Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Algoritmos de Colonia de Hormigas para el Problema del Viajante de Comercio por Familias y para el Problema de Ruteo de Vehículos por Familias

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Alexis Soifer

Directora: Irene Loiseau

Buenos Aires, 2015

ALGORITMOS DE COLONIA DE HORMIGAS PARA EL PROBLEMA DEL VIAJANTE DE COMERCIO POR FAMILIAS Y PARA EL PROBLEMA DE RUTEO DE VEHÍCULOS POR FAMILIAS

En este trabajo presentamos una nueva variante del problema de ruteo de vehículos derivado del problema de viajante de comercio generalizado por familias (FTSP). Esta variante se denomina problema de ruteo de vehículos generalizado por familias (FVRP). En FTSP y FVRP un subconjunto de nodos debe ser visitado por cada conjunto en el grafo. El objetivo es minimizar la distancia total recorrida. Describiremos una formulación matemática para FVRP y propondremos una variante de la metaheurística colonia de hormigas conocida como sistema de la mejor-peor hormiga (SMPH) para la resolución de ambos problemas. Finalmente presentaremos los resultados obtenidos y los compararemos con los conocidos hasta el momento.

Palabras claves: FTSP, FVRP, Metaheurísticas, Optimización con Colonia de Hormigas (OCH), Sistema de la Mejor-Peor Hormiga (SMPH).

ANT COLONY OPTIMIZATION ALGORITHMS FOR FAMILY TRAVELING SALESMAN PROBLEM AND FAMILY VEHICLE ROUTING PROBLEM

We introduce a new variant of the vehicle routing problem derived from the family traveling salesman problem (FTSP). This variant is called family vehicle routing problem (FVRP). In FTSP and FVRP a subset of nodes must be visited for each node cluster in the graph. The objective is to minimize the distance traveled.
We describe an integer programming formulation for FVRP and we propose a variant of the metaheuristic ant colony system known as the best-worst ant system (BWAS) to solve both problems. Finally computational results will be presented and compared with the known to the present.

Keywords: FTSP, FVRP, Metaheuristics, Ant Colony Optimization (ACO), Best-Worst Ant System (BWAS).

AGRADECIMIENTOS

En primer lugar a mi directora de tesis, Irene, por presentarme este desafío y por estar siempre predispuesta a ayudarme desde el primer momento.

A mis viejos, Silvana y Saúl, de quienes heredé la cultura del estudio y me brindaron la posibilidad de cursar todos estos años. Fueron mi mano derecha de todos los días, me facilitaron todas las herramientas y recursos que necesité, y sin ellos no podría haber llegado donde estoy ahora.

Al resto de mi familia de quienes siempre recibí un apoyo incondicional, a mi abuela Berta, al tío Chiche, la tía Lala, y a Mario. Especialmente a mi abuela Raquel que aunque no esté la llevo siempre en mi corazón.

A Ana Luz con quien compartí estos años de mi vida y aprendimos tanto juntos.

A mis amigos de la vida que siempre me alentaron para que no estudie los fines de semana, a Brian, Browar, Román, Manolo, Moncho, Torke, Flor y Jony.

A mis compañeros de facultad por ayudarnos en cada parcial, final o TP, con apuntes, resúmenes, o lo que hubiera disponible, Leo, Calcu, Bender (tío Ben), Ivan, Ale, Uri, Vale, Pablo, Tincho, Kuja, More, Fabri, Julián, Juan Pablo, Coby, entre otros.

A mis compañeros del trabajo por estar a mi lado la mayor parte de cada día, a Lean, Marina, Charly, Walter, Victor, Ana, Pedro, Pablo, Roberto, Jorge, Damián, Paula, Gaby, y otros.

A todo el DC, especialmente a los profesores que siempre están dispuestos a explicar los mismos temas durante horas y a resolver guías enteras de ejercicios para que todos los alumnos incorporen los conceptos de las materias que imparten.

A todos aquellos que me acompañaron durante estos años recorriendo el camino del aprendizaje no solo de la computación sino de la vida.

Índice general

1. INTRODUCCIÓN

El problema del viajante de comercio (o TSP por sus siglas en inglés) fue definido a comienzos del 1800 por los matemáticos Hamilton y Kirkman [1] y consiste en encontrar la ruta más corta que recorra un conjunto de ciudades. Una generalización posible sería tener que recorrer conjuntos de ciudades, eligiendo de cada uno cual ciudad incluir en el camino. A este problema se lo denomina *problema del viajante de comercio generalizado* (GTSP) [3]. Otro caso aún más general fue definido matemáticamente por Morán-Mirabal, Gonzalez-Velarde, y Resende [5], y consiste en recorrer n de m elementos de distintos conjuntos. Este caso se denomina *problema del viajante de comercio generalizado por familias* (FTSP).

Por otro lado tenemos una familia de problemas similares donde debemos recorrer clientes, cada uno con cierta demanda. Dicha demanda se abastece con una flota de vehículos. Estos problemas que generalizan al TSP se denominan *problemas de ruteo de vehículos*. La importancia de resolver estos problemas radica en la porción importante del costo del transporte reflejado en el precio final de los productos. Dentro de la estructura de costos hasta un 35 % del valor del producto corresponde a transporte [16]. Por lo tanto mejorando las rutas se pueden obtener ahorros significativos. Según Toth y Vigo [4] las aplicaciones en casos reales han demostrado que el ahorro logrado por estas técnicas en los procesos de distribución ronda entre el 5 % y el 20 %.

En este trabajo vamos a presentar una definición matemática para el problema de ruteo de vehículos generalizado por familias (FVRP) que consiste en recorrer una cantidad arbitraria de clientes de cada conjunto.

Estos problemas pertenecen a la clase $\mathcal{NP}$-Duro para los cuales no se conocen algoritmos eficientes (de tiempo polinomial) que garanticen encontrar soluciones óptimas. Para dar noción de lo que esto implica mencionamos que en marzo de 2005 se resolvió una instancia de TSP de 33.810 puntos cuyo cálculo tomó aproximadamente 15.7 años - CPU [2].

En la práctica no se suelen resolver estos problemas de manera exacta, sino que se buscan soluciones de menor calidad a un tiempo considerablemente más rápido. Estos métodos se denominan heurísticos [9]. Hay una clase de heurísticas de alto nivel que utilizan heurísticas de bajo nivel a gran escala para producir mejores soluciones, y se denominan metaheurísticas [10].

En este trabajo, se intentarán encarar los problemas FTSP y FVRP con una metaheurística inspirada en el comportamiento de las hormigas, propuesta por Dorigo, Maniezzo, y Colorni [11]. La tesis se compone de las siguientes secciones:

Sección 2: Definiciones de los problemas a resolver
Se presentarán las formulaciones matemáticas de los problemas FTSP y FVRP.

Sección 3: Colonia de hormigas

Se presentarán las distintas variantes de colonia de hormigas.

Sección 4: Algoritmo propuesto

Se presentará el algoritmo utilizado en las resoluciones de los problemas FTSP y FVRP.

Sección 5: Resultados obtenidos

Se brindará en detalle la descripción de cada uno de los parámetros utilizados y los valores que determinamos para los mismos. Luego presentaremos los resultados provenientes de la utilización del algoritmo para la resolución de las instancias que disponemos para los problemas FTSP, GVRP y FVRP, siendo GVRP un problema de ruteo de vehículos con mayor cantidad de instancias conocidas que nos servirá para comparar nuestro algoritmo con resultados existentes.

Sección 6: Conclusiones

Se desarrollarán las conclusiones provenientes del trabajo realizado teniendo en cuenta los resultados obtenidos.

Sección 7: Trabajo Futuro

Se mencionarán posibles ideas para desarrollar en próximos trabajos.

2. DEFINICIONES DE LOS PROBLEMAS

2.1. FTSP

2.1.1. Definición

Supongamos que estamos parados en la puerta un almacén. Este almacén contiene familias de productos (yerba, galletitas, vino, etc.). Ahora supongamos que tenemos un pedido con distintos productos del almacén y el problema consiste en encontrar la ruta más corta para recoger todos los productos, y regresar a la puerta. Un ejemplo concreto sería tener una lista con 3 paquetes de yerba, 2 de galletitas y 3 de vino, y un almacén con conjuntos de estos productos distribuidos por todo el local. Este problema generaliza al problema GTSP (Generalized TSP) donde hay que visitar un individuo de cada familia, y también generaliza al clásico TSP donde hay una única familia en la cual debemos visitar cada nodo. Al generalizar un problema $\mathcal{NP}$-Duro, también nos encontramos ante un problema para el cual no se conoce ningún algoritmo exacto de tiempo polinomial que lo resuelva. Esta clase de problemas son tan costosos en términos computacionales que muchas veces se opta por resolverlo con heurísticas, es decir métodos que no nos aseguran la mejor solución pero que en muchos casos otorgan buenos resultados a mucho menor costo.

Sea G un grafo tal que $G = \{V \cup \{v_0\}, E\}$ con $V = \{v_1, ..., v_n\}$, $E = \{(v_i, v_j) : v_i, v_j \in V \cup \{v_0\}, i < j\}$, y $K = \{F_1, ..., F_l\}$ una partición del conjunto V donde cada F_l contiene los nodos de la familia l. El nodo v_0 es el origen del recorrido y el resto de los nodos son los candidatos a visitar. Consideramos a $nf_l = |F_l|$ la cantidad de visitas en la familia l, y a $KN = \sum_{l=1,...,|K|} nv_l$ la cantidad total de visitas en el recorrido. Sea $c(v_i, v_j) > 0, v_i, v_j \in V \cup \{v_0\}$ la distancia entre cada par de nodos, el objetivo es minimizar la distancia total del recorrido que comience y termine en el origen y recorra nf_l nodos de cada familia $F_l \in K$. En la imagen 2.1 vemos la solución óptima para un caso donde $|V| = 29, |K| = 4$ y $KN = 18$. Las 18 visitas se obtienen con $nv_1 = 6, nv_2 = 6, nv_3 = 1$ y $nv_4 = 5$ [5].

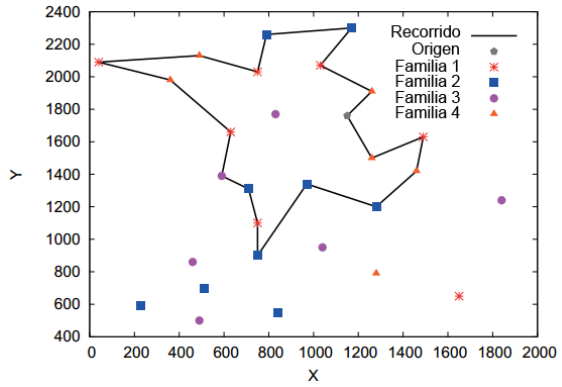

Fig. 2.1: Ejemplo de una posible solución a un problema FTSP con 29 nodos, 4 familias y 18 visitas presentada en [5].

2.1.2. Formulación matemática

La siguiente formulación para el FTSP es la propuesta por Morán-Mirabal, González-Velarde y Resende [5].

Formulación:

I. $min \sum_{v_i \in V \cup \{v_0\}} \sum_{v_j \in N \cup \{v_0\}} c_{i,j} \times x_{i,j}$

Sujeto a:

II. $\sum_{v_i \in V} x_{v_0,i} = 1$

III. $\sum_{v_j \in V} x_{j,v_0} = 1$

IV. $\sum_{v_i \in V} x_{i,j} \leq 1 \quad \forall v_j \in V$

V. $\sum_{v_j \in V} x_{i,j} \leq 1 \quad \forall v_i \in V$

VI. $\sum_{v_i \in V \cup \{v_0\}} \sum_{v_j \in V \cup \{v_0\}} x_{i,j} = KN + 1$

VII. $\sum_{v_i \in V \cup \{v_0\}} \sum_{v_j \in F_l} x_{i,j} = nv_l \quad l = 1, 2, \ldots, K$

VIII. $\sum_{v_i \in F_l} \sum_{v_j \in V \cup \{0\}} x_{i,j} = nv_l \quad l = 1, 2, \ldots, K$

IX. $\sum_{v_i \in V \cup \{v_0\}} x_{i,j} - \sum_{v_i \in V \cup \{0\}} x_{j,i} = 0 \quad \forall v_j \in V$

X. $\sum_{v_i \in S} \sum_{v_j \in S} x_{i,j} \leq |S| - 1 \quad S \subset V \cup \{v_0\}$

XI. $x_{i,j} \in \{0, 1\} \quad \forall v_i, v_j \in V$

El objetivo principal (I) es minimizar la distancia. Las restricciones (II) y (III) requieren que debamos comenzar y terminar en el depósito (nodo 0). Las restricciones (IV) y (V) requieren que cada nodo sea visitado a lo sumo una única vez. La restricción (VI) requiere que el total de nodos del recorrido deba que ser $KN + 1$, es decir, la cantidad de visitas de cada familia y el depósito. Las restricciones (VII) y (VIII) requieren que tengamos que cumplir con la cantidad de salidas y entradas (respectivamente) dentro de los nodos de cada familia. La (IX) expresa la conservación de flujo. La restricción (X) es la eliminación de los subtours. Finalmente la última restricción define a las variables binarias $x_{i,j}$ en 0 y 1.

2.2. FVRP

Como mencionamos anteriormente, este problema generaliza al FTSP. El objetivo es minimizar la distancia recorrida por un conjunto de vehículos que cumplan con la demanda de los clientes. Estos clientes están agrupados en familias, y la característica principal de esta variante es que solo hay que visitar una cantidad predeterminada de clientes por familia. Un asunto fundamental será determinar cuales clientes visitar y cuales no. FVRP también generaliza al problema GVRP (Generalized VRP) en donde se debe visitar un nodo por cada familia. Algunas de las ventajas de encarar esta problemática con mejores metodologías afectan a diversas áreas, entre las que tenemos: correo postal, manufactura, logística y distribución de mercancía en puertos, entre otros. Un ejemplo concreto sería disponer de una flota de barcos y querer depositar mercadería en 3 puertos de Argentina, 5 de Brasil, y 3 de España. Considerando que cada país posee más puertos, el algoritmo deberá escoger cuales puertos son más convenientes para incluir en el recorrido. Como nuestro único objetivo es minimizar la distancia no utilizaremos ninguna restricción en cuanto a la cantidad de vehículos.

Sea G un grafo tal que $G = \{V \cup \{v_0\}, E\}$ con $V = \{v_1, ..., v_n\}$, $E = \{(v_i, v_j) : v_i, v_j \in V \cup \{v_0\}, i < j\}$, y $K = \{F_1, ..., F_l\}$ una partición del conjunto V donde cada F_l contiene los nodos de la familia l. El nodo v_0 es el origen del recorrido y el resto de los nodos son los candidatos a visitar, Q es la capacidad de carga de los vehículos, m es la variable que determinará la cantidad de vehículos utilizados, y n la cantidad de nodos. Consideramos a $nf_l = |F_l|$ la cantidad de visitas en la familia l, y a $KN = \sum_{l=1,...,|K|} nv_l$ la cantidad total de visitas en el recorrido. Sea $c(v_i, v_j) > 0, v_i, v_j \in V \cup \{v_0\}$ la distancia entre cada par de nodos y $d(v_i), v_i \in V$ la demanda de cada nodo, asumiendo que la demanda del depósito es 0, el objetivo de FVRP es minimizar la distancia total de los recorridos que comiencen y terminen en el depósito, y que entre ellos hayan visitado nf_l nodos de cada familia l.

La formulación de este trabajo extiende a la presentada por Ha, Bostel, Langevin y Rousseau para GVRP [8], donde se utiliza un grafo auxiliar que contiene una copia del depósito y se trata el problema como uno de flujo. Para eso se define el grafo $\bar{G} = (\bar{V}, \bar{E})$ con $\bar{V} = V \cup \{v_{n+1}\}$ donde v_{n+1} es la copia del depósito. Sea $V' = \bar{V} \setminus \{v_0, v_{n+1}\}$ el conjunto de nodos sin incluir el depósito ni su copia. Consideramos $\bar{E} = E \cup \{(v_i, v_{n+1}), v_i \in V'\}$ como el conjunto de aristas de G incluyendo una arista desde cada cliente hacia la copia del depósito. Definimos $c(v_i, v_{n+1}) = c(v_0, v_i) \forall v_i \in V'$, donde determinamos que el costo del viaje entre cada cliente y la copia del depósito será la misma que entre dicho cliente y el depósito original.

Consideraremos a las variables de flujo $f_{i,j}$ como la carga de un vehículo luego de pasar por el eje i y a $f_{j,i}$ como la capacidad restante del vehículo antes de pasar por el eje i. Entonces $f_{j,i} = Q - f_{i,j}$. Para entender mejor estas variables presentamos el gráfico 2.2 también proveniente de [8]. Este muestra una posible solución de GVRP para el conjunto de familias F1, ..., F5 con demandas 10, 20, 20, 20 y 20 respectivamente. La capacidad de los vehículos es de 50. Las lineas sólidas representan las cargas (los $f_{i,j}$) mientras que las punteadas representan los espacios libres (los $f_{j,i}$).

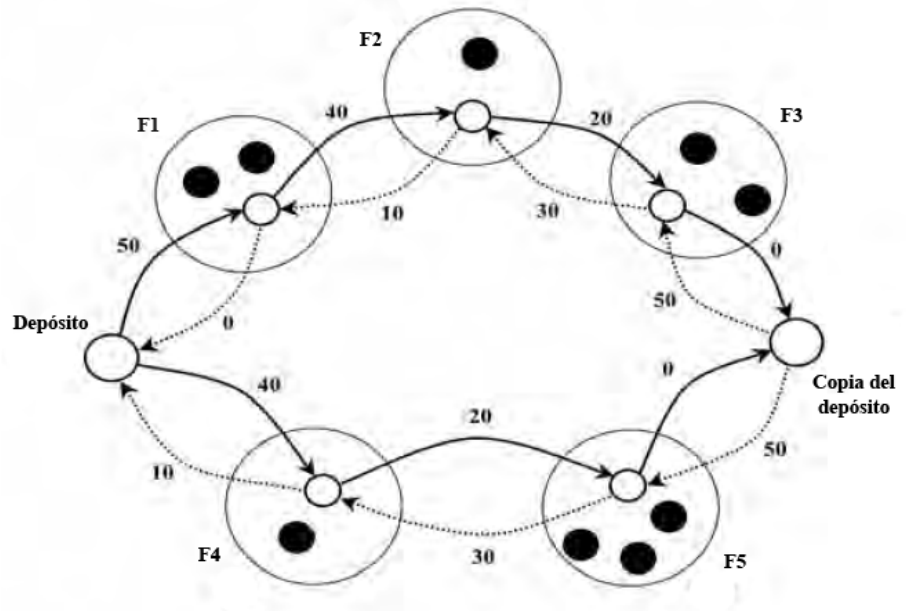

Fig. 2.2: Ejemplo de una posible solución a un problema GVRP presentada en [8].

