es de esperarse que la aplicación de alguna forma más sistemática de ajuste, que testee las distintas combinaciones posibles podría llevar a una mejor performance.

REFERENCIAS

[1]. P. Augerat, J. M. Belenguer, E. Benavent, A. Corberán, D. Naddef, G. Rinaldi. "Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem", Research Report 949-M, Universite Joseph Fourier, Grenoble, France.

[2]. B. Bullnheimer, R. F. Hartl and C. Strauss.
"An improved ant system algorithm for the vehicle routing problem". Annals of Operations Research, 89: 319-328, (1999).

[3]. B. Bullnheimer, R. F. Hartl and C. Strauss.
"Applying the Ant System to the Vehicle Routing Problem". Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston, pp. 109-120, (1998).

[4]. L.D. Bodin, B. L. Golden, A. A. Assad, and M. Ball.
"Routing and scheduling of vehicles and crews, the state of art". Computers and Operations Research ,10:63-212, (1983).

[5]. H.M. Botee, E. Bonabeau.
"Evolving Ant Colony Optimization". Adv. Complex Systems 1, 149-159, (1998).

[6]. N. Christofides.
"Vehicle routing". In "The Traveling Salesman Problem", E.L. Lawer, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shrnoys, editors, Wiley, Chichester, UK, pp 431- 448, (1985).

[7]. N. Christofides and S. Eilon,
"An Algorithm for the Vehicle Dispatching Problem". Operational Research Quarterly 20:309-318, (1969)

[8]. N. Christofides, A. Mingozzi and P. Toth.
" The vehicle routing problem". In "Combinatorial Optimizations", N. Christofides, A. Mingozzi, P. Toth and C. Sandi, editors, Wiley, Chichester, UK, pp 315-338, (1979).

[9]. N. Christofides, A. Mingozzi and P. Toth.
"Exact Algorithms for Solving the Vehicle Routing Problem Based on Spanning Trees and Shortest Path Relaxations". Mathematical Programming **20**, 255, (1981).

[10]. G. Clarke and J.V. Wright.
"Scheduling of vehicles from a central depot to a number of delivery points". Operations Research, 12:568:581, (1964).

[11]. A. Colorni, M. Dorigo, V. Maniezzo.
"Distributed Optimization by Ant Colonies". First European Conference on Artificial Life, pp 134-142, (1991).

[12]. M. Dorigo, V. Maniezzo and A. Colorni.
"The Ant System: Optimization by a Colony of Cooperating Agentes". IEEE Transactions

on Systems, Man, and Cibernetics – Part B, Vol.26, Nº 1, pp1-13, (1996).

[13]. M. Dorigo, L. M. Gambardella.
"Ant Colony System: A Cooperative Learning Approch to the Traveling Salesman Problem". IEEE Transactions on Evolutionary Computation, Vol.1, Nº 1, (1997).

[14]. M.L.Fisher.
"Optimal solution of vehicle routing problems using minimum *k*-trees". Operations Research, 42:626-642, (1994).

[15]. L. M. Gambardella, E. Taillard and G. Agazzi.
"MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows". Technical Report IDSIA, (1999).

[16]. M.Grötchel & O. Holland.
"Solution of large escale symetric traveling salesman problems", Math. Programming, 51, 141-202, (1991).

[17]. K. Helsgaun.
"An effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic".

[18]. D. S. Johnson, L. A. Mc Geoch.
"The Traveling Salesman Problem: A Case Study in Local Optimization", (Nov. 1995).

[19]. G.A.P. Kindervater, M.W.P. Savelsbergh.
"Vehicle Routing 2. Handling Side Constraints".

[20]. G. Laporte.
"The vehicle routing problem: An overview of exact and approximate algorithms". European Journal of Operational Research. 59:345-358, (1992).

[21]. G. Laporte, Y.Nobert.
"Exact algorithms for the vehicle routing problem". Annals of Discrete Mathematics, 31:147-184, (1987).

[22]. S. Lin.
"Computer solutions of the traveling salesman problem". Bell System Technical Journal, 44:2245-2269, (1965).

[23]. S. Lin, B.W. Kernighan.
"An effective heuristic algorithm for the traveling salesman problem". Operations Research, 21:498-516, (1973).

[24]. D. Negrotto.
"Tesis de Licenciatura, basada en la resolución del problema CVRP utilizando métodos exactos". Actualmente en desarrollo.

[25]. I. Or.
"Traveling Salesman-type combinatorial optimization problems and the relation to the logistics of regional blood banking. Ph.D. dissertation. Department of Industrial

Engeneering and Management Sciences, NorthWestern University, Evanston, (1976).

[26]. I. H. Osman.
"Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem". Annals of Operations Research, 41:421-451, (1993).

[27]. P. Toth, D. Vigo, editors.
"The Vehicle Routing Problem". SIAM monographs on discrete mathematics and applications, (Dic. 2000).

[28]. P. Toth, D. Vigo.
"Exact algoritms for the vehicle routing". In "Fleet Managament and Logistic", T. Grainic and G. Laporte, editors, Kluwer, Boston, MA, pp. 1-31, (1998).

[29]. P. Toth, D. Vigo.
"Models, relaxations and exact approches for the capacitated vehicle routing problem". Discrete Applied Mathematics.