Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro ISYS, Laboratorio de Inteligencia Artificial

Framework basado en Colonias de Hormigas artificiales para la resolución de problemas de optimización

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por el Bachiller
Enrique Alejandro Areyán Viqueira
C.I. 17.023.919

Ignacio Calderón López
Haydemar Nuñez

Caracas, 25 / 01 / 2010

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Acta del Veredicto

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación, para dictaminar sobre el Trabajo Especial de Grado titulado: "Framework basado en Colonias de Hormigas Artificiales para la resolución de problemas de optimización" y presentado por el bachiller Enrique Alejandro Areyán Viqueira, cédula de identidad V –17.023.919, para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del jurado, se fijó el día 25 de enero de 2010 a las 10:30 a.m., para que su autor lo defendiera en forma pública, lo que hizo en el aula PB-III de la Escuela de Computación, mediante una presentación oral del contenido del Trabajo Especial de Grado, luego de lo cual respondió a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió APROBARLO.

En fe de lo cual se levanta la presente Acta, en la Ciudad Universitaria de Caracas a los veinticinco días del mes de enero del año dos mil diez, dejándose también constancia de que actúo como Coordinadora del Jurado la Profesora Haydemar Núñez.

Jurado Principal

_____	_____
Profesora Esmeralda Ramos	Profesor Eugenio Scalise
_____	_____
Lic. Ignacio Calderón (Tutor)	Profesora Haydemar Nuñez (Tutora)

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis tutores Haydemar Nuñez e Ignacio Calderón por toda la ayuda prestada en la elaboración del presente trabajo. En especial, quiero agradecer a Ignacio por toda su desinteresada dedicación y apoyo desde la concepción inicial del trabajo, la construcción del Software y hasta la última página escrita. Muchas Gracias y cuenta conmigo en lo que pueda ayudarte en el futuro.

A mis padres, María José Viqueira y Jesuardo Areyán, y mis hermanos, Mariana Areyán y Eduardo Areyán, por todo el apoyo incondicional a lo largo de los años. Sepan que en mí tienen un amigo incondicional.

A Karina, fuente de inspiración y apoyo incondicional en todos mis proyectos, por más descabellados que pueden parecer al principio. Sin ti este nuevo logro se sentiría vacío. Gracias.

A Hernán Rosas, un gran amigo quién me guió en los primeros pasos hacia pensar como un Profesional de la Computación.

A mi Alma Mater, la Universidad Central de Venezuela (UCV), en donde por cuestiones del azar o el destino me tocó cursar la primera etapa de mis estudios Universitarios. No cambiaría un solo día de los que he vivido aquí.

DEDICATORIA

A mi madre, la base en la que todo ha sido posible.

A mi padre, de quién heredé una infinita curiosidad intelectual y quién con su ejemplo me demuestra a diario que la vida es lo que uno quiera hacer de ella.

Índice

Introducción 2
CAPÍTULO I: MARCO TEÓRICO 4
1.1 Introducción 4
1.2 Colonias de Hormigas (biológicas) 4
1.3 Experimento de Puente Doble 5
1.4 Modelo de Colonias de Hormigas 6
1.5 Algoritmos basados en OCH. 10
CAPÍTULO II: MARCO APLICATIVO 30
2.1 Introducción 30
2.2 Planteamiento del Problema 30
2.3 Objetivos 31
2.4 Propuesta de Framework 32
2.5 Diseño del Framework 33
2.6 Aplicación Ejemplo 51
2.7 Experimentos de Verificación 58
CONCLUSIONES Y RECOMENDACIONES 84
Referencias 87
ANEXOS 89
Anexo 1. Pseudo Código del algoritmo SHMM 89
Anexo 2. Pseudo Código del algoritmo SCH 91
Anexo 3. Grafos Utilizados 93

Introducción

Las teorías emergentes sobre el caos explican que, de sistemas gobernados por reglas muy simples, es posible obtener comportamientos y resultados complejos que a simple vista pueden considerarse caóticos y erráticos, sin mucho sentido ni aplicación práctica, pero que al ser estudiados con mayor detenimiento resultan en formas sorprendentes que hacen pensar justo lo contrario. Por ejemplo, en matemáticas, el conjunto de Mandelbrot muestra esta situación: de la iteración y retroalimentación positiva del siguiente polinomio cuadrático complejo: $P_c : z \to z^2 + c$, se obtienen estructuras arduo complejas, que al ser graficadas muestran un mundo sorprendente en sí mismas.

Asimismo, dichas teorías tratan de obtener un conocimiento más preciso sobre sistemas en la naturaleza que a simple vista parecen ser caóticos, pero que funcionan con increíble precisión. En el presente trabajo se tratará con uno de estos sistemas: Las Colonias de Hormigas. Una vez entendida y superada la simplicidad de cada uno de sus miembros y su aparente comportamiento caótico, nos encontramos con que esta sociedad de insectos logra organizarse e interactuar de forma tal de cumplir con los objetivos primordiales de cualquier ser vivo: búsqueda de alimento, organización del trabajo, cuidado de la cría, etc.

En particular, se hará una revisión de los modelos computacionales derivados del comportamiento de algunas sociedades de hormigas en la búsqueda de alimento. Sorprendentemente, dichos modelos no sólo sirven como ejemplo del orden dentro