En GVRP se asume que satisfaciendo la demanda de cualquier nodo se cumple también con la demanda de su familia. Consideramos a $d'(l) \geq 0, l = 1, ..., |K|$ como la demanda de la familia l. Luego:

$$d(v_i) = d'(l) \ \forall v_i \in F_l, l = 1, ..., |K|.$$

En FVRP asumiremos que cualquier conjunto de nodos que debamos visitar en cada familia cumplirá con demanda de su grupo, entonces:

$$\sum_{v_i \in V''} d(v_i) \geq d'(l) \ \ \forall V'' \subseteq F_l, |V''| = nv_l \quad l = 1, ..., |K|.$$

En este caso estamos generalizando al GVRP de dos formas. Por un lado agregamos la posibilidad de recorrer más de un nodo, y por el otro permitimos que el conjunto de nodos que se visiten superen a la demanda de la familia. Entonces el algoritmo tendrá que tener en cuenta la capacidad del vehículo además de la distancia por recorrer.

2.2.1. Formulación matemática

La base de la formulación que desarrollaremos a continuación proviene de la formulación del problema de ruteo de vehículos generalizado presentada por Ha, Bostel, Langevin y Rousseau [8].

Formulación:

I. $min \sum_{(v_i,v_j)\in \bar{E}} c_{i,j} x_{i,j}$

Sujeto a:

II. $\sum_{v_i \in F_l} y_i = nv_l \quad \forall F_l \in F$

III. $\sum_{v_i \in \bar{V}, i<k} x_{i,k} + \sum_{v_j \in \bar{V}, j>k} x_{k,j} = 2y_k \quad \forall v_k \in V'$

IV. $\sum_{v_j \in \bar{V}} (f_{j,i} - f_{i,j}) = 2d(v_i)y_i \quad \forall v_i \in V'$

V. $\sum_{v_j \in V'} f_{v_0,j} = \sum_{v_i \in V'} d(v_i)y_i$

VI. $\sum_{j \in V'} f_{n+1,j} = mQ$

VII. $f_{i,j} + f_{j,i} = Qx_{i,j} \quad \forall \{v_i, v_j\} \in \bar{E}$

VIII. $f_{i,j} \geq 0, f_{j,i} \geq 0 \quad \forall \{v_i, v_j\} \in \bar{E}$

IX. $x_{i,j} \in \{0,1\} \quad \forall \{v_i, v_j\} \in \bar{E}$

X. $y_i \in \{0,1\} \quad \forall v_i \in V'$

XI. $m \in \mathbb{N}$

El objetivo principal es el mismo que en FTSP, minimizar la distancia recorrida. La restricción (II) limita la cantidad de visitas por familia. La restricción (III) limita la cantidad de visitas por vértice a 1. Entre los puntos (IV) y (VII) definimos las variables relacionadas con el flujo. En particular la restricción (IV) limita el flujo por nodo según su demanda. La restricción (V) determina que el flujo de salida del depósito equivale al total de la demanda de los nodos utilizados. La restricción (VI) limita el flujo total a la capacidad de los transportes. La restricción (VII) determina que la suma del espacio libre de un vehículo y de la carga que este trae debe ser igual a su capacidad. Las últimas restricciones son definiciones de variables.

3. COLONIA DE HORMIGAS

3.1. Introducción

Como se mencionó anteriormente existen problemas que afectan a diversos campos y que son muy difíciles de resolver por su tamaño y complejidad. Los métodos exactos no resultan eficientes en instancias de gran tamaño o bien nunca logran determinar la solución óptima, mientras que las heurísticas clásicas no siempre producen soluciones de buena calidad. Las metaheurísticas son métodos que emplean heurísticas para producir soluciones de mejor calidad [10]. Estas están inspiradas en diversos campos como la genética, la biología, la física, entre otros. Entre ellas encontramos métodos como simulating annealing, la búsqueda tabú, la búsqueda local iterativa, GRASP ("greedy randomized adapatative search procedures"), algoritmos evolutivos, optimización por enjambre de partículas o redes neuronales.

La metaheurística basada en colonia de hormigas se desarrolló a partir de observar como ciertas especies de hormigas con capacidades visuales muy limitadas logran alcanzar su alimento realizando muy buenos recorridos [11]. Ellas se comunican a través de una sustancia química que depositan a medida que realizan su recorrido denominada feromona. Esta sustancia le permite saber al resto de la colonia que aquel camino fue utilizado. Un grupo de hormigas abandona su hogar en busca de alimento y ante una bifurcación un grupo elige una alternativa, mientras que el otro va por la restante (figura 3.1). Un grupo llegará antes que el otro, luego recoge la comida y regresa por donde vino. Mientras tanto otro grupo parte desde la colonia y se encuentra ante la misma bifurcación, pero esta vez por un sendero ha vuelto el primer grupo. Este nuevo grupo detecta mayor presencia de feromonas en el camino más corto favoreciendo la toma de la mejor decisión. Esta elección no es exacta sino probabilística, de hecho se asume que cuando ciertos individuos eligen otro camino promueven el descubrimiento de nuevas rutas. Con el paso del tiempo la feromona se evapora provocando que los peores caminos dejen de utilizarse.

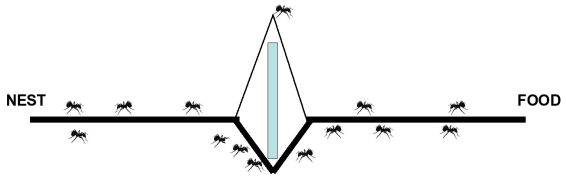

Fig. 3.1: El experimento del puente mencionado en [11] con hormigas argentinas (*iridomyrmex humilis*) muestra como esta especie de hormiga logra hallar el mejor camino para conseguir alimento.

3.2. De las hormigas naturales, a la metaheurística

La idea general del algoritmo se basa en abstraer el comportamiento observado y representarlo artificialmente. Por un lado representamos el sendero de las hormigas con un grafo donde cada nodo representa una bifurcación, y donde cada arista contiene dos datos: información heurística e información de los rastros de feromona. La información heurística es información *a priori* y nos ayuda durante la ejecución del algoritmo para influir a la colonia para que converja más rápidamente, mientras que la información de los rastros de feromona es un valor que se va acumulando con el transcurso del tiempo dependiendo de la calidad de los caminos se que encuentren.

Por otro lado las hormigas son agentes computacionales independientes unos de otros y construyen soluciones tomando decisiones en cada intersección teniendo en cuenta la información que mencionamos anteriormente. Cada hormiga representa una solución completa del problema. En la variante clásica cuando las hormigas finalizan su recorrido se evalúa la calidad de las soluciones obtenidas y en base a estos resultados se deposita mayor o menor cantidad de feromona. Se desarrollaron distintas variantes de esta metaheurística y sus diferencias principales radican en como se deposita la feromona, ya sea durante el recorrido o al finalizarlo, en base a la cantidad de hormigas que se tomarán en cuenta para realizar el depósito, o bien el tratamiento especial que se le dará a la feromona depositada por la mejor hormiga.

3.3. Estructura básica

En la estructura general del algoritmo durante un período de tiempo se crean distintas generaciones de hormigas que recorrerán concurrentemente (sincrónica o asincrónicamente) los distintos nodos *construyendo* en cada caso una solución al problema (algoritmo 1). Las hormigas toman en cuenta las feromonas depositadas en iteraciones anteriores y un valor heurístico que corresponde a cada problema en particular (algoritmo 3). Durante el recorrido pueden o no realizar modificaciones en la feromona. Algunas variantes no solo agregan feromona sino que además la restan ya sea durante el recorrido para evitar que distintas hormigas utilicen el mismo camino o bien al finalizar la iteración para evitar que una mala solución sea tenida en cuenta. Luego se procede a una evaporación de la feromona favoreciendo que las hormigas exploren nuevas regiones y colaborando a evitar que se estanque la búsqueda [12]. Finalmente se pueden realizar distintas acciones finales (acciones del demonio, algoritmo 1), entre ellas: depositar feromona extra en el recorrido de la mejor hormiga, depositar feromona aleatoriamente (comportamiento genético), o mejorar los recorridos obtenidos con búsquedas locales.

Sea $G_{ch} = \{V_{ch}, E_{ch}\}$ grafo conexo. Definimos al vecindario de $v_i \in V_{ch}$ como $N(v_i) = v_j : (v_i, v_j) \in E_{ch}$. Cada hormiga recorre el grafo moviéndose a través del vecindario de cada nodo, teniendo en cuenta ciertas restricciones que debe cumplir. Dicho conjunto de restricciones será Ω. Luego los posibles movimientos de una hormiga desde v_i en la iteración k estarán dados por el conjunto de vecinos del nodo v_i, restando los movimientos que no cumplan con las restricciones Ω. Llamamos a la función que determina estos posibles movimientos como $V_\Omega^k(v_i)$. Esta función también tiene en cuenta la memoria de la hormiga hasta la iteración actual. Como mencionamos previamente la decisión sobre cual será el

siguiente nodo está determinada por feromonas depositadas en iteraciones anteriores y un valor heurístico. Consideramos a $\eta_{i,j}$ como la información heurística y $\tau_{i,j}$ como el valor de la feromona correspondientes al eje $(v_i, v_j) \in E_{ch}$. Sea σ el parámetro que determina la velocidad de evaporación del valor de la feromona en cada iteración, al cual denominamos *factor de evaporación* y α y β los parámetros que determinan la ponderación de la información proveniente de las feromonas y la de la información heurística en la toma de decisión del movimiento de cada hormiga respectivamente. Si $\alpha = 0$ no se utilizaría la información de la feromona y si $\beta = 0$ no se utilizaría la información heurística calculada previamente. Definiremos $p^k_{i,j}$ como la probabilidad de transición entre entre los nodos v_i y v_j.

Algoritmo 1: Estructura básica SH

```
Inicialización de parámetros();
while condición de reinicio do
    // Generación de hormigas y actividad
    for w = 1 hasta número_hormigas do
        Nueva Hormiga(w);
    // Evaporación de la feromona
    forall the (v_i, v_j) ∈ E_ch do
        τ_{i,j} *= 1 - σ;
    Acciones del demonio();
```

Algoritmo 2: Nueva hormiga

```
Data: id de la hormiga
inicializa nueva hormiga(id_hormiga);
m = actualiza memoria hormiga();
while estado actual != estado objetivo do
    t = leer los datos de la tabla de transiciones();
    pt = calcular probabilidades de transición(t, m, Ω);
    siguiente estado = aplicar política de transición(pt);
    mover al siguiente estado(siguiente estado);
    if actualización de feromona en linea paso a paso then
        depositar feromona en el arco visitado();
        actualizar datos de la tabla de transiciones();
    m = actualizar estado interno();
if actualización de feromona en linea a posteriori then
    forall the e in arcos visitados do
        depositar feromona en el arco visitado();
        actualizar datos de la tabla de transiciones();
```

Algoritmo 3: Calcular probabilidades de transición en la versión clásica del SH.

Data: tabla de datos de transiciones, memoria de la hormiga, Ω

$$p_{i,j}^k = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta}{\sum_{u \in V_\Omega^k(i)} [\tau_{i,u}]^\alpha * [\eta_{i,u}]^\beta} & \text{si } j \in V_\Omega^k(i) \\ 0 & \text{en otro caso} \end{cases} \quad ([a])$$

[a] τ corresponde al valor de la feromona de la arista, mientras que η al valor de la información heurística.

3.4. Algoritmos de colonia de hormigas

3.4.1. Sistema de hormigas (SH)

Es el sistema básico con el que se comenzó el desarrollo de esta metaheurística (algoritmo 1). Todas las hormigas aportan feromona cuando terminan su proceso. Una mejora sobre este modelo, es que la mejor hormiga deposita feromona extra en su recorrido. Las siguientes variantes tienen mejor rendimiento [17].

3.4.2. Sistema de colonia de hormigas (SCH)

Como mencionamos, el modelo anterior no tenía muy buen rendimiento ([17]). Colorni, Dorigo, Maniezzo [13] proponen este sistema el cual se diferencia del anterior en lo siguiente:

Feromona: Solo la mejor hormiga deposita su feromona sobre su camino. La evaporación se realiza a medida que las hormigas pasan por las aristas. De esta manera se disminuye la probabilidad de que distintas hormigas utilicen los mismos recorridos, favoreciendo la diversificación.

Transición: Se introduce una regla pseudo-aleatoria de transición en la cual hay un porcentaje (por lo general 90 %) en el cual se elige la mejor opción (la que posee mayor nivel de feromonas e información heurística), mientras que en el otro 10 % se utiliza la regla convencional (algoritmo 3) [14].

3.4.3. Sistema de hormigas Min-Max (SHMM)

Este sistema difiere del anterior limitando los niveles de feromona entre τ_{min} y τ_{max}, calculados en una corrida previa del algoritmo. Cuando se alcanza el τ_{max} se reinicia el algoritmo inicializando las mejores aristas con τ_{max}. Esto favorece la explotación de las mejores soluciones [15].

3.4.4. Sistema de hormigas con ordenación

Es otra extensión del SH propuesto por Bullnheimer, Hartl y Strauss [11]. Se ordenan las hormigas para actualizar la feromona, siendo la mejor la más privilegiada.

3.4.5. Sistema de la mejor-peor hormiga (SMPH)

Esta versión de colonia de hormigas es la que utilizamos en este trabajo. Detallaremos a continuación las características esenciales.

Feromona: Se agrega en el mejor camino conocido hasta el momento, y se resta en el peor. Además se agrega o resta feromona aleatoriamente, mejorando la diversidad. Este concepto proviene de los algoritmos evolutivos [17].

Transición: Se usa la misma regla del sistema de hormigas clásico.

Evaporación: Se aplica a todas las aristas por igual, también proveniente del sistema clásico.

Proceso de mejora: Se realiza una búsqueda local en las mejores hormigas.

Reinicio: Si no hubo mejoras en cierta cantidad de iteraciones, o cierto tiempo, se reinicia la cantidad de feromona.

En la siguiente sección describiremos como se aplica esta metaheurística a los problemas que queremos resolver.

4. ALGORITMOS DE COLONIAS DE HORMIGAS PROPUESTOS

Comenzaremos con la aplicación de esta metaheurística al problema FTSP, y luego desarrollaremos las principales diferencias respecto a la variante para resolver FVRP.

4.1. Algoritmo de colonia de hormigas para el FTSP

4.1.1. Estructura general

Algoritmo 4: Proceso principal.

```
Inicializar información heurística();
while condición de reinicio do
    Reiniciar feromona();
    while condición de corte do
        Generación de hormigas y actividad();
        Evaporación de la feromona();
        Acciones del demonio();
```

Un factor importante en esta etapa es el reinicio de la feromona, que será determinante en la diversificación de los caminos encontrados ya que reducimos la probabilidad de estancamiento en mínimos locales. Luego ejecutaremos a las tres acciones principales de esta metaheurística: la generación de hormigas, la evaporación de feromonas, y las acciones del demonio. A continuación desarrollaremos cada etapa del algoritmo.

4.1.1.1 Inicialización de la información heurística

La información heurística (algoritmo 3) se determina al inicio del programa. Existen diversas formas de determinar este valor. En algunos casos se utilizan otros métodos heurísticos para calcular distintas soluciones y con esa información escoger que aristas se verán más beneficiadas. En otros casos se utiliza una función que devuelve mayores valores a medida que la distancia entre cada par de nodos disminuye. Sea $\eta_{i,j}$ la información heurística del eje (v_i, v_j). La siguiente sería una función factible:

$$\eta_{i,j} = \frac{1}{c(r,s)}$$

Durante el desarrollo de nuestro algoritmo notamos que no resultaba conveniente trabajar con valores tan variables como la distancia. Por un lado buscamos mantener un equilibrio entre los niveles de feromona y la información heurística, y por otro lado el hecho de poder detener el algoritmo en cualquier momento y en cualquier instancia y comprender con mayor precisión los valores de la información entre cada par de nodos para efectuar mejores conclusiones. Entonces se determinó la normalización de todos los valores tanto de las feromonas como de la información heurística entre 1 y 100.

Sean d_{max} y d_{min} la mayor y menor distancia existente en el grafo respectivamente. Luego, con esos datos podemos realizar la normalización y así obtener valores relativos dentro de la escala que nos resulte cómoda. Definimos a $\iota(x, x_{min}, x_{max}, a, b)$ como la función de normalización del valor x entre a y b:

$$\iota(x, x_{min}, x_{max}, a, b) = a + \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}}$$

Luego obtenemos el valor final de la información heurística para cada eje entre 1 y 100:

$$\eta_{i,j} = \frac{1}{\iota(c(i,j), d_{min}, d_{max}, 1, 100)} \times 100$$

4.1.1.2 Criterios de reinicio y corte

Detallaremos los criterios elegidos para las condiciones de reinicio y de corte utilizadas en el proceso principal (algoritmo 4).

Condición de reinicio: cuando se cumple esta condición actualizamos la feromona con valor cero, dejando al proceso como en su estado inicial. Se detienen los reinicios cuando se logra la cantidad deseada, o se supera el tiempo máximo sin obtener ninguna mejora. Tanto la cantidad máxima de reinicios como el tiempo máximo sin obtener mejoras son parámetros del programa que describiremos en la sección 5.

Condición de corte: en este bucle es donde las hormigas depositan su feromona y esta es aprovechada por las siguientes generaciones. Para finalizar estas iteraciones utilizamos dos condiciones de corte: cantidad de iteraciones sin mejora, y el período de tiempo máximo sin mejoras. En el primer caso consideramos que la cantidad de iteraciones debería ser proporcional a la cantidad de nodos. Sin embargo en instancias muy grandes o muy pequeñas no se observó un buen rendimiento ya que ocurría que se iteraba menos de lo que se podría o más de lo que hubiera sido conveniente. Como nuestro objetivo es que el proceso se adapte con mayor facilidad a distintas instancias decidimos utilizar tanto un umbral de cantidad mínima y máxima de iteraciones, como un valor proporcional según la cantidad de nodos.

4.1.2. Generación de hormigas y actividad

Este procedimiento consiste en la creación de las hormigas y luego en la realización del movimiento de cada una (algoritmo 5). Como mencionamos previamente en la descripción del algoritmo las hormigas son totalmente independientes unas de otras dentro de cada iteración, y por eso esta etapa de la búsqueda es paralelizable. Esta última característica no se encuentra en las variantes de colonia de hormigas donde se actualiza la feromona a medida que se realizan los movimientos, como por ejemplo en primeros experimentos del SH. En nuestra implementación efectivamente realizamos dicha paralelización.

El movimiento de cada hormiga estaba determinado en principio de la misma forma que en el SH convencional mencionado en el algoritmo 3. Un cambio introducido en la implementación actual es que el cálculo de probabilidad mencionado no se aplica en la totalidad de los nodos, sino en un vecindario restringido. Llamamos $V_r(v_i)$ al vecindario

restringido del nodo v_i que estará determinado por un cierto porcentaje de sus nodos más cercanos. Esta restricción se debe a que según diversas observaciones la probabilidad de que el mejor camino contenga aristas entre nodos muy lejanos es muy baja [16]. En cambio si no se encontrara un vecino disponible efectivamente se procede con el resto de los nodos. Según pruebas realizadas la mejora con la introducción del vecindario fue muy contundente. En la sección 5 describiremos con mayor precisión dicha mejora.

En el algoritmo 5 de la sección 3 mencionamos que los posibles movimientos de una hormiga en la iteración k están dados por un conjunto de restricciones. En este caso la restricción es que podamos movernos a nodos sin visitar de familias que aún no hayamos terminado de recorrer, siempre priorizando el vecindario restringido mencionado anteriormente.

Algoritmo 5: Generación de hormigas y actividad.

for *h = 0 to cantidad de hormigas a generar* **do**
- Inicializar hormiga();
- **while** *resten movimientos por realizar* **do**
  - v_i = nodo actual;
  - **forall the** $v_j \in V_r(v_i)$, v_j *no visitado* **do**

$$p_{i,j}^k = \begin{cases} \frac{[\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta}{\sum_{u \in V_\Omega^k(i)} [\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta} & \text{si } s \in V_\Omega^k(i) \\ 0 & \text{en otro caso} \end{cases} ;$$

  - **if** $\nexists v_j \in V_r(v_i)$ **then**
    - **forall the** $v_j \in V(v_i)$, v_j *no visitado* **do**

$$p_{i,j}^k = \begin{cases} \frac{[\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta}{\sum_{u \in V_\Omega^k(i)} [\tau_{i,u}]^\alpha * [\eta_{i,u}]^\beta} & \text{si } j \in V_\Omega^k(i) \\ 0 & \text{en otro caso} \end{cases} ;$$

  - seleccionamos el siguiente nodo entre los disponibles según su probabilidad de ser visitado y movemos la hormiga();

4.1.3. Evaporación de la feromona

Esta etapa del algoritmo se conserva de la misma forma que en SH (algoritmo 1) [11]. El objetivo es disminuir las feromonas de los ejes en un cierto porcentaje. Este parámetro lo denominamos *factor de evaporación* y lo representamos con σ. A mayor porcentaje de evaporación, mayor el porcentaje con el que disminuirá la feromona. En el algoritmo 6 observamos que se itera sobre todos los ejes para aplicar la modificación correspondiente.

Algoritmo 6: Evaporación de la feromona.

forall the $(v_i, v_j) \in E$ **do**
- $\tau_{i,j}$ *= 1 - σ;

4.1.4. Acciones del demonio

Esta sección es fundamental para aprovechar toda la información proveniente de las hormigas. En primer lugar vamos a aplicar una búsqueda local (que detallaremos más adelante) a un determinado porcentaje de hormigas mejor ubicadas según el costo de su recorrido. A continuación actualizamos la mejor y la peor solución encontrada. Luego depositamos feromona en el mejor recorrido encontrado hasta el momento y restamos del peor. Finalmente se aplica una mutación en la feromona para ayudar a diversificar el algoritmo [17]. A continuación describiremos mejor cada uno de estos pasos.

Algoritmo 7: Acciones del demonio.

```
forall the hormiga entre las mejores hormigas do
    Búsqueda Local (hormiga);
Actualizar mejor y peor hormiga hasta el momento();
Actualización de feromonas();
Realizar mutación();
```

4.1.4.1 Depósito de feromona

Tanto en el depósito de la feromona extra como en la sustracción de la misma necesitamos determinar correctamente las cantidades. Sea c un circuito que represente una solución al problema y $l(c)$ la longitud del circuito c, definimos a κ como la función que calcula el promedio de las distancias normalizadas de sus ejes:

$$\kappa(c) = \frac{\sum_{(v_i,v_j)\in c} \iota(c(i,j), d_{min}, d_{max}, 1, 100)}{l(c)}$$

Ahora utilizamos esta última ecuación para definir la función $\lambda(c)$ que le otorga el valor a cada camino encontrado. A medida que el recorrido es más corto se devuelve un resultado más elevado.

$$\lambda(c) = \frac{1}{\kappa(c)} \times 100$$

Sea γ un parámetro que determina la proporción del valor del camino que utilizaremos para aumentar la feromona de cada eje del mejor camino. A este parámetro lo llamamos *factor de mejora*. A mayor valor de dicho factor aumentamos la intensificación de la búsqueda. Finalmente definimos a μ como el valor de la feromona a depositar en cada eje de la mejor solución:

$$\mu(c) = \lambda(c) \times \gamma$$

Sea δ un parámetro que determina la proporción del valor del camino que utilizaremos para disminuir la feromona de cada eje del peor camino. A este parámetro lo llamamos *factor de desmejora*. Definimos a ν como el valor de la feromona a retirar en cada eje de la peor solución.

$$\nu(c) = \lambda(c) \times \delta$$

Sea $\tau_{i,j}$ el valor de la feromona del eje (v_i, v_j), presentamos la función que se encarga de actualizar la feromona mencionada en el algoritmo 7.

Algoritmo 8: Actualización de feromonas.

```
c1 = mejor solución encontrada();
valor_suma = μ(c1);
forall the (v_i, v_j) ∈ c1 do
    τ_{i,j} += valor_suma;
c2 = peor solución encontrada();
valor_resta = μ(c2);
forall the (v_i, v_j) ∈ c2 do
    τ_{i,j} -= valor_resta;
```

4.1.4.2 Mutación de la feromona

En el proceso de mutación debemos determinar que porcentaje de aristas se verán afectadas y la cantidad de feromona utilizar (algoritmo 9). Para determinar este último valor se emplea el valor que se suma a cada eje en la actualización de feromonas (función μ) [17]. En esta implementación buscamos que a medida que se estanque la búsqueda se emplee una mayor cantidad de feromona. Por eso al valor de la feromona depositado por la mejor solución lo multiplicamos por valores de entre 0.5 y 1.5 según nos acerquemos a la cantidad máxima de iteraciones sin que se produzcan mejoras durante el proceso. Desde ya, para obtener siempre valores entre 0.5 y 1.5 debemos realizar una normalización de la cantidad de iteraciones efectuadas hasta el momento (por eso la utilización de la función ι). Sea ξ la cantidad de iteraciones sin mejora actual, que varía entre 1 y π siendo esta la máxima cantidad de iteraciones sin mejora aceptada a lo largo del proceso, definimos a ρ como la cantidad de feromona que altera cada arista de la mutación:

$$\rho = \mu(c) \times \iota(\xi, 1, \pi, 0{,}5, 1{,}5)$$

Por otro lado necesitamos intensificar la diversificación a medida que se estanca el algoritmo. Cuando la cantidad de iteraciones o tiempo transcurrido sin que se produzca ninguna mejora supera a la mitad de los máximos estipulados, realizamos alteraciones en un porcentaje mayor de aristas. Por eso disponemos de los parámetros *porcentaje normal* y *porcentaje fuerte* de mutación que serán utilizados según corresponda. En pruebas preliminares esta modificación produjo buenos resultados, más adelante en la sección 5 entraremos más en detalle.

Por último, el valor calculado previamente para modificar un eje puede utilizarse tanto para sumarse como para restarse. Esta determinación será tomada de manera aleatoria. El eje a modificar de cada nodo pertenecerá a su vecindario restringido. Pruebas realizadas nos indican que si la modificación es en cualquier eje de cada nodo esta realmente no logra los resultados esperados, en cambio modificando parte de su vecindario restringido la mutación surge mayor efecto.

A continuación presentaremos el algoritmo de mutación mencionado en el algoritmo 7.

Algoritmo 9: Detalle de la mutación

```
c = mejor solucion encontrada();
ρ = μ(c) × ι(ξ, 1, π, 0,5, 1,5);
if ξ ≥ π/2 then
    p = porcentaje fuerte de mutación;
else
    p = porcentaje normal de mutación;
forall the v_i ∈ V do
    a = aleatorio(0, 100);
    if a < p then
        v_j = nodo aleatorio perteneciente a V_r(v_i);
        suma o resta = aleatorio(0,1);
        if suma o resta = 1 then
            τ_{i,j} += ρ;
        else
            τ_{i,j} -= ρ;
```

4.1.5. Búsqueda local (BL)

La búsqueda local consiste en partir de una solución inicial y aplicar reiteradamente alguna transformación hasta que deje de haber mejoras. En nuestra búsqueda utilizamos los siguientes algoritmos: reubicación, intercambio de nodos, or-opt, intercambio de caminos, 2-opt, e intercambio interfamilia. Luego de describir cada uno, detallaremos el orden y la cantidad de veces que los ejecutamos. Un detalle no menor, es que en la implementación contamos con dos alternativas: recorrer todo el espacio de búsqueda, o intentar realizar movimientos al azar hasta que haya una cierta cantidad de iteraciones sin ninguna mejora.

En el caso donde recorremos todo el espacio de búsqueda no nos movemos al mejor vecino sino al primero cuyo costo sea menor al actual. Luego de realizar dicho movimiento no volvemos a comenzar la exploración desde el comienzo sino que partimos desde la posición actual ya que no se encuentran grandes mejoras en el espacio ya explorado. Por otro lado también aplicamos la variante del primer mejor vecino encontrado en el caso donde realizamos movimientos al azar.

4.1.5.1 Reubicación

Esta transformación toma un nodo y lo mueve dentro del camino. Por lo tanto la complejidad de recorrer todo el vecindario será de $O(\textit{longitud camino}^2)$. En la figura 4.1 se podrá apreciar con claridad el efecto resultante de realizar dicho movimiento.

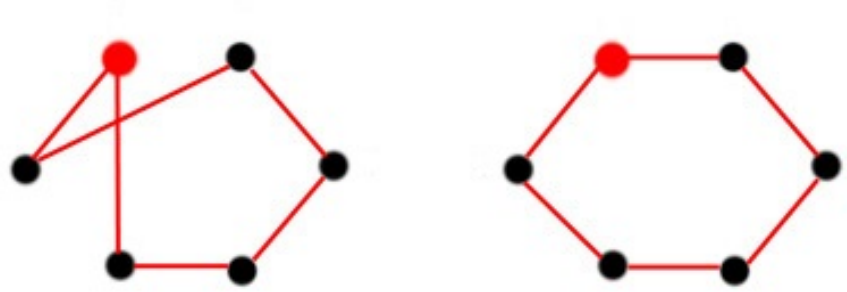

Fig. 4.1: Reubicación: se puede observar que al cambiar la ubicación del nodo rojo dentro del circuito reducimos la distancia total.

4.1.5.2 Intercambio

En este caso se seleccionan 2 nodos y se intercambian sus posiciones. Nuevamente la complejidad de recorrer el vecindario será de $O(longitud\ camino^2)$, ya que por cada nodo, tratamos de intercambiarlo con otro y así reducir el costo total del recorrido (figura 4.2).

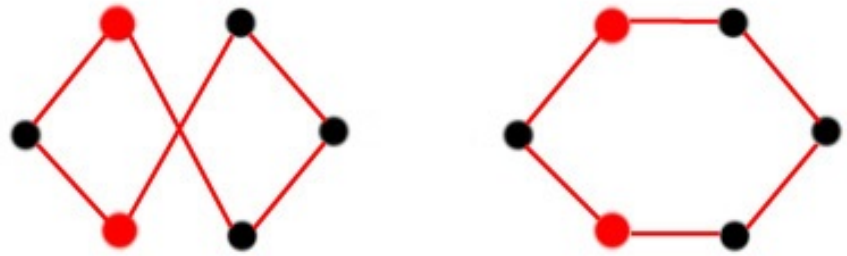

Fig. 4.2: Intercambio: cambiando los nodos entre sí la distancia total se reduce.

4.1.5.3 Or-opt

Or-opt es una generalización de reubicación, toma una porción del camino (de tamaño variable) y la quiere cambiar de lugar. Esta vez no solamente hay que tener en cuenta la nueva ubicación sino la longitud del camino que queremos mover (figura 4.3). Es por eso que la complejidad es $O(longitud\ camino^3)$ ya que hay que recorrer el camino, y por cada distancia, intentar reubicar la cadena en alguna otra parte del mismo.

4.1.5.4 Intercambio de caminos

En esta transformación, se toman 2 porciones del camino y se intercambian entre sí (figura 4.4). Es una generalización del intercambio entre nodos, pero con cadenas de longitud variable. Trabaja en un mayor espacio de búsqueda, donde la complejidad de recorrerlo es $O(longitud\ camino^3)$, por los mismos motivos que en or-opt.

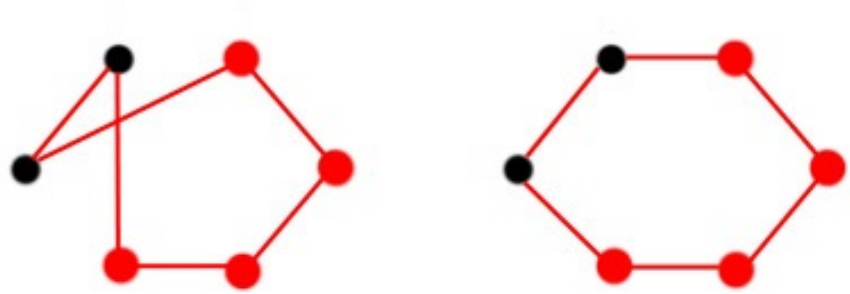

Fig. 4.3: Or-opt: tomamos el mismo ejemplo que en la búsqueda local reubicación, pero esta vez en lugar de mover el primer nodo hacia el final, movemos el resto de los nodos hacia el principio.

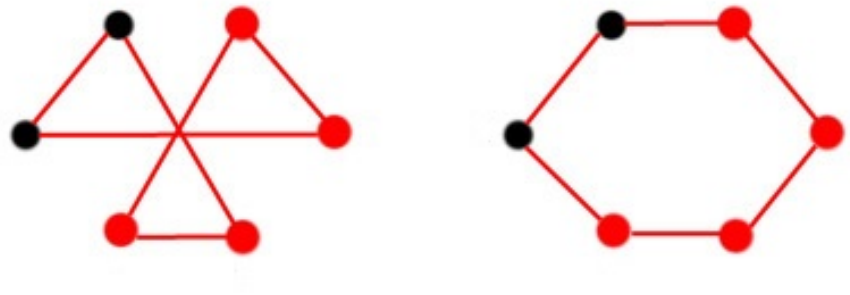

Fig. 4.4: Intercambio de caminos: se puede observar que el intercambio de ambos tramos reduce la distancia total.

4.1.5.5 2-opt

2-opt selecciona un tramo del camino y lo invierte (figura 4.5). La complejidad de recorrer el espacio es de $O(longitud\ camino^2)$ ya que por cada sección del camino (nodo + longitud) intentamos realizar la operación.

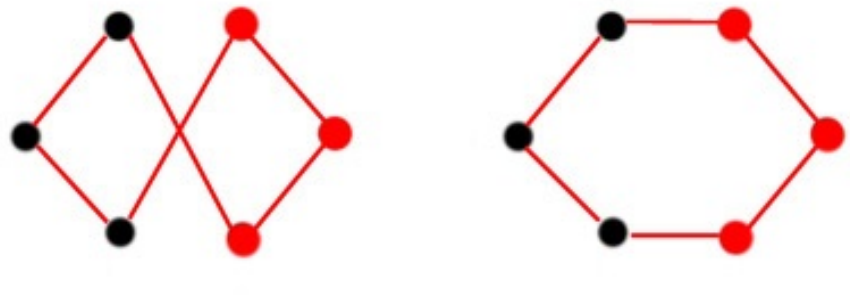

Fig. 4.5: 2-opt: en este gráfico se muestra como invirtiendo la cadena reducimos la distancia total.

4.1.5.6 Intercambio interfamilia

Esta transformación es la única que incluye mejoras tomando en cuenta los nodos no utilizados en las familias. Por cada familia se obtienen 2 conjuntos: el de los nodos utilizados y el de los no utilizados en el camino. Luego se intentan intercambiar uno de cada conjunto. El espacio del vecindario introducido es de $O(cantidad\ nodos^2)$.

4.1.5.7 Utilización de los algoritmos de mejora

Según distintas pruebas, las búsquedas sobre espacios reducidos y luego sobre espacios más amplios producen buenos resultados [16]. Es por eso que realizamos las operaciones en el siguiente orden:

1. reubicación
2. intercambio
3. or-opt
4. intercambio de caminos
5. 2-opt
6. intercambio interfamilia

Estas operaciones las ejecutamos una cierta cantidad de veces. En distintas pruebas realizadas determinamos que repitiendo las operaciones 4 veces obtenemos un balance adecuado entre tiempo y calidad. Como mencionamos anteriormente debemos optar por recorrer todo el espacio o solo una parte. En la sección 5 evaluaremos que conviene elegir, y en caso de recorrer solo una parte del espacio cual será la cantidad máxima de iteraciones que permitiremos sin notar ninguna mejora.

4.2. Algoritmo de colonia de hormigas para el FVRP

Como mencionamos previamente FVRP generaliza al FTSP ya que permite que los nodos tengan capacidades y se dispone de una cantidad infinita de transportes con capacidad limitada para visitarlos. El algoritmo para la resolución de FVRP se basa en el algoritmo de FTSP introduciendo una serie de cambios que contemplan la utilización de m vehículos, incluyendo las demandas de los clientes y la capacidad del transporte. A continuación enumeramos las etapas del algoritmo FTSP y mencionamos dichos cambios.

Estructura general: En esta etapa no se registran cambios, ya que solo estamos ejecutando la estructura básica de esta versión del OCH.

Generación de hormigas y actividad: Como en FTSP, cada hormiga construye una solución completa. Con respecto al movimiento de las hormigas realizamos leves cambios debido a la naturaleza del problema (algoritmo 10). En primer lugar se verifica la capacidad del transporte antes de considerar el movimiento hacia otro nodo. Si no hay nodos disponibles se finaliza el recorrido y se agrega otro vehículo. Por otro lado agregamos la posibilidad de que los vehículos retornen al depósito prematuramente. Se deben cumplir las siguientes condiciones: no debe haber ningún vecino disponible

y el depósito debe estar a menor distancia que cualquier otro nodo disponible, o bien más cercano que el vecino más distante. Si se dan estas condiciones, utilizamos el valor del parámetro *probabilidad de finalización temprana* para definir si damos por finalizado el recorrido o no.

Evaporación de la feromona: Ocurre de la misma manera que en el caso anterior, respetando la estructura básica del algoritmo.

Acciones del demonio: Son idénticas a las del algoritmo de FTSP salvo que en lugar de repartir feromona en base al mejor recorrido, se toman en cuenta todos los recorridos de la mejor solución que hubo hasta el momento.

Búsqueda local: Se realizan importantes cambios que serán descriptos en esta sección.

Algoritmo 10: Generación de hormigas y actividad para FVRP.

for *h = 0 to cantidad de hormigas a generar* **do**
  Inicializar hormiga();
  while *resten movimientos por realizar y el vehículo tenga capacidad* **do**
    v_i = nodo actual;
    forall the $v_j \in V_r(v_i)$, v_j *no visitado y con capacidad para atenderlo* **do**

$$p_{i,j}^k = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta}{\sum_{u \in V_\Omega^k(i)} [\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta} & \text{si } s \in V_\Omega^k(i) \\ 0 & \text{en otro caso} \end{cases} ;$$

    if $\nexists v_j \in V_r(v_i)$ **then**
      forall the $v_j \in V(v_i)$, v_j *no visitado y con capacidad para atenderlo* **do**

$$p_{i,j}^k = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha * [\eta_{i,j}]^\beta}{\sum_{u \in V_\Omega^k(i)} [\tau_{i,u}]^\alpha * [\eta_{i,u}]^\beta} & \text{si } j \in V_\Omega^k(i) \\ 0 & \text{en otro caso} \end{cases} ;$$

      if *la distancia al depósito es menor que la del vecino más lejano o es menor que cualquier nodo disponible para movernos* **then**
        finalizamos el recorrido con una probabilidad definida por parámetro();
    if $\exists v_j$ *con probabilidad de ser visitado* **then**
      seleccionamos el siguiente nodo entre los disponibles según su probabilidad de ser visitado y movemos la hormiga();
    else
      agregamos otro vehículo y comenzamos nuevamente los movimientos desde el depósito();

4.2.1. Búsqueda Local (BL)

En esta etapa del algoritmo mantendremos las transformaciones que se empleaban en FTSP aplicadas a cada vehículo de la solución de FVRP. La única transformación que no se realiza es el intercambio interfamilia ya que se podría estar intercambiando un nodo de un vehículo por un nodo utilizado en algún otro. En este caso utilizamos otra transformación que contempla intercambios teniendo en cuenta todos los vehículos a la vez (intercambio interfamilia intervehículos). Por otro lado agregamos distintas transformaciones que apliquen sobre el conjunto de vehículos.

4.2.1.1 Reubicación intervehículos

Su funcionamiento es igual a la reubicación en el recorrido de un solo vehículo, con la diferencia que intenta intercambiar nodos entre los recorridos de distintos vehículos, respetando su capacidad máxima. Su complejidad es la misma que en reubicación para un único vehículo.

4.2.1.2 Intercambio intervehículos

Su funcionamiento es igual al intercambio en el recorrido de un solo vehículo, con la generalización que se realiza para todos los vehículos. Su complejidad es la misma que en intercambio para un único vehículo.

4.2.1.3 Intercambio de caminos intervehículos

Su funcionamiento es igual al intercambio de caminos en el recorrido de un solo vehículo, con la generalización que se realizan cruces entre los recorridos de todos los vehículos. Su complejidad es la misma que en intercambio de caminos para un único vehículo.

4.2.1.4 Intercambio de cola

Intentar cambiar todas cadenas de longitud arbitraría entre sí, requiere un trabajo computacional muy grande. Si evitamos explorar todo el espacio de búsqueda reducimos el trabajo pero perdemos la oportunidad de realizar mayor cantidad de mejoras. El intercambio de cola posee buen rendimiento en un espacio de búsqueda mucho menor, del orden de $O(cantidad\ nodos^2)$ [16].

4.2.1.5 Intercambio interfamilia intervehículos

Su funcionamiento es igual al intercambio interfamilia en el recorrido de un solo vehículo solo que teniendo en cuenta que el nodo a reemplazar no debe estar en ningún recorrido de otro vehículo (ya que ese movimiento ya lo realiza intercambio intervehículos). Su complejidad es la misma que en el intercambio interfamilia para un único vehículo.

4.2.1.6 Utilización de los algoritmos de mejora

Primero aplicamos los algoritmos entre caminos, y luego los que son por caminos. El siguiente proceso lo realizamos dos veces.

1. reubicación intervehículos
2. intercambio intervehículos
3. intercambio de caminos intervehículos
4. intercambio de cola
5. por cada camino: reubicación
6. por cada camino: intercambio
7. por cada camino: or-opt
8. por cada camino: intercambio de caminos
9. por cada camino: 2-opt
10. intercambio interfamilia intervehículos

Una excepción a la utilización de los algoritmos de mejora por camino se realiza cuando el mismo posee hasta 6 nodos. En esos casos resulta más conveniente resolver de manera exacta que con heurísticas ya que el tiempo de ejecución es despreciable. Para lograr ese objetivo realizamos todas las combinaciones posibles.

Si bien el intercambio interfamilia (tanto para uno o muchos vehículos) tiene buen rendimiento (sección 5) no agregamos más transformaciones de este tipo porque creemos que el hecho de tener muchas hormigas iterando gran cantidad de veces esta cubriendo la funcionalidad de intentar encontrar rutas con distintos subconjuntos de clientes de cada familia.

5. RESULTADOS COMPUTACIONALES

5.1. Introducción

A lo largo de esta sección presentaremos los resultados obtenidos aplicando los algoritmos sobre diversas instancias, y analizaremos el comportamiento observado. El problema FTSP cuenta con instancias y resultados conocidos que serán utilizados para comparar nuestro algoritmo con los publicados en [5]. El problema FVRP no cuenta con resultados conocidos hasta la actualidad por lo tanto compararemos nuestro algoritmo contra otros utilizando instancias del problema GVRP. Luego presentaremos el método utilizado para crear nuevas instancias de FVRP y realizaremos la evaluación correspondiente.

A continuación presentaremos la metodología y resultados obtenidos en la búsqueda de los mejores parámetros. En esta variante de colonia de hormigas tenemos que tener en cuenta parámetros clásicos de los métodos heurísticos como cantidad de iteraciones máximas, o períodos de tiempo máximos; variables clásicas aplicadas en las búsquedas locales como por ejemplo la cantidad de vecinos a visitar; y también variables más específicas como el factor de evaporación, entre otros.

5.2. Determinación de parámetros

Como mencionábamos anteriormente, hay muchos parámetros a tener en cuenta. Procederemos a enumerarlos y explicar brevemente que representan dentro del algoritmo.

1. Factor de evaporación: determina la velocidad con la que se evapora la feromona depositada en los ejes.
2. Factor de mejora: determina la cantidad de feromona a depositar en cada eje de la mejor solución.
3. Factor de desmejora: determina la cantidad de feromona a retirar de cada eje de la peor solución.
4. Proporción de mutación normal: determina cuán probable es que un eje sufra alteraciones en su feromona al azar.
5. Proporción de mutación fuerte: determina cuán probable es que un eje sufra alteraciones en su feromona al azar, en etapas avanzadas del algoritmo donde tiende a estancarse en mínimos locales.
6. α: determina cuán probable es que se seleccione un eje según su cantidad de feromona. A mayor α, se toma más en cuenta aquél valor.
7. β: determina cuán probable es que se seleccione un eje según su información heurística, en este caso es proporcional al costo de recorrer el eje. A mayor β, se toman más en cuenta los ejes de menor costo.

8. Proporción de hormigas: proporción de hormigas en base a la cantidad de nodos de la instancia.

9. Cantidad mínima de hormigas: cantidad mínima de hormigas, independientemente de la cantidad de nodos.

10. Cantidad máxima de hormigas: cantidad máxima de hormigas, independientemente de la cantidad de nodos.

11. Porcentaje de hormigas - búsqueda local: no conviene que a todas las hormigas se le aplique una búsqueda local por el tiempo de procesamiento que se requiere. Por eso ordenamos las hormigas según la calidad de sus soluciones, y nos quedamos con las mejores.

12. Cantidad máxima de hormigas - búsqueda local: tope máximo de la cantidad de hormigas a las que se le aplicará la búsqueda local.

13. Porcentaje de vecinos: restringiendo los movimientos entre nodos a un cierto porcentaje de vecinos (nodos ordenados por cercanía) obtenemos mejores resultados que al intentar movernos entre todos los nodos. En caso de no disponer de vecinos, los movimientos no tendrán restricciones.

14. Probabilidad de finalización temprana: aplica solo para el problema FVRP. Determina la probabilidad de que un vehículo retorne sin haber completado su capacidad al depósito.

15. Proporción de cantidad de iteraciones general: factor que multiplica a la cantidad de nodos, para obtener la cantidad de iteraciones sin mejora que se requieren para que se de por finalizado el programa.

16. Cantidad mínima de iteraciones general: cantidad mínima de iteraciones sin mejoras que debe haber para dar por finalizado el programa, independientemente de la cantidad de nodos.

17. Cantidad máxima de iteraciones general: cantidad máxima de iteraciones sin mejoras que debe haber para dar por finalizado el programa, independientemente de la cantidad de nodos.

18. Proporción de cantidad de iteraciones búsqueda local: factor que multiplica a la longitud de las soluciones, para obtener la máxima cantidad de iteraciones sin mejoras que debe haber en la búsqueda local para que se de por finalizada su ejecución.

19. Cantidad de reinicios de feromona: cantidad de limpiezas de la feromona depositada en los ejes que intentará realizar el programa.

20. Tiempo transcurrido sin mejoras para el reinicio de la feromona: período de tiempo máximo que debe transcurrir sin que ocurra ninguna mejora para reiniciar los valores de feromona depositada en los ejes.

21. Tiempo transcurrido sin mejora máximo: período de tiempo máximo que debe transcurrir sin ninguna mejora para dar por finalizado el algoritmo, independientemente de los reinicios de feromona.

5.2.1. Instancias seleccionadas para FTSP

Al ser un problema relativamente nuevo, no disponemos de gran variedad de instancias conocidas. Es por eso que utilizaremos para determinar los parámetros algunas instancias propuestas por Morán-Mirabal, Gonzalez-Velarde, y Resende [5]. Dichas instancias se encuentran agrupadas por familias. "Burma14", "bayg29" y "att48" son familias de instancias relativamente pequeñas, contienen 14, 29 y 48 nodos respectivamente. Según pruebas preliminares, en este tipo de instancias los parámetros seleccionados no influyen significativamente en el resultado ya que el algoritmo obtiene buenos resultados en comparación con los obtenidos en [5]. Por lo tanto utilizaremos una instancia de cada una de las siguientes familias: "bier127", "a280", "gr666" y "pr1002", que contienen 127, 280, 666, y 1002 nodos respectivamente. Las instancias serán: bier127_10_1001_1002, a280_20_1001_1002, gr666_30_1001_1003, pr1002_40_1001_1002. A partir de ahora cuando nos referiremos a cualquier instancia de la forma *nombre-número* estaremos hablando de la instancia con dicho nombre que finalice con el número indicado. Por ejemplo gr666-3 se refiere a gr666_30_1001_1003.

5.2.2. Criterio de evaluación

En [5] los autores obtuvieron las soluciones óptimas usando CPLEX [18] en las instancias de hasta 48 nodos. Para instancias de 127 nodos y en una de 280 nodos utilizaron CPLEX pero retornando la mejor solución encontrada (que no es la óptima) hasta cierta cantidad de tiempo. Para el resto de las instancias compararon sus resultados con el mejor valor proveniente de las ejecuciones de los algoritmos BRKGA y GRASP+evPR con una duración de 36.000 segundos. En las pruebas que realizaremos vamos a determinar la calidad de nuestras soluciones utilizando una métrica que nos permita comparar objetivamente si una solución es mejor que otra. Tomando el mejor valor conocido hasta el momento proveniente de los métodos GRASP+evPR, BRKGA, CPLEX (métodos utilizados en [5]), utilizaremos como medida el error relativo. Este se define de la siguiente manera:

$$E_r = \frac{Mejor\ Valor\ Conocido - Resultado}{Resultado}$$

Esta misma métrica la utilizaremos en todas las ocasiones en las que será necesario evaluar la calidad de una solución obtenida. Solo en el caso de la instancia *bier127_10_1001_1002* emplearemos como mejor valor un resultado obtenido por el método que estamos desarrollando, ya que es menor al publicado con los demás métodos.

El algoritmo está implementado en C#, compilado con el Visual Studio 2012. Todas las pruebas serán realizadas sobre una PC Toshiba Satellite L555, de procesador Intel Core i5-430M de 2.27 GHz, con 4 GB de RAM y Windows 7 de 64-bits como sistema operativo.

En la sección actual se realizan evaluaciones sobre distintos parámetros. Cada uno de los valores obtenidos resultan de promediar los resultados de 5 ejecuciones de cada una de las instancias mencionadas previamente salvo que se indique lo contrario. Con respecto al resto de los parámetros el objetivo fue que estos colaboren aumentando la influencia de los que se está evaluando. En los casos del factor de evaporación, factores de mejora y

desmejora y factores de mutación (normal y fuerte) se utilizó un α de 0,8 y un β de 0,2 ya que el hecho de que las hormigas decidan su movimiento basándose fuertemente en los valores de la feromona colabora a la evaluación de dichos parámetros. Para los parámetros relacionados con cantidad de hormigas (en total, y en búsqueda local), condiciones de reinicio y finalización temprana de vehículos los valores de α y β fueron de 1 y 3 respectivamente.

El resto de los parámetros se mantuvieron para todas las pruebas realizadas con los siguientes valores: factor de evaporación - 0,4, factor de mejora y de desmejora - 1, proporción de mutación normal y fuerte - 0.1 y 0,5 respectivamente, proporción de hormigas - 0,5, cantidad mínima de hormigas - 10, cantidad máxima de hormigas - 75, porcentaje de hormigas en la búsqueda local - 0,5, cantidad máxima de hormigas en la búsqueda local - 15, porcentaje de vecinos - 0,25, proporción en la cantidad de iteraciones - 2, cantidad mínima y máxima en cantidad de iteraciones - 600 y 100 respectivamente, proporción de cantidad de iteraciones en búsqueda local - 2, cantidad de reinicios - 1, tiempo transcurrido sin mejoras para el reinicio de la feromona - 300 seg. y tiempo transcurrido sin mejora máximo - 420 seg.

5.2.3. Factor de evaporación

Se utiliza al finalizar cada iteración y su valor determina en que porcentaje se reduce la feromona de cada eje. A mayor factor de evaporación, menor será el valor de la feromona luego de cada iteración, provocando que el algoritmo converja más lentamente. Un valor demasiado alto provocará que se tengan menos en cuenta los resultados de las mejores hormigas obtenidos en iteraciones anteriores, mientras que un valor demasiado bajo provocará que se conserven niveles altos de feromona evitando que se destaquen las mejores soluciones. Se evaluó este parámetro sobre los factores 0.0, 0.2, 0.3, 0.4, 0.6, 0.8 y 1.

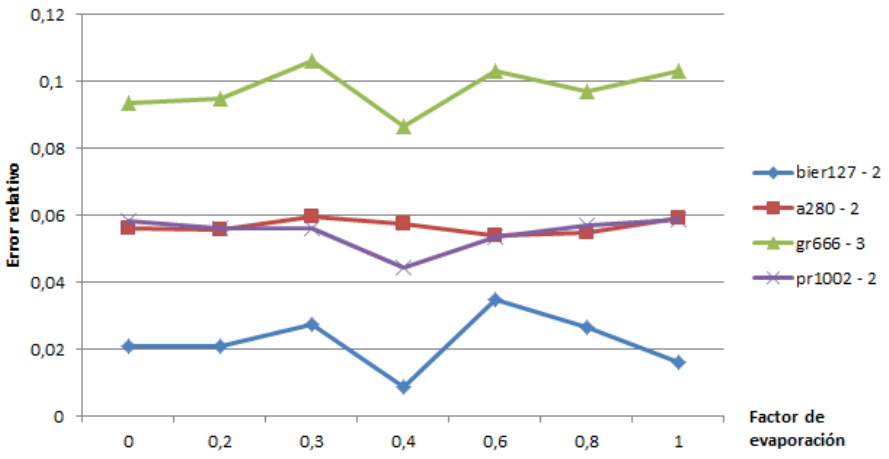

Fig. 5.1: Evaporación: error relativo de las instancias seleccionadas para los valores 0.0, 0.2, 0.3, 0.4, 0.6, 0.8 y 1.

Observando la figura 5.1 en todas las instancias notamos una tendencia favorable al factor de evaporación 0.4. Si observamos el gráfico de iteraciones (figura 5.2) notamos que hay una tendencia a disminuir la cantidad de las mismas a medida que el factor de

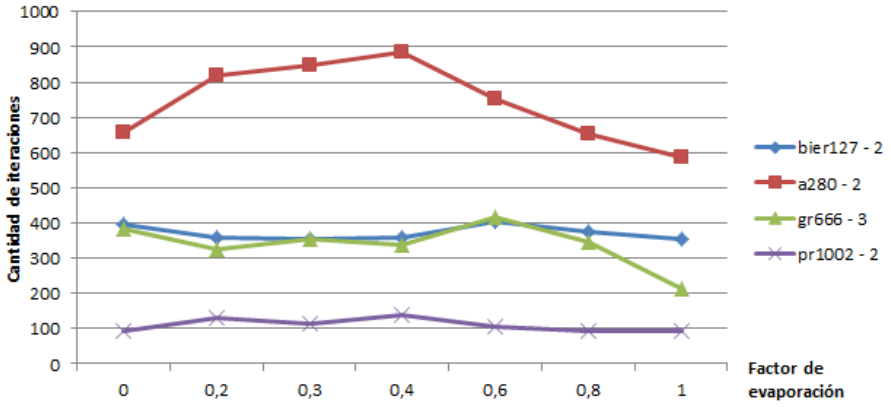

Fig. 5.2: Evaporación: cantidad de iteraciones de las instancias seleccionadas para los valores 0.0, 0.2, 0.3, 0.4, 0.6, 0.8 y 1.

evaporación se acerca a 1. Esto ocurre debido a que al haber tan poca feromona en las aristas las hormigas terminan basando su decisión en la información heurística que siempre es la misma, evitando la diversificación y estancando la búsqueda más rápidamente.

5.2.4. Factores de mejora y de desmejora

Según se detalló en la sección 4 al aumentar estos factores estamos aplicando más feromona a los ejes del mejor camino encontrado hasta el momento, y disminuyendo la del peor. Cuando el valor de los factores de mejora y desmejora son muy altos favorecemos la intensificación, ya que aumenta la probabilidad de que futuras generaciones de hormigas recorran el mejor camino. Por otro lado, valores bajos de dichos factores favorecen la diversificación, ya que futuras generaciones de hormigas no se verán tan atraídas por recorrer los mejores ejes encontrados. Las pruebas fueron realizadas sobre serie de combinaciones entre ambos factores. Se probaron todas las combinaciones entre los valores 0.5, 1, 1.5 y 2.

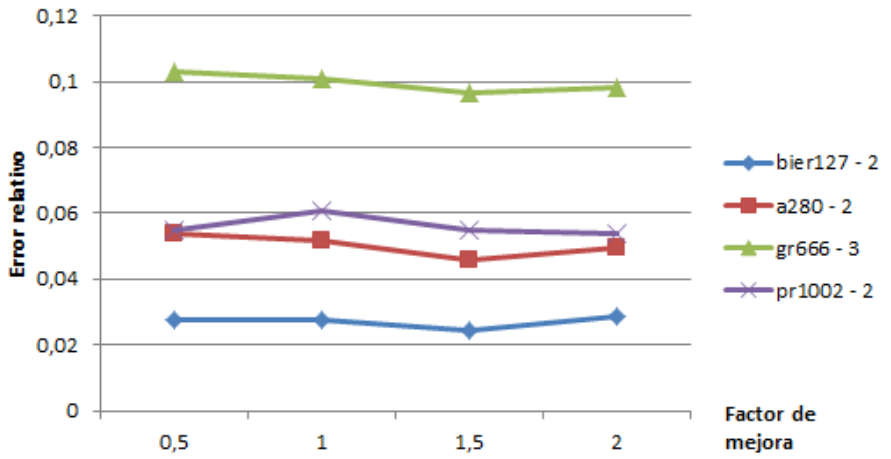

Fig. 5.3: Factor de mejora: promedio de los errores relativos tomando en cuenta los valores obtenidos con los distintos factores de desmejora.

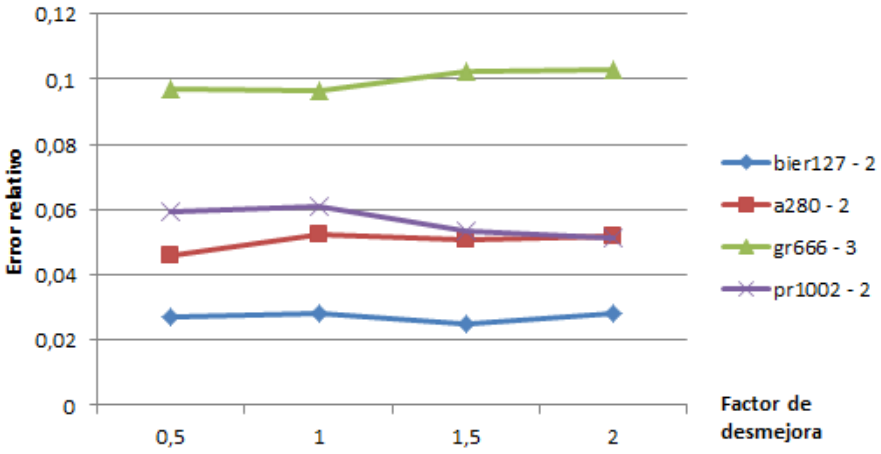

Fig. 5.4: Factor de desmejora: promedio de los errores relativos tomando en cuenta los valores obtenidos con los distintos factores de mejora.

En ningún gráfico se percibe grandes variaciones en la calidad del resultado obtenido. En cuanto al factor de mejora (figura 5.3) el valor 1.5 posee el mejor rendimiento. En cuanto al factor de desmejora (figura 5.4) es donde menos variaciones hay, de acá inferimos que no es demasiado determinante. A pesar de eso los valores bajos (0.5 y 1) poseen levemente un mejor rendimiento. Se evaluaron también la cantidad de iteraciones y en ningún caso resultaron determinantes, no se presentan los gráficos porque prácticamente no hay variaciones. El hecho de que el factor de desmejora no aporte gran impacto puede deberse a que la feromona de los ejes se inicializa con valor 0. Disminuir feromona en ejes con valor 0 no afecta a la búsqueda. Se realizaron pruebas con valores de inicialización más elevados aunque el hecho de que la feromona se evapore en cada iteración provoca que rápidamente los ejes que no aumenten el valor de su feromona queden con valores muy bajos por lo que tampoco se logra un gran cambio. Finalmente en base a las pruebas realizadas tomamos en cuenta que la mejor calidad de las soluciones se obtuvo con los valores 1.5 para el factor de mejora y 0.5 para el de desmejora.

5.2.5. Factores de mutación normal y fuerte

Para entender mejor estos valores recordemos que para aplicar la mutación realizamos un recorrido por cada nodo, seleccionamos un eje del vecindario al azar y la probabilidad de que variemos su feromona estará determinado por estos factores. La diferencia entre ambos es que el primero se aplica mientras que se encuentren mejoras en el algoritmo, mientras que el otro se aplica cuando la búsqueda se está estancando. Luego la probabilidad de mutación fuerte se comienza a aplicar cuando se alcanzó la mitad de la máxima cantidad de iteraciones sin mejora o bien ya ha transcurrido la mitad del tiempo máximo sin que se produzcan mejoras. La introducción de este cambio es producto de creer que al estancarse el algoritmo en un mínimo local debemos favorecer a la diversificación, depositando o retirando feromonas al azar con el fin de que futuras generaciones de hormigas recorran otros caminos. En pruebas preliminares el hecho de introducir este último parámetro produjo mejoras en los resultados.

Las primeras pruebas que presentaremos se realizaron modificando cualquier arista al azar, no solo perteneciente al vecindario sino a todos lo ejes sin utilizar la probabilidad de mutación fuerte. Los valores a evaluados fueron 0.0, 0.25, 0.5, 0.75 y 1.

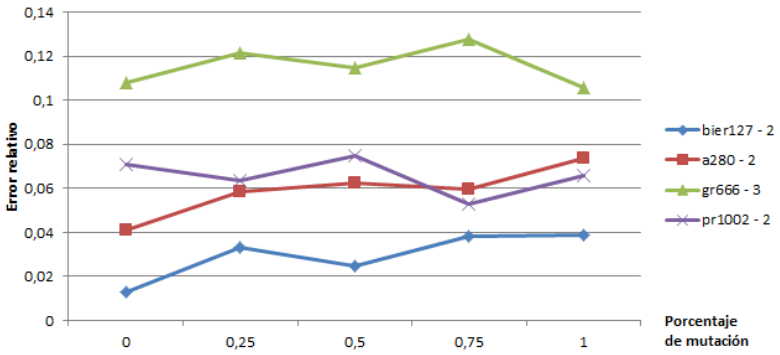

Fig. 5.5: Mutación normal: error relativo de las instancias seleccionadas para los valores 0.0, 0.25, 0.5, 0.75 y 1.

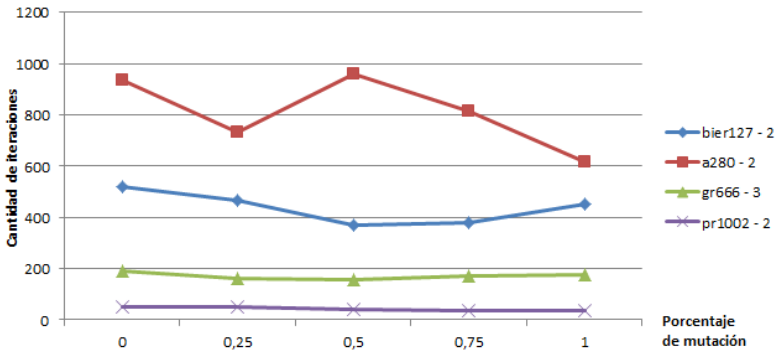

Fig. 5.6: Mutación normal: cantidad de iteraciones de las instancias seleccionadas para los valores 0.0, 0.25, 0.5, 0.75 y 1.

De las figuras 5.5 y 5.6 no podemos obtener buenas conclusiones ya que las pruebas arrojan resultados muy variados. La único que determinamos es que la mutación no causaba el efecto esperado. Luego cambiamos el algoritmo, realizando la modificación de la feromona solo para las aristas vecinas para los valores 0.0, 0.1, 0.25, 0.5, 0.75, y 1.

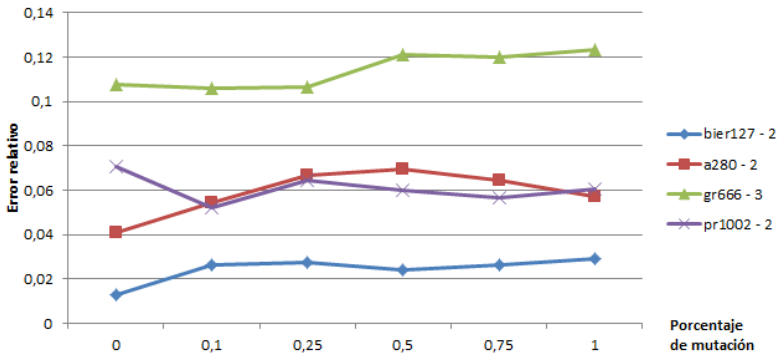

Fig. 5.7: Mutación normal en vecinos: error relativo de las instancias seleccionadas para los valores 0.0, 0.1, 0.25, 0.5, 0.75 y 1.

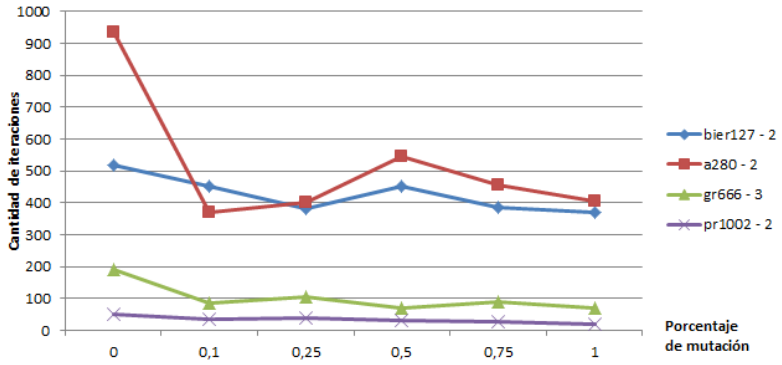

Fig. 5.8: Mutación normal en vecinos: cantidad de iteraciones de las instancias seleccionadas para los valores 0.0, 0.1, 0.25, 0.5, 0.75 y 1.

Realizando el cambio observamos que en las figuras 5.7 y 5.8 se percibe con mayor claridad el efecto de la mutación, especialmente en la cantidad de iteraciones efectuadas. Primero mencionamos que en el caso de la mayor instancia, el mejor resultado se encuentra con el valor 0.1. Por otro lado, si bien baja un poco la calidad de las soluciones de las instancias más chicas en comparación con no utilizar la mutación (valor 0.0), la cantidad de iteraciones necesarias para encontrar la mejor solución disminuye considerablemente. Esto puede deberse a que al utilizar levemente la mutación en las aristas de menor costo de cada nodo aumenta la diversificación acotada a un conjunto donde probablemente se encuentre el mejor valor.

Veamos ahora que sucede si utilizamos la probabilidad de mutación fuerte, manteniendo la probabilidad normal de mutación en 0.1. Los valores a emplear serán 0.1 (sin mutación fuerte), 0.2, 0.3, y 0.4.

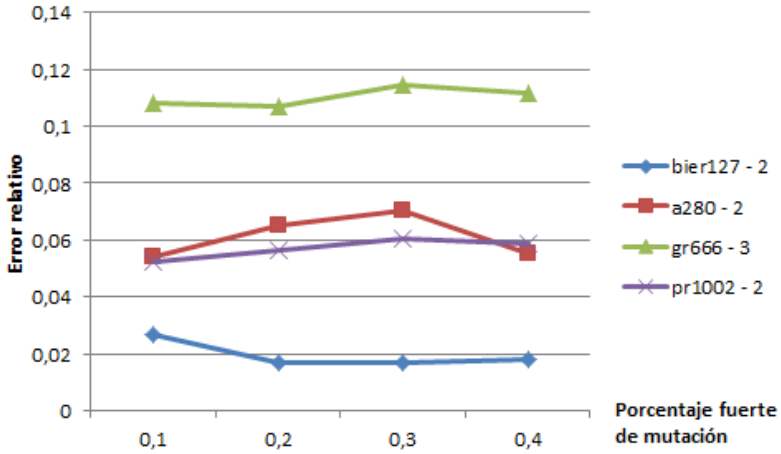

Fig. 5.9: Mutación fuerte en vecinos: error relativo de las instancias seleccionadas para los valores 0.1, 0.2, 0.3, 0.4. Valor de mutación normal: 0.1.

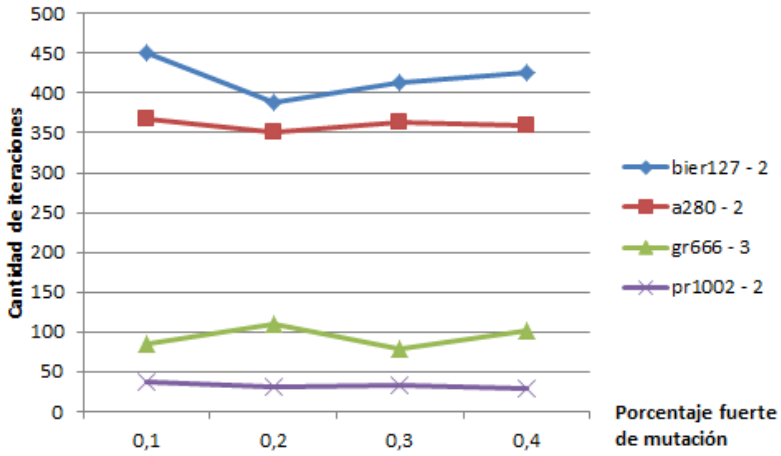

Fig. 5.10: Mutación fuerte en vecinos: cantidad de iteraciones de las instancias seleccionadas para los valores 0.1, 0.2, 0.3, 0.4. Valor de mutación normal: 0.1.

Si bien en las figuras 5.9 y 5.10 no se observan mejoras al utilizar este cambio, si hay una tendencia en disminuir la cantidad de iteraciones necesarias tomando el valor 0.2. La instancia bier127 es la que más se ve beneficiada de utilizar este valor, mientras que al resto le disminuye la calidad o bien no la altera significativamente. No se pueden obtener conclusiones definitivas sobre el aumento o no de la feromona en etapas avanzadas de la búsqueda. Por un leve mejor rendimiento nuestra decisión para las pruebas definitivas será utilizar la probabilidad de mutación fuerte con valor 0.2.

5.2.6. α y β

Estos valores representan los criterios que se utilizarán en la toma de decisiones de cada hormiga al momento de seleccionar a que nodo moverse. A mayor valor de α más se tomará en cuenta la información proveniente de la feromona. A mayor valor de β más se tomará en cuenta la información heurística que permanece constante a lo largo de la

ejecución del algoritmo y que está basada en la distancia entre los nodos. Estos valores deben evaluarse en conjunto, aumentando α y disminuyendo β o viceversa. Se realizarán pruebas para todas las combinaciones entre los valores 0.5, 1, 2 y 3 sobre los cuales ya se han realizado muchas pruebas en distintas aplicaciones [11].

	bier127 - 2	a280 - 2	gr666 - 3	pr1002 - 2
$\alpha = 0{,}5$; $\beta = 0.5$	0,02126	0,0495	0,1095	0,0534
$\alpha = 0{,}5$; $\beta = 1$	0,03153	0,0348	0,1134	0,0493
$\alpha = 0{,}5$; $\beta = 2$	0,0140	0,00368	0,0632	0,0011
$\alpha = 0{,}5$; $\beta = 3$	0,0065	0,0081	0,0373	0,0229
$\alpha = 1$; $\beta = 0{,}5$	0,0224	0,0454	0,1085	0,0618
$\alpha = 1$; $\beta = 1$	0,0225	0,0363	0,0891	0,0440
$\alpha = 1$; $\beta = 2$	0,0185	0,0203	0,0665	0,0319
$\alpha = 1$; $\beta = 3$	**0,0067**	**0,0004**	**0,0369**	0,0346
$\alpha = 2$; $\beta = 0{,}5$	0,0195	0,0368	0,0820	0,0551
$\alpha = 2$; $\beta = 1$	0,0173	0,0396	0,0951	0,0600
$\alpha = 2$; $\beta = 2$	0,0100	0,0396	0,0726	0,0378
$\alpha = 2$; $\beta = 3$	0,0151	0,0234	0,0415	0,0190
$\alpha = 3$; $\beta = 0{,}5$	0,0107	0,0485	0,0934	0,0375
$\alpha = 3$; $\beta = 1$	0,0238	0,0435	0,0604	0,0517
$\alpha = 3$; $\beta = 2$	0,0077	0,0265	0,0565	0,0446
$\alpha = 3$; $\beta = 3$	0,0181	0,0411	0,0390	**0,0065**

Tab. 5.1: α y β: promedio del error relativo de las instancias seleccionadas para las combinaciones entre los valores 0.5, 1, 2 y 3.

Según las pruebas realizadas presentadas en la tabla 5.1 la mejor combinación de α y β es 1 y 3 respectivamente. En 3 de las 4 instancias se encontró el menor promedio de error relativo y en la mayor instancia si bien el mejor α fue de 3, el mejor β también fue de 3 como en los casos anteriores. A partir de ahora consideraremos como mejor valor un α de 1 y un β de 3.

5.2.7. Hormigas

En principio se realizaron pruebas preliminares donde se buscaba encontrar que proporción de hormigas debía haber, en relación a la cantidad de nodos. Sin embargo, posteriormente sucedió que en instancias muy chicas se podría haber utilizado mayor cantidad de hormigas sin afectar el rendimiento del algoritmo, mientras que en instancias muy grandes demasiada cantidad de hormigas provocaba que en un mismo período se realizaran menos iteraciones, aprovechando menos las feromonas de las mejores hormigas y convergiendo más lentamente. Con el fin de tener un mejor panorama acerca del comportamiento del algoritmo según la cantidad de hormigas se realizaron distintas pruebas.

Por un lado probamos con aumentar la cantidad de hormigas, sin aumentar la cantidad de hormigas que realizan búsqueda local. Por otro lado mantuvimos una cantidad elevada de hormigas y variamos los valores de la cantidad de hormigas que si realizan búsqueda local. Los valores de la cantidad de hormigas evaluadas fueron: 10, 25, 50, 75, 100 y 150, con una cantidad máxima que realizaron búsqueda local de 20. Más adelante fijamos la cantidad de hormigas en 75 y variamos la cantidad que realizan búsqueda local en los valores 10, 20, 30, 40 y 50.

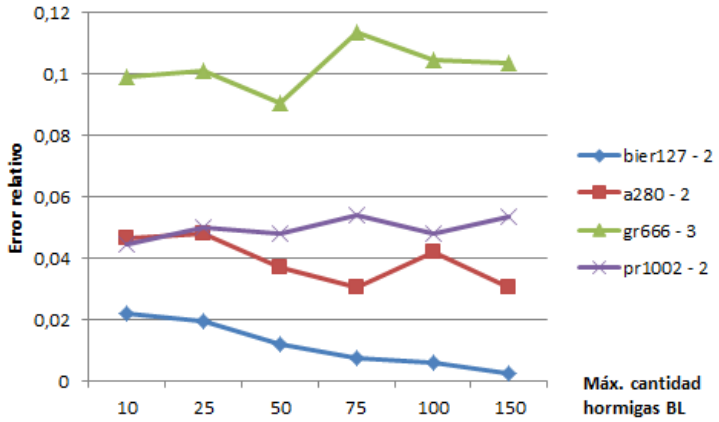

Fig. 5.11: Hormigas: error relativo de las instancias seleccionadas para los valores 10, 25, 50, 75, 100, 150. Máxima cantidad que realizan búsqueda local: 20.

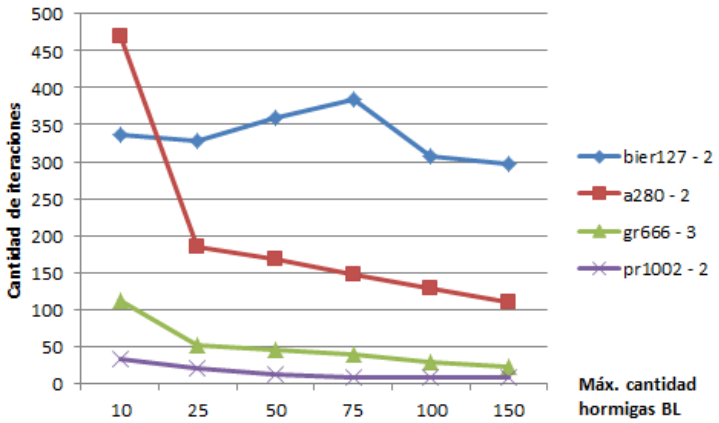

Fig. 5.12: Hormigas: promedio de cantidad de iteraciones para los valores 10, 25, 50, 75, 100, 150. Máxima cantidad que realizan búsqueda local: 20.

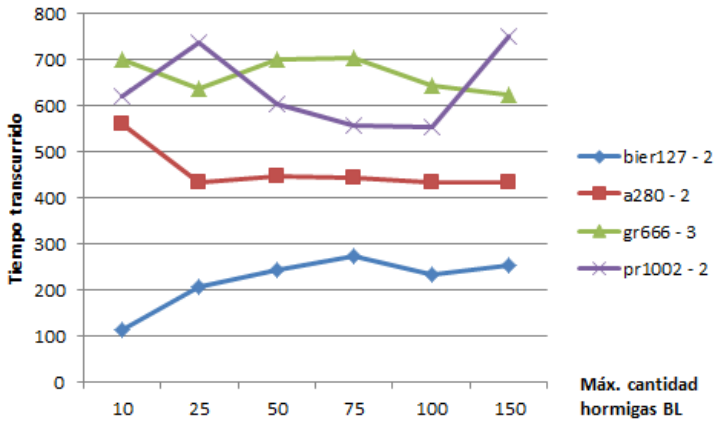

Fig. 5.13: Hormigas: promedio de tiempo de búsqueda para los valores 10, 25, 50, 75, 100, 150. Máxima cantidad que realizan búsqueda local: 20.

En base a los resultados presentados en las figuras 5.11, 5.12 y 5.13 podemos apreciar que si bien es evidente la mejora aumentando las hormigas en los casos donde la cantidad de nodos es menor como bier127 y a280, no es evidente en los casos más grandes como

gr666 y pr1002. Por otro lado la cantidad de iteraciones disminuye, pero no así la cantidad de tiempo empleado. Es decir que a mayor cantidad de hormigas más tiempo se consume por iteración, disminuyendo la cantidad total de iteraciones. Una conclusión que podemos obtener es que cuando la cantidad de nodos no es muy grande, podemos aumentar la cantidad de hormigas sin afectar significativamente el rendimiento. En cambio en instancias grandes, no necesariamente conviene aumentar la cantidad de hormigas tan rápidamente.

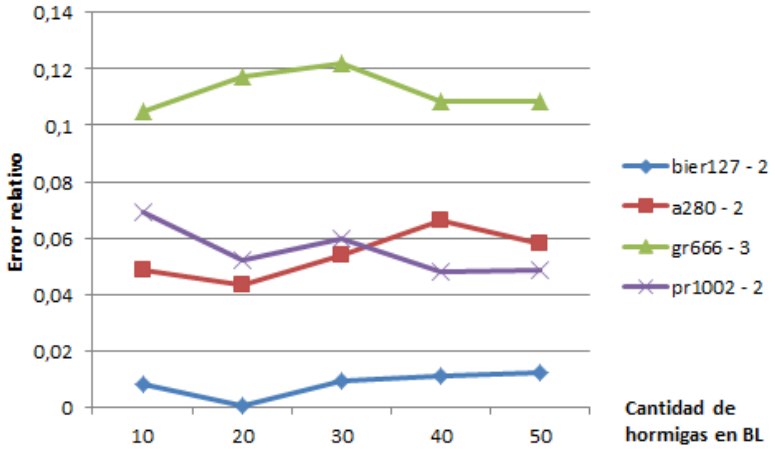

Fig. 5.14: Cantidad de hormigas en la búsqueda local: error relativo de las instancias seleccionadas para los valores 10, 20, 30, 40, 50. Cantidad de hormigas: 75.

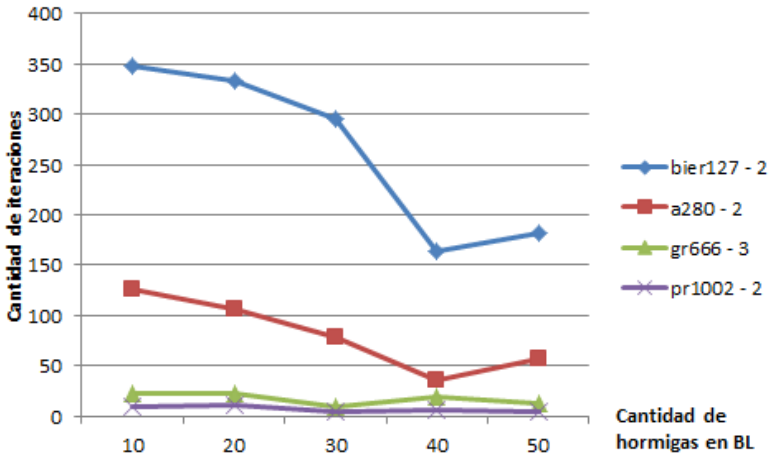

Fig. 5.15: Cantidad de hormigas en la búsqueda local: cantidad de iteraciones promedio de las instancias seleccionadas para los valores 10, 20, 30, 40, 50. Cantidad de hormigas: 75.

Teniendo en cuenta las figuras 5.14, 5.15 y 5.16 podemos notar un hecho importante: la cantidad de hormigas en la búsqueda local afecta significativamente el rendimiento de la colonia de hormigas. En primer lugar la instancia de 127 nodos obtuvo su mejor resultado con 20 hormigas, y disminuyó el rendimiento a medida que se agregaron más hormigas con búsqueda local. En instancias más grandes si bien mejoró el resultado, se empleó mucho más tiempo y disminuyó la cantidad de iteraciones. En definitiva disminuir la cantidad de iteraciones provoca que la colonia no evolucione. El enfoque que debemos emplear debe ser el de aumentar la cantidad de iteraciones, no la cantidad de hormigas en la búsqueda local.

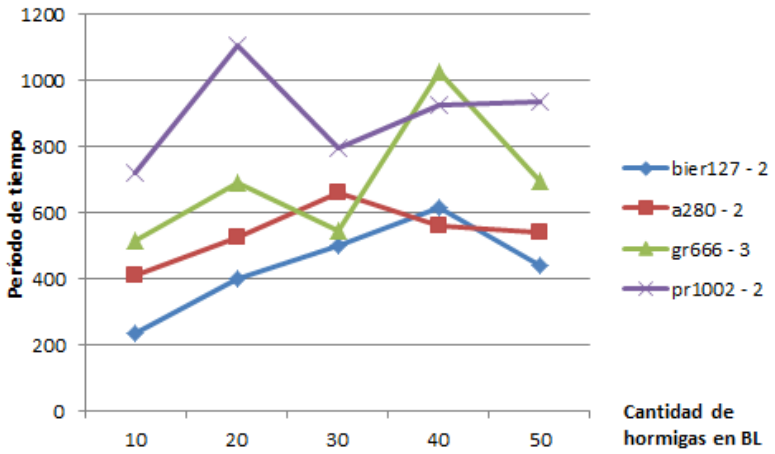

Fig. 5.16: Cantidad de hormigas en la búsqueda local: período de tiempo promedio de las instancias seleccionadas para los valores 10, 20, 30, 40, 50. Cantidad de hormigas: 75.

Procedemos a tomar como mejor valor una cantidad de 20 hormigas para dicha búsqueda.

5.2.8. Cantidad de vecinos

Cuando una hormiga decide realizar un movimiento tiene que evaluar las posibles alternativas basándose en la información heurística y la cantidad de feromona. Según pruebas preliminares al reducir las alternativas de sus movimientos a un cierto porcentaje de vecinos ayudamos a que la convergencia se produzca hacia un mejor resultado y más rápidamente. En aquellas pruebas preliminares el hecho de introducir el concepto de vecindario utilizando un 25 % de los nodos como vecinos produjo mejoras significativas. Las pruebas fueron realizadas para los valores 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.55 y 0.6.

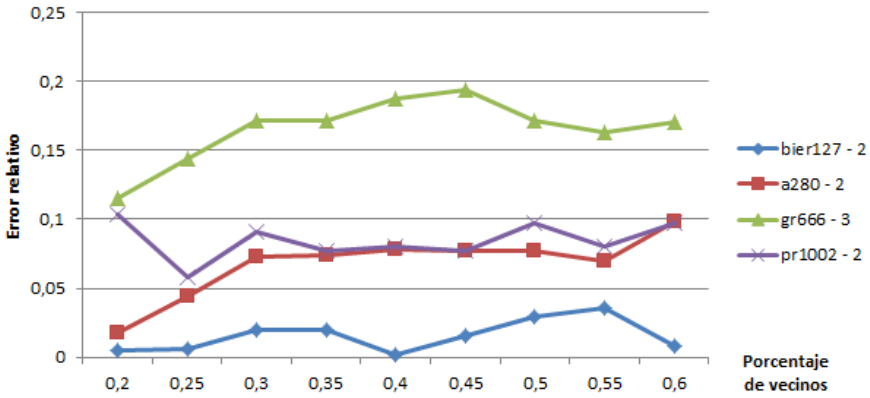

Fig. 5.17: Calidad de las soluciones obtenidas en base al tamaño del vecindario.

Como podemos observar en la figura 5.17 considerando vecindarios más acotados que ronden entre 20 % y 25 % logramos mejores resultados.

5.2.9. Búsqueda local (BL)

La búsqueda local utilizada en este algoritmo es parte fundamental en la calidad del resultado. Según diversas pruebas realizadas el tiempo empleado en la búsqueda local va desde el 50 % hasta el 95 % del tiempo total del algoritmo (figura 5.18). Por eso cualquier mejora en el rendimiento de esta funcionalidad afecta significativamente al resto.

La búsqueda local se puede aplicar a todas las hormigas o solo a las mejores. Un factor importante a tener en cuenta es la cantidad de hormigas que participarán en esta búsqueda. El parámetro *porcentaje de hormigas - búsqueda local* representa la proporción de hormigas que se verán involucradas. En instancias chicas podríamos realizar esta operación en todas las hormigas, pero en instancias grandes hay que tener mucho cuidado ya que una cantidad excesiva consumiría tanto tiempo que podría ocurrir que al no haber mejoras en un período prolongado, el algoritmo termine realizando pocas iteraciones.
Otro factor importante es el hecho de recorrer todo el espacio de búsqueda o bien seleccionar al azar distintos movimientos para realizar en cada búsqueda local. Si no recorremos todo el espacio, debemos definir que cantidad de iteraciones sin que se produzcan mejoras aceptamos como máximo. En este caso, dicha cantidad será proporcional a la longitud del circuito a optimizar. En este último caso, disponemos del parámetro *proporción de cantidad de iteraciones búsqueda local*.

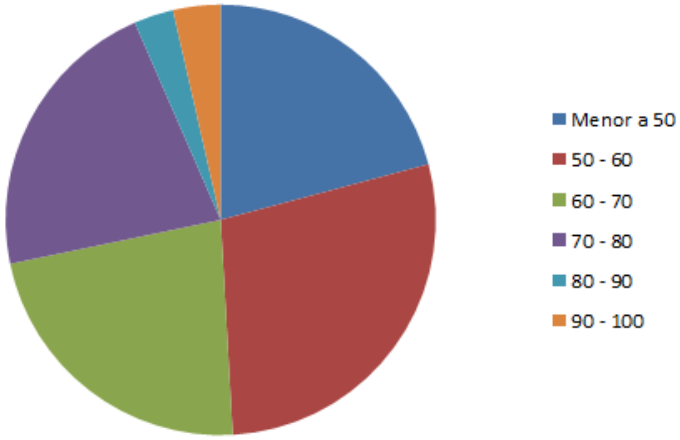

Fig. 5.18: Distribución del porcentaje de tiempo empleado en búsqueda local sobre el total de la ejecución en una única corrida de todas las instancias presentadas en [5].

5.2.9.1 Porcentaje de hormigas utilizadas para la BL

Se realizaron distintas pruebas de rendimiento durante el desarrollo de este trabajo, y en ningún caso resultaba conveniente que se superaran las 25 hormigas para la búsqueda local. Es decir, si tenemos 25 hormigas podríamos aplicarle la búsqueda al 100 % de las mismas, pero si tenemos 100 hormigas no conviene que la búsqueda se aplique en más del 25 %. Este valor lo denominaremos *cantidad máxima de hormigas para realizar búsqueda local* y es otro parámetro del programa.

5.2.9.2 Cantidad de iteraciones en la BL

El límite máximo soportado en el cual se finalizará cada BL es proporcional a la longitud del camino a mejorar. Diversas pruebas realizadas demuestran que si la proporción supera 2 o 3 veces la longitud del camino el rendimiento se reduce considerablemente. Por este motivo fijamos el valor en 2.

5.2.9.3 Espacio explorado

Esta es la sección más importante dentro de la BL. Para evaluar el rendimiento de cada algoritmo utilizamos dos contadores, uno para determinar la cantidad de veces que se intenta realizar un movimiento, y otro para determinar la cantidad de veces que se producen mejoras. A continuación veamos los datos obtenidos en distintas ejecuciones que se realizaron sobre las instancias gr666_30_1001_1002 (gr666-2) y gr666_30_1001_1003 (gr666-3).

Instancia	Reubicación	Intercambio	Or-Opt	Inter. de caminos	2-Opt	Inter. Familia
gr666-2	26.458 / 1.597.569	3.458 / 1.597.569	1.569 / 1.193.447	143 / 751.635	32.092 / 5.648.476	37.830 / 1.893.481
gr666-2	193.954 / 2.629.322	8.897 / 2.629.322	2.874 / 1.600.345	136 / 582.159	72.015 / 9.703.811	69.211 / 1.912.277
gr666-2	148.333 / 3.192.368	9.918 / 3.192.368	4.151 / 2.175.002	143 / 675.821	41.214 / 11.972.491	80.510 / 2.608.268
gr666-3	199.494 / 4.891.649	14.769 / 4.891.649	6.857 / 3.476.721	260 / 1.187.873	139.606 / 19.291.032	120.971 / 4.586.540
gr666-3	315.697 / 4.506.731	15.724 / 4.506.731	4.806 / 2.651.880	224 / 946.523	118.701 / 15.669.758	113.227 / 3.443.686
gr666-3	23.412 / 1.298.975	2.788 / 1.298.975	1.250 / 987.815	73 / 368.720	26.965 / 4.716.149	32.355 / 1.738.130

Tab. 5.2: Mejoras sobre total de la cantidad de iteraciones. Los datos corresponden a 3 ejecuciones sobre las instancias gr666-2 y gr666-3.

Como se puede observar en la tabla 5.2 la mayor parte de las mejoras las realizan reubicación, 2-opt y el intercambio de familia, mientras que intercambio de caminos se encuentra en el último lugar. Como la complejidad de recorrer todo el espacio en intercambio de caminos es de $O(n^3)$, solo lo vamos a realizar cuando la cantidad de nodos no sea muy grande. Según distintas pruebas realizadas recorriendo todo el espacio en instancias con soluciones menores a 64 nodos no perdemos tanto rendimiento, sin embargo en instancias mayores seleccionaremos cambios al azar hasta que la cantidad de iteraciones sin mejora no supere el tope descripto en la sección 4. No obstante en todos los casos la búsqueda *intercambio de nodos en familia* explorará todo el espacio ya que la relación entre el costo computacional y el beneficio obtenido es muy bueno.

Puede ocurrir que el intercambio de caminos se encuentre solapado por el intercambio de nodos o simplemente no sea muy efectivo en estos casos pero de cualquier forma como la complejidad es muy alta y no nos produce un beneficio acorde en cuanto a las mejoras que aporta realizaremos la siguiente modificación: evitamos ejecutar esta transformación y exploraremos todo el espacio en los algoritmos reubicación, intercambio, 2-opt (los mejores algoritmos) con una complejidad menor del orden de $O(n^2)$. El resto de las transformaciones tendrán el tope descripto anteriormente. A continuación se muestra una comparación de los resultados producto del cambio, tanto de la calidad de la solución obtenida como del tiempo de ejecución para la instancia gr666-2 en 2 corridas del proceso.

Costo	T. (s)	Reubicación	Intercambio	Or-Opt	Inter. de caminos	2-Opt	Inter. Familia
1.757,92	402	193.954 / 2.629.322	8.897 / 2.629.322	2.874 / 1.600.345	136 / 582.159	72.015 / 9.703.811	69.211 / 1.912.277
1.718,94	636	148.333 / 3.192.368	9.918 / 3.192.368	4.151 / 2.175.002	143 / 675.821	41.214 / 11.972.491	80.510 / 2.608.268

Tab. 5.3: Resultados anteriores al cambio de la exploración del espacio de búsqueda.

Costo	T. (s)	Reubicación	Intercambio	Or-Opt	Inter. de caminos	2-Opt	Inter. Familia
1.372,97	1457	123.024 / 1.174.440	505 / 1.174.440	211 / 1.096.619	0 / 0	44.060 / 157.669.774	146.283 / 5.931.086
1.374,97	754	57.889 / 544.158	234 / 544.158	115 / 512.930	0 / 0	20.028 / 5.759.610	20.309 / 72.770.059

Tab. 5.4: Resultados posteriores al cambio de la exploración del espacio de búsqueda.

Como podemos observar en las tablas 5.3 y 5.4 el costo computacional es significativamente menor en los casos posteriores al cambio en el espacio de búsqueda, aunque tenemos un aumento de hasta 100 % del tiempo total de la ejecución. En este trabajo tomamos la decisión de mantener esta exploración teniendo en cuenta que priorizamos la calidad de los resultados por sobre el tiempo de ejecución.

5.2.10. Cantidad de iteraciones y tiempos de ejecución

La cantidad de iteraciones a realizar es fundamental para aprovechar al máximo la feromona depositada por las mejores soluciones encontradas. Cuando transcurre un determinado período de tiempo o bien se alcanza la máxima cantidad de iteraciones sin mejora se realiza un reinicio de feromona, pudiendo converger hacia una mejor solución que en el caso anterior. Una vez alcanzada la cantidad de reinicios de feromona pasados como parámetro o al superarse el período máximo sin registrar mejoras se da por finalizada la ejecución del programa. Cuando el tiempo transcurrido sin mejoras para el reinicio de la feromona se completa el algoritmo se comporta como en su estado inicial. Un tiempo prolongado para provocar este reinicio sería innecesario ya que podría no producirse ninguna mejora, mientras que un período muy corto podría evitar el hallazgo de mejores soluciones.

Los valores de los parámetros mencionados deberán buscarse teniendo en cuenta en que iteraciones se producen mejoras y en que momento deja de haberlas. Parte de los datos que se obtienen de la ejecución del programa nos indican aquellos momentos de mejora. A continuación presentaremos varios momentos de mejora en distintas corridas para las instancias en las que estuvimos trabajando, con los siguientes parámetros:

1. *Proporción de cantidad de iteraciones sin mejora aceptadas sobre el total de nodos*: 2. Por ej. para 127 nodos dicha cantidad es de 254.

2. *Tiempo máximo sin mejoras aceptado*: 12 minutos.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	110.448,54	0	0,12
110.448,54	104.659,41	1	0,23
104.659,41	98.633,24	2	0,36

Tab. 5.5: bier127 - 2: primera ejecución. Cantidad total de iteraciones: 257, tiempo transcurrido: 28 seg.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	104.126,16	0	0,11
104.126,16	102.978,75	1	0,22
102.978,75	101.138,09	12	1,44
101.138,09	100.562,97	15	1,78
100.562,97	99.703,54	19	2,20
99.703,54	99.630,09	29	3,31
99.630,09	98.553,17	64	7,15
98.553,17	98.514,27	97	10,71
98.514,27	98.410,21	146	15,97
98.410,21	97.416,33	189	20,80
97.416,33	95.675,80	351	38,60

Tab. 5.6: bier127 - 2: segunda ejecución. Cantidad total de iteraciones: 606, tiempo transcurrido: 66 seg.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	101.621,62	0	0,10
101.621,62	99.389,37	15	1,79
99.389,37	98.450,92	113	12,54
98.450,92	97.962,00	145	16,01
97.962,00	96.623,22	228	25,33

Tab. 5.7: bier127 - 2: tercera ejecución. Cantidad total de iteraciones: 483, tiempo transcurrido: 53 seg.

Analicemos las tablas 5.5, 5.6 y 5.7. En el primer caso nos encontramos rápidamente con un buen camino, luego, producto de la feromona depositada en la iteración 0, se logran mejoras en las iteraciones 1 y 2. Lamentablemente luego de haber mejorado 3 veces consecutivas en soluciones vecinas, no logramos evitar caer en el mínimo local. En el segundo caso encontramos mejoras hasta la iteración 351 por lo que podemos afirmar que en este caso logramos aprovechar la alta cantidad de iteraciones. Por último en el tercer caso tenemos un intermedio donde se lograron mejoras hasta la iteración 228, pero no se pudo seguir mejorando. En principio pareciera que la cantidad de iteraciones y el tiempo extra que se dejó correr el algoritmo a partir de la última mejora alcanzarían para determinar que efectivamente se había alcanzado un buen valor.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	1.695,16	0	2,72
1.695,16	1.621,60	3	11,26
1.621,60	1.459,48	16	47,13

Tab. 5.8: gr666 - 2: primera ejecución. Cantidad total de iteraciones: 252, tiempo transcurrido: 797 seg.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	1.594,11	0	2,79
1.594,11	1.588,10	12	36,15
1.588,10	1.573,56	33	108,70
1.573,56	1.535,66	70	223,26
1.535,66	1.522,35	196	618,72
1.522,35	1.494,38	387	1224,38

Tab. 5.9: gr666 - 2: segunda ejecución. Cantidad total de iteraciones: 627, tiempo transcurrido: 1977 seg.

Costo Anterior	Nuevo Costo	Nro. de iteración	Segundos transcurridos
-	1.705,74	0	2,76
1.705,74	1.626,24	1	5,60
1.626,24	1.501,03	8	25,10
1.501,03	1.482,99	68	215,48

Tab. 5.10: gr666 - 2: tercera ejecución. Cantidad total de iteraciones: 308, tiempo transcurrido: 966 seg.

Con respecto a las tablas 5.8, 5.9 y 5.10 tenemos el primer caso, donde tanto la cantidad de iteraciones como el tiempo no lograron que se obtengan mejores resultados que en la iteración 16. Luego en el segundo caso tuvieron que pasar 10 minutos y 191 iteraciones para que se produzca la última mejora. Por último en el tercer caso el proceso encontró el mejor resultado en el segundo 215, 190 segundos después de haber encontrado el anterior. Las conclusiones que surgen a raíz de estos datos es que 10 minutos sin que se produzcan mejoras es un tiempo excesivo, ya que es muy poco habitual que se logren mejores resultados después de transcurrido ese tiempo. 5 minutos pareciera ser una mejor medida, luego podríamos limpiar la feromona y reiniciar el proceso. Es más probable que se encuentren mejores resultados en 2 ejecuciones de 5 minutos que en una de 10.

5.3. Resultados para el FTSP

Las instancias que presentamos provienen del trabajo publicado a comienzos de este año en [5]. En este trabajo realizamos la ejecución del algoritmo 5 veces en todas los casos. Con respecto a la tabla 5.11 la columna *objetivo* corresponde a resultados óptimos provenientes de ejecuciones de CPLEX en las instancias de hasta 48 nodos. Para el resto de las instancias utilizamos el mejor valor conocido hasta la actualidad. En la columna *Dif. 1* tomamos el mejor promedio de ambos algoritmos y lo comparamos con nuestro promedio. En la columna *Dif. 2* comparamos el mejor valor conocido hasta el momento (el mínimo entre *objetivo*, BRKGA, y GRASP+PR) con el mejor valor obtenido en nuestra ejecución. Por último mostramos el tiempo promedio en segundos de las búsquedas realizadas. Los parámetros utilizados fueron:

1. Factor de evaporación: 0,4.
2. Factor de mejora: 1,5.
3. Factor de desmejora: 0,5.
4. Probabilidad de mutación normal: 0,1.
5. Probabilidad de mutación fuerte: 0,2.
6. α: 1.
7. β: 3.
8. Proporción de hormigas: 0,3.
9. Cantidad mínima de hormigas: 20.
10. Cantidad máxima de hormigas: 100.
11. Porcentaje de hormigas - Búsqueda local: 0,5.
12. Cantidad máxima de hormigas - Búsqueda local: 20.
13. Porcentaje de vecinos: 0,25.
14. Proporción de cantidad de iteraciones general: 2.
15. Cantidad mínima de iteraciones general: 600.
16. Cantidad máxima de iteraciones general: 1000.
17. Proporción de cantidad de iteraciones búsqueda local: 2.
18. Cantidad de reinicios para feromona: 5.
19. Tiempo transcurrido sin mejoras para el reinicio de la feromona (en segundos): 420.
20. Tiempo transcurrido sin mejora máximo (en segundos): 800.
21. Algoritmos utilizados en la búsqueda local: reubicación y 2-opt exploran todo el espacio, intercambio, intercambio de caminos e intercambio en familia iteran hasta alcanzar 2 veces la longitud del camino sin que se produzcan mejoras. No realizamos or-opt ya que según los experimentos realizados no posee un buen rendimiento.

	Objetivo	BRKGA		GRASP+evPR		Colonia de hormigas - SMPH				
Nombre de instancia		Promedio	Mejor	Promedio	Mejor	Promedio	Dif. 1	Mejor	Dif. 2	T. (s)
burma14_3_1001_1001(*)	13,93	13,93	13,93	13,93	13,93	13,93	0,00	13,93 %	0,00 %	2
burma14_3_1001_1002(*)	25,66	25,66	25,66	25,66	25,66	25,66	0,00	25,66 %	0,00 %	5
burma14_3_1001_1003(*)	11,89	11,89	11,89	11,89	11,89	11,89	0,00	11,89 %	0,00 %	1
bayg29_4_1001_1001(*)	5345,86	5345,86	5345,86	5345,86	5345,86	5345,86	0,00 %	5345,86	0,00 %	22
bayg29_4_1001_1002(*)	5791,01	5791,01	5791,01	5791,01	5791,01	5791,01	0,00 %	5791,01	0,00 %	29
bayg29_4_1001_1003(*)	5544,33	5544,33	5544,33	5544,33	5544,33	5544,33	0,00 %	5544,33	0,00 %	28
att48_5_1001_1001(*)	23686,02	**23686,02**	**23686,02**	23709,62	**23686,02**	23954,54	1,13 %	23834,13	0,63 %	224
att48_5_1001_1002(*)	20609,09	**20609,09**	**20609,09**	20635,57	20635,57	21026,73	2,03 %	20679,09	0,34 %	92
att48_5_1001_1003(*)	9024,58	9024,58	9024,58	9024,58	9024,58	9024,58	0,00 %	9024,58	0,00 %	25
bier127_10_1001_1001	36800,39	36950,75	36913,74	36856,17	36800,39	**34794,32**	-5,59 %	**34500,98**	-6,25 %	1475
bier127_10_1001_1002	97615,41	98333,46	98216,10	98370,63	97615,41	**92531,88**	-5,90 %	**92135,29**	-5,61 %	334
bier127_10_1001_1003	50513,10	50891,36	50513,10	50920,77	50715,49	**48438,17**	-4,82 %	**48104,21**	-4,77 %	1107
a280_20_1001_1001	1891,16	2164,40	2126,34	1898,17	1891,16	**1834,60**	-3,35 %	**1807,16**	-4,44 %	2336
a280_20_1001_1002	1697,48	1980,84	1925,28	1702,33	1697,48	**1631,90**	-4,14 %	**1616,53**	-4,77 %	2060
a280_20_1001_1003	1597,25	1740,78	1720,23	1598,49	1597,25	**1496,19**	-6,40 %	**1479,74**	-7,36 %	1837
gr666_30_1001_1001	1817,06	2733,62	2625,69	1848,50	1817,06	**1678,66**	-9,19 %	**1670,47**	-8,07 %	2168
gr666_30_1001_1002	1443,05	2390,46	2275,80	1451,10	1443,05	**1372,69**	-5,40 %	**1370,13**	-5,05 %	1152
gr666_30_1001_1003	1358,12	3062,17	2426,59	1403,00	1384,18	**1315,40**	-6,24 %	**1292,71**	-4,82 %	2038
pr1002_40_1001_1001	163461,79	432665,72	421061,63	164721,66	163461,79	**150481,23**	-8,65 %	**149774,43**	-8,37 %	3247
pr1002_40_1001_1002	182144,13	445954	421761,00	184440,18	182144,13	**163014,26**	-11,62 %	**162302,50**	-10,89 %	2384
pr1002_40_1001_1003	149456,63	326561,61	284856,22	149838,13	149456,63	**138584,06**	-7,51 %	**138031,59**	-7,64 %	1561

Tab. 5.11: Resultados de las ejecuciones realizadas para todas las instancias presentadas de FTSP [5]. (*) El valor objetivo contiene resultados óptimos provenientes de ejecuciones de CPLEX en las instancias de hasta 48 nodos. Para el resto de las instancias utilizamos el mejor valor conocido hasta la actualidad.

En la tabla 5.11 podemos observar que el algoritmo responde con buenos resultados tanto en instancias medianas (127, 280 nodos) como en instancias grandes (666, 1002 nodos). El caso más llamativo ocurre en las instancias chicas de 48 nodos donde no se logró alcanzar el mejor resultado presentado previamente.

5.4. FVRP

Como FVRP es un problema que se define en este trabajo generaremos nuevas instancias para poder probar nuestro algoritmo. Para poder comparar con resultados previos utilizaremos instancias del problema generalizado de ruteo de vehículos (GVRP).

5.4.1. Generación de nuevas instancias de FVRP

Las instancias de FVRP fueron generadas a partir de tomar las instancias ya existentes de FTSP y agregando la capacidad del vehículo y la demanda por cada cliente. La capacidad del vehículo es de 1000 unidades y la demanda de cada cliente se distribuye de la siguiente manera:

1. 30 % de que la demanda de un cliente sea entre 1 y 100 unidades
2. 50 % de que la demanda de un cliente sea entre 100 y 600 unidades
3. 20 % de que la demanda de un cliente sea entre 600 y 1000 unidades

Como aclaramos en la definición del problema cualquier conjunto de nodos que visitemos en cada familia cumplirá con la demanda del grupo. Por eso consideramos la demanda de cada familia equivalente a la del menor conjunto de nodos que podamos visitar. Consideramos a la función $d'(F_l)$ a la demanda de la familia l. Luego:

$$d'(F_l) = min(\textstyle\sum_{v \in V'} d(v)) \;\; \forall V' \subseteq F_l, |V'| = nv_l \quad \forall F_l \in K.$$

5.4.2. Probabilidad de finalización temprana

En este problema utilizamos los mismos parámetros que en FTSP y además determinamos una probabilidad de finalización temprana que describiremos a continuación. Se utiliza cuando nos encontramos en un nodo cuyo vecindario ya se encuentra visitado. Como se detalló en la sección *aplicación*, si estamos ubicados en un lugar *cercano* al depósito o el nodo más cercano disponible se encuentra a mayor distancia del depósito, finalizaremos el recorrido con esta probabilidad. Las instancias de prueba fueron definidas como mencionamos previamente (sección 5.4.1). Por otro lado al no tener una cota inferior del problema presentaremos los valores obtenidos en esta etapa de pruebas para las instancias modificadas de bayg29, att48, y bier127 en lugar de los errores relativos.

	bayg29 - 1	bayg29 - 2	bayg29 - 3	att48 - 2	att48 - 1	att48 - 3	bier127 - 2	bier127 - 3	bier127 - 1
0	10.011,21	13.129,05	10.102,44	67.833,79	18.719,52	57.573,10	113.201,56	72.219,47	51.979,86
0,25	9.905,91	13.129,05	10.115,46	68.492,60	18.978,86	57.234,50	113.120,95	71.209,43	51.896,94
0,5	**9.808,18**	13.203,20	10.128,47	68.092,22	18.671,52	57.430,99	111.847,50	70.576,61	51.494,76
0,75	9.835,07	13.168,07	10.127,47	**67.280,90**	17.782,73	58.352,78	**111.267,19**	**70.392,49**	51.687,80
1	9.928,96	**13.128,05**	**10.102,11**	67.867,53	**17.763,96**	**56.874,75**	112.404,34	70.415,52	**51.395,43**

Tab. 5.12: Finalización temprana: resultados promedio obtenidos para los valores 0, 0.25, 0.5, 0.75 y 1.

	bayg29 - 1	bayg29 - 2	bayg29 - 3	att48 - 2	att48 - 1	att48 - 3	bier127 - 2	bier127 - 3	bier127 - 1
0	2,54	**1,62**	2,62	17,38	3,42	5,24	149,85	47,31	62,26
0,25	**2,15**	1,84	2,54	14,05	4,00	11,21	131,52	54,27	65,65
0,5	4,21	2,33	**2,51**	**9,10**	3,93	6,89	118,97	**47,22**	68,76
0,75	3,27	2,20	2,56	12,60	3,13	**4,49**	106,34	51,23	62,91
1	3,32	1,87	3,20	18,90	**2,50**	5,64	**90,65**	57,61	**48,74**

Tab. 5.13: Finalización temprana: tiempo transcurrido promedio para los valores 0, 0.25, 0.5, 0.75 y 1.

	bayg29 - 1	bayg29 - 2	bayg29 - 3	att48 - 2	att48 - 1	att48 - 3	bier127 - 2	bier127 - 3	bier127 - 1
0	4	6	6	10	4	10	25,2	18,2	19,2
0,25	4,2	6	6	10,2	4	10,6	26,2	19,6	19,4
0,5	4,2	**7,2**	6,2	**11**	**4,2**	10,8	26,2	19,4	20
0,75	**4,8**	7	7	**11**	**4,2**	**11,6**	26	19,8	**20,2**
1	4,2	7	6	**11**	**4,2**	11	**26,6**	**20**	**20,2**

Tab. 5.14: Finalización temprana: cantidad de vehículos utilizados promedio para los valores 0, 0.25, 0.5, 0.75 y 1.

En la tabla 5.12 podemos observar que los mejores resultados siempre se encuentran con un porcentaje de finalización temprana mayor o igual a 0.5. Más aún, en 4 de las 9 pruebas realizadas se obtiene como mejor resultado ingresando un 100 % de probabilidad de finalizar el recorrido prematuramente. En la tabla 5.13 los datos no reflejan que al modificar este parámetro se produzca un cambio en la duración del proceso. Las mejoras obtenidas se deben a que el algoritmo completa la capacidad de cada vehículo sin ser esta la mejor forma de minimizar distancias. Por lo tanto finalizando prematuramente agregamos vehículos al azar en las soluciones obtenidas. La tabla 5.14 muestra como en las instancias donde se utiliza mayor cantidad de vehículos se obtienen mejores resultados. En base a la calidad de las soluciones presentadas el valor a utilizar como parámetro en las ejecuciones que realizaremos más adelante debería ser un valor cercano ubicado entre el 85 % y el 100 %.

5.4.3. Resultados para el FVRP

Al ser este un problema nuevo y no disponer de resultados para comparar, utilizamos como objetivo el mejor valor obtenido en 5 ejecuciones de 1:30hs. para cada instancia. Luego realizamos 5 ejecuciones de a lo sumo 15-25 minutos y comparamos los resultados. Los parámetros son los mismos que los utilizados en FTSP, agregando el porcentaje de finalización temprana con valor 0.85.

En la tabla 5.15 se puede observar que la diferencia lograda por las ejecuciones de corta duración no supera el 2 %. Por otro lado en algunos casos como pr1002_40_1001_1003 se logra encontrar un mejor valor, aunque no muy lejano al objetivo.

	Objetivo	Colonia de hormigas - SMPH				
Nombre de instancia		Promedio	Dif.	Mejor	Dif.	Tiempo (s)
burma14_3_1001_1001	16,35	16,35	0,00 %	16,35	0,00 %	16
burma14_3_1001_1002	35,71	35,71	0,00 %	35,71	0,00 %	36
burma14_3_1001_1003	11,04	11,04	0,00 %	11,04	0,00 %	8
bayg29_4_1001_1001	9526,61	9550,40	0,25 %	9526,61	0,00 %	140
bayg29_4_1001_1002	13127,05	13128,05	0,01 %	13128,05	0,01 %	137
bayg29_4_1001_1003	10100,44	10101,11	0,01 %	10100,44	0,00 %	167
att48_5_1001_1001	66871,44	67122,93	0,38 %	66871,44	0,00 %	453
att48_5_1001_1002	56325,95	56686,07	0,64 %	56325,95	0,00 %	268
att48_5_1001_1003	17127,95	17128,04	0,001 %	17127,95	0,00 %	140
bier127_10_1001_1001	100878,28	100912,74	0,03 %	100779,87	-0,10 %	1335
bier127_10_1001_1002	217376,55	217638,36	0,12 %	216421,97	-0,44 %	966
bier127_10_1001_1003	137983,46	138455,36	0,34 %	138072,24	0,06 %	893
a280_20_1001_1001	15707,51	15874,69	1,06 %	15750,98	0,28 %	1308
a280_20_1001_1002	12912,01	13141,79	1,78 %	13093,27	1,40 %	995
a280_20_1001_1003	11867,62	12082,31	1,81 %	12022,72	1,31 %	1516
gr666_30_1001_1001	14847,04	15076,57	1,55 %	15039,83	1,30 %	1081
gr666_30_1001_1002	13333,87	13427,90	0,71 %	13300,78	-0,25 %	1016
gr666_30_1001_1003	13309,33	13304,34	-0,04 %	13273,73	-0,27 %	1164
pr1002_40_1001_1001	2060679,36	2078976,64	0,89 %	2061861,39	0,06 %	1415
pr1002_40_1001_1002	2936641,74	2960511,50	0,81 %	2952446,41	0,54 %	863
pr1002_40_1001_1003	2198170,55	2201729,33	0,16 %	2189537,48	-0,39 %	830

Tab. 5.15: Resultados de las ejecuciones realizadas para todas las instancias generadas de FVRP.

5.4.4. Ejecución de instancias de GVRP

El problema de ruteo de vehículos generalizado (GVRP) consiste en encontrar un conjunto de rutas de longitud mínima que pasen por distintos conjuntos de clientes, donde los recorridos incluyen exactamente un cliente de cada conjunto y satisfacen las limitaciones de capacidad [8]. Es un caso particular de FVRP. Bektas, Erdogan, Ropke [7] presentaron diversas instancias en agrupadas en distintas categorías. Las categorías A, B y P están constituidas desde 16 hasta 101 nodos mientras que las categorías G y M son las más grandes y contienen desde 101 hasta 262 nodos. Estas instancias fueron resueltas por un algoritmo de branch and cut y la metaheurística de búsqueda por entornos grandes (LNS por sus iniciales en inglés). Por otro lado Ha, Bostel, Langevin y Rousseau [8] presentaron una nueva formulación y resolvieron las mismas instancias con una metaheurística híbrida basada en GRASP y ELS. Para poder comparar la eficacia de nuestro algoritmo con los existentes ejecutaremos nuestra aplicación con dichas instancias y las compararemos con los resultados de las ejecuciones del branch and cut [7], LNS [7], y la metaheurística híbrida mencionada anteriormente [8].

Los valores de los parámetros escogidos son los mismos que en la resolución de FTSP, con el valor del parámetro **probabilidad de finalización temprana** en 0.85.

Con respecto a la segunda columna de la tabla 5.16 tomamos en cuenta el valor obtenido por el algoritmo de branch and cut en las instancias en las cuales finalizó la ejecución, mientras que en caso contrario se tomó en cuenta el mejor valor de las demás metaheurísticas. Se puede observar que nuestro algoritmo alcanza los mejores resultados en todas las instancias de tipo A, B y P. En las grandes instancias de tipo M se alcanza una efectividad bastante elevada de 4/8 aunque no se logró obtener el mismo resultado en las instancias aún más grandes de tipo G con 262 nodos. El detalle de los resultados se encuentran a continuación en la sección 5.4.5.

Tipo de instancia	OCH-SMPH alcanza el mejor valor publicado
A	54/54
B	46/46
P	48/48
M	4/8
G	0/2

Tab. 5.16: Resumen de los resultados obtenidos en base a las ejecuciones de las instancias presentadas en [7].

5.4.5. Detalle de los resultados computacionales de las ejecuciones de las instancias de GVRP

Los resultados de las instancias de GVRP que se encuentran a continuación fueron presentadas por Bektas, Erdogan, Ropke [7]. Las columnas m_1, m_2 y m_3 corresponden a la cantidad de vehículos utilizados en los algoritmos LNS y branch & cut, la metaheurística propuesta por Ha et. al., y OCH-SMPH respectivamente.

	Bektas et. al.			Ha et. al.			
Nombre de instancia	m_1	LNS	Branch & Cut	m_2	Metaheurística	m_3	OCH-SMPH
A-n32-k5-C11	2	386	386	2	386	2	386
A-n32-k5-C16	2	519	519	3	508	3	508
A-n33-k5-C11	2	318	315	2	315	2	315
A-n33-k5-C17	3	451	451	3	451	3	451
A-n33-k6-C11	2	370	370	2	370	2	370
A-n33-k6-C17	3	465	465	3	465	3	465
A-n34-k5-C12	2	419	419	2	419	2	419
A-n34-k5-C17	3	489	489	3	489	3	489
A-n36-k5-C12	2	396	396	2	396	2	396
A-n36-k5-C18	2	505	505	3	502	3	502
A-n37-k5-C13	2	347	347	2	347	2	347
A-n37-k5-C19	3	432	432	3	432	3	432
A-n37-k6-C13	2	431	431	2	431	2	431
A-n37-k6-C19	3	584	584	3	584	3	584
A-n38-k5-C13	2	367	367	2	367	2	367
A-n38-k5-C19	3	476	476	3	476	3	476
A-n39-k5-C13	2	364	364	2	364	2	364
A-n39-k5-C20	3	557	557	3	557	3	557
A-n39-k6-C13	2	403	403	2	403	2	403
A-n39-k6-C20	3	544	544	3	544	3	544
A-n44-k6-C15	2	503	503	3	491	3	491
A-n44-k6-C22	3	608	608	3	608	3	608
A-n45-k6-C15	3	474	474	3	474	3	474
A-n45-k6-C23	4	613	613	4	613	4	613
A-n45-k7-C15	3	475	475	3	475	3	475
A-n45-k7-C23	4	674	674	4	674	4	674
A-n46-k7-C16	3	462	462	3	462	3	462
A-n46-k7-C23	4	593	593	4	593	4	593
A-n48-k7-C16	3	451	451	3	451	3	451
A-n48-k7-C24	4	667	667	4	667	4	667
A-n53-k7-C18	3	440	440	3	440	3	440
A-n53-k7-C27	4	603	603	4	603	4	603
A-n54-k7-C18	3	482	482	3	482	3	482
A-n54-k7-C27	4	690	690	4	690	4	690
A-n55-k9-C19	3	473	473	3	473	3	473
A-n55-k9-C28	5	699	699	5	699	5	699
A-n60-k9-C20	3	595	595	3	595	3	595
A-n60-k9-C30	5	769	769	5	769	5	769
A-n61-k9-C21	4	473	473	4	473	4	473
A-n61-k9-C31	5	638	638	5	638	5	638
A-n62-k8-C21	3	596	596	3	596	3	596
A-n62-k8-C31	4	740	740	4	740	4	740
A-n63-k10-C21	4	593	593	4	593	4	593
A-n63-k10-C32	5	801	801	5	801	5	801
A-n63-k9-C21	3	642	-	3	642	3	642
A-n63-k9-C32	5	912	-	5	912	5	912
A-n64-k9-C22	3	536	536	3	536	3	536
A-n64-k9-C32	5	763	763	5	763	5	763
A-n65-k9-C22	3	500	500	3	500	3	500
A-n65-k9-C33	5	682	682	5	682	5	682
A-n69-k9-C23	3	520	520	3	520	3	520
A-n69-k9-C35	5	680	680	5	680	5	680
A-n80-k10-C27	4	710	-	4	710	4	710
A-n80-k10-C40	5	997	998	5	997	5	997

Tab. 5.17: Resultado de las ejecuciones realizadas sobre las instancias de tipo A presentadas en [7].

		Bektas et. al.			Ha et. al.		
Nombre de instancia	m_1	LNS	Branch & Cut	m_2	Metaheurística	m_3	OCH-SMPH
B-n31-k5-C11	2	356	356	2	356	2	356
B-n31-k5-C16	3	441	441	3	441	3	441
B-n34-k5-C12	2	369	369	2	369	2	369
B-n34-k5-C17	3	472	472	3	472	3	472
B-n35-k5-C12	2	501	501	2	501	2	501
B-n35-k5-C18	3	626	626	3	626	3	626
B-n38-k6-C13	2	370	370	2	370	2	370
B-n38-k6-C19	3	451	451	3	451	3	451
B-n39-k5-C13	2	280	280	2	280	2	280
B-n39-k5-C20	3	357	357	3	357	3	357
B-n41-k6-C14	2	407	407	2	407	2	407
B-n41-k6-C21	3	481	481	3	481	3	481
B-n43-k6-C15	2	343	343	2	343	2	343
B-n43-k6-C22	3	483	483	3	483	3	483
B-n44-k7-C15	3	395	395	3	395	3	395
B-n44-k7-C22	4	540	540	4	540	4	540
B-n45-k5-C15	2	422	410	2	410	2	410
B-n45-k5-C23	3	497	497	3	497	3	497
B-n45-k6-C15	2	336	336	2	336	2	336
B-n45-k6-C23	4	478	478	4	478	4	478
B-n50-k7-C17	3	393	393	3	393	3	393
B-n50-k7-C25	4	449	449	4	449	4	449
B-n50-k8-C17	3	598	598	3	598	3	598
B-n50-k8-C25	5	916	916	5	916	5	916
B-n51-k7-C17	3	511	511	3	511	3	511
B-n51-k7-C26	4	651	651	4	651	4	651
B-n52-k7-C18	3	359	359	3	359	3	359
B-n52-k7-C26	4	450	450	4	450	4	450
B-n56-k7-C19	3	356	356	3	356	3	356
B-n56-k7-C28	4	486	486	4	486	4	486
B-n57-k7-C19	3	558	558	3	558	3	558
B-n57-k7-C29	4	751	751	4	751	4	751
B-n57-k9-C19	3	681	681	3	681	3	681
B-n57-k9-C29	5	942	942	5	942	5	942
B-n63-k10-C21	3	599	599	3	599	3	599
B-n63-k10-C32	5	816	816	5	816	5	816
B-n64-k9-C22	4	452	452	4	452	4	452
B-n64-k9-C32	5	509	509	5	509	5	509
B-n66-k9-C22	3	609	609	3	609	3	609
B-n66-k9-C33	5	808	808	5	808	5	808
B-n67-k10-C23	4	558	558	4	558	4	558
B-n67-k10-C34	5	673	673	5	673	5	673
B-n68-k9-C23	3	523	523	3	523	3	523
B-n68-k9-C34	5	704	704	5	704	5	704
B-n78-k10-C26	4	606	606	4	606	4	606
B-n78-k10-C39	5	803	803	5	803	5	803

Tab. 5.18: Resultado de las ejecuciones realizadas sobre las instancias de tipo B presentadas en [7].

		Bektas et. al.			Ha et. al.		
Nombre de instancia	m_1	LNS	Branch & Cut	m_2	Metaheurística	m_3	OCH-SMPH
P-n16-k8-C6	4	170	170	4	170	4	170
P-n16-k8-C8	5	239	239	5	239	5	239
P-n19-k2-C10	2	147	147	2	147	2	147
P-n19-k2-C7	1	111	111	1	111	1	111
P-n20-k2-C10	2	154	154	2	154	2	154
P-n20-k2-C7	1	117	117	1	117	1	117
P-n21-k2-C11	2	160	160	2	160	2	160
P-n21-k2-C7	1	117	117	1	117	1	117
P-n22-k2-C11	2	162	162	2	162	2	162
P-n22-k2-C8	1	111	111	1	111	1	111
P-n22-k8-C11	5	314	314	5	314	5	314
P-n22-k8-C8	4	249	249	4	249	4	249
P-n23-k8-C12	5	312	312	5	312	5	312
P-n23-k8-C8	3	174	174	3	174	3	174
P-n40-k5-C14	2	213	213	2	213	2	213
P-n40-k5-C20	3	294	294	3	294	3	294
P-n45-k5-C15	2	238	238	2	238	2	238
P-n45-k5-C23	3	337	337	3	337	3	337
P-n50-k10-C17	4	292	292	4	292	4	292
P-n50-k10-C25	5	410	410	5	410	5	410
P-n50-k7-C17	3	261	261	3	261	3	261
P-n50-k7-C25	4	353	353	4	353	4	353
P-n50-k8-C17	3	262	262	3	262	3	262
P-n50-k8-C25	4	392	-	5	372	5	372
P-n51-k10-C17	4	309	309	4	309	4	309
P-n51-k10-C26	6	427	427	6	427	6	427
P-n55-k10-C19	4	301	301	4	301	4	301
P-n55-k10-C28	5	415	415	5	415	5	415
P-n55-k15-C19	6	378	378	6	378	6	378
P-n55-k15-C28	8	555	-	9	551	9	551
P-n55-k7-C19	3	271	271	3	271	3	271
P-n55-k7-C28	4	361	361	4	361	4	361
P-n55-k8-C19	3	274	274	3	274	3	274
P-n55-k8-C28	4	361	361	4	361	4	361
P-n60-k10-C20	4	325	325	4	325	4	325
P-n60-k10-C30	5	443	-	5	443	5	443
P-n60-k15-C20	5	382	-	6	374	6	374
P-n60-k15-C30	8	565	-	8	565	8	565
P-n65-k10-C22	4	372	372	4	372	4	372
P-n65-k10-C33	5	487	487	5	487	5	487
P-n70-k10-C24	4	385	385	4	385	4	385
P-n70-k10-C35	5	485	485	5	485	5	485
P-n76-k4-C26	2	320	309	2	309	2	309
P-n76-k4-C38	2	383	383	2	383	2	383
P-n76-k5-C26	2	309	309	2	309	2	309
P-n76-k5-C38	3	405	405	3	405	3	405
P-n101-k4-C34	2	374	370	2	370	2	370
P-n101-k4-C51	2	455	455	2	455	2	455

Tab. 5.19: Resultado de las ejecuciones realizadas sobre las instancias de tipo P presentadas en [7].

		Bektas et. al.			Ha et. al.		
Nombre de instancia	m_1	LNS	Branch & Cut	m_2	Metaheurística	m_3	OCH-SMPH
M-n101-k10-C34	4	458	458	4	458	4	458
M-n101-k10-C51	5	542	542	5	542	5	542
M-n121-k7-C41	3	527	527	3	527	3	527
M-n121-k7-C61	4	719	-	4	719	4	719
M-n151-k12-C51	4	483	-	4	483	4	489
M-n151-k12-C76	6	659	-	6	659	6	681
M-n200-k16-C100	8	791	-	9	789	9	844
M-n200-k16-C67	6	605	-	6	605	6	616

Tab. 5.20: Resultado de las ejecuciones realizadas sobre las instancias de tipo M presentadas en [7].

		Bektas et. al.			Ha et. al.		
Nombre de instancia	m_1	LNS	Branch & Cut	m_2	Metaheurística	m_3	OCH-SMPH
G-n262-k25-C131	12	3249	-	13	3303	13	3458
G-n262-k25-C88	9	2476	-	9	2477	9	2560

Tab. 5.21: Resultado de las ejecuciones realizadas sobre las instancias de tipo G presentadas en [7].

6. CONCLUSIONES

En este trabajo se empleó la metaheurística colonia de hormigas en su variante mejor-peor hormiga para la resolución de FTSP, GVRP y FVRP. Con respecto a la utilización de la metaheurística analizamos cada etapa del algoritmo y sus posibles mejoras, así como los mejores valores de sus parámetros. Determinamos que limitando los movimientos de las hormigas a un vecindario reducido se ayuda a una mejor y más rápida convergencia en la búsqueda. Analizamos el valor de la mutación desde el inicio hasta el estancamiento del algoritmo, intensificándola en esta última etapa. Observamos que los efectos producidos por el uso de la peor hormiga no afectan significativamente al resultado. Evaluamos distintas búsquedas locales y sus mejoras para luego ser utilizadas tanto en uno como en múltiples caminos.

Con respecto a la resolución de FTSP nos enfrentamos a un problema que había sido resuelto de 3 formas distintas: con CPLEX, y con las metaheurísticas BRKGA y GRASP+evPR. Los resultados que obtuvimos superaron en todas las instancias con más de 127 nodos a los conocidos hasta la actualidad. En las instancias más grandes se encuentran las mayores diferencias.

Por otro lado presentamos una formulación para el problema de ruteo de vehículos generalizado por familias (FVRP) y lo resolvimos utilizando el mismo algoritmo de FTSP con algunas diferencias relativas al problema. Comparamos nuestro algoritmo ejecutando instancias conocidas de GVRP con muy buenos resultados. Generamos nuevas instancias y presentamos valores de referencia para futuros análisis.

BRKGA, GRASP+evPR, y OCH-SMPH tienen como similitud que se basan en las mejores soluciones encontradas en las iteraciones anteriores para luego intensificar la búsqueda en dicha dirección. Ambas también tienen criterios de diversificación. BRKGA tiene la dificultad de tener que transformar en cromosomas a sus soluciones y realizar operaciones entre ellos de formas poco naturales. Sus resultados en instancias grandes son significativamente inferiores a los obtenidos por GRASP+evPR y OCH-SMPH. GRASP+evPR tiene un conjunto de buenas soluciones e intenta intensificar la búsqueda en aquellas direcciones, sin embargo en espacios muy grandes deben realizarse muy buenas búsquedas locales para obtener mejores resultados. Sin buenas búsquedas locales GRASP+evPR no alcanzaría buenos resultados. OCH-SMPH tiene como ventaja que la utilización de los mejores resultados previos se aprovechan en la construcción de otras soluciones con una buena combinación entre diversificación e intensificación. Además, luego de finalizar los recorridos utilizamos búsqueda local al igual que GRASP+evPR.

7. TRABAJO FUTURO

OCH-SMPH demostró ser una muy buena opción para resolver FTSP. En los últimos años se ha investigado mucho esta metaheurística, y aún podría haber mayor desarrollo ya que encara los problemas difíciles en grafos de manera natural.

El problema de ruteo de vehículos generalizado por familias (FVRP) es una nueva variante que debe ser más investigada. Por un lado el hecho de agregarle restricciones como ventanas de tiempo o nuevos objetivos como la reducción de vehículos quedó fuera de nuestro alcance. Por otro lado se podrían obtener mejoras utilizando heurísticas que determinen la cantidad de vehículos necesarios y heurísticas que agreguen nuevos vehículos para luego utilizar el algoritmo con una cantidad determinada de transportes.

Una posibilidad para continuar investigando sería la de no establecer la cantidad de visitas por familia a priori. La única restricción podría ser la de cumplir con la demanda de la familia y que la cantidad de visitas a la misma surja en la resolución del problema.

Bibliografía

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook **The Traveling Salesman Problem: A Computational Study**, Princeton Series in Applied Mathematics, Princeton University Press, 2006.

[2] W. Cook, D. Espinoza, M. Goycoolea, **Computing with Domino-Parity Inequalities for the TSP**, INFORMS Jornal Of Computing, vol. 19, p 356-365, 2005.

[3] I. Kara, H. Guden, O. N. Koc, **New Formulations for the Generalized Traveling Salesman Problem**, 5th EURO/INFORMS Joint International Meeting, İstanbul, Turkey, 2003.

[4] P. Toth, D.Vigo, **The Vehicle Routing Problem**, Society for Industrial and Applied Mathematics, Philadelphia, United States, 2002.

[5] L.F. Morán-Mirabal, J.L. González-Velarde, M.G.C. Resende, **Randomized heuristics for the family traveling salesperson problem**, International Transactions in Operational Research, vol. 21, p 41–57, 2014.

[6] I. Kara, T. Bektas, **Integer Linear Programming Formulation of the Generalized Vehicle Routing Problem**, 5th EURO/INFORMS Joint International Meeting, İstanbul, Turkey, 2003.

[7] T. Bektas, G. Erdogan, S. Ropke **Formulations and Branch-and-Cut Algorithms for the Generalized Vehicle Routing Problem**, Transportation Science, vol. 45, Informs, p 299-316, 2011.

[8] M. H. Ha, N. Bostel, A. Langevin, L. M. Rousseau, **An Exact Algorithm and a Metaheuristic for the Generalized Vehicle Routing Problem**, Computers & Operations Research, vol. 43, Elsevier Publishing, p 9-19, 2014.

[9] Z. Michalewitcz, D. Fogel **How to Solve It: Modern Heuristics**, Springer, 2004.

[10] M. Gendreau, J. Y. Potvin, **Handbook of Metaheuristics**, International Series in Operations Research & Management Science, vol. 146, Springer, 2010.

[11] A. Colorni, M. Dorigo, V. Maniezzo, **Distributed Optimization by Ant Colonies**, European Conference on Artificial Life, Paris, France, Elsevier Publishing, p 134-142, 1991.

[12] A. Atehortúa, **Ant Colony Optimization: Generalities and Study of Ant System Algorithm and a Job Shop Aplication**, National University of Colombia, Bogotá, Colombia 2012 (Tesis).

[13] A. Colorni, M. Dorigo, V. Maniezzo, **Ant system: optimization by a colony of cooperating agents**, IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 26, p 29-41, 1996.

[14] V. Maniezzo, L. M. Gambardella, F. de Luigi, **Ant Colony Optimization**, New Optimization Techniques in Engineering, Springer, p 101-117, 2004.

[15] T. Stützle, H. H. Hoos, **Max-Min Ant System**, Future Generation Computer Systems 16, Elsevier Publishing, p 889–914, 2000.

[16] S. Balseiro, **Logística y Distribución: Algoritmos para problemas de ruteo de vehículos con restricciones de capacidad y ventanas de tiempo**, University of Buenos Aires, Buenos Aires, Argentina, 2006 (Tesis).

[17] S. Alonso, O. Cordón, I. Fernández de Viana, F. Herrera, **La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques**, Optimización inteligente : técnicas de inteligencia computacional para optimización, Servicio de publicaciones de la Universidad de Málaga, p 263-272 2004.

[18] IBM CPLEX Optimizer, **http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/**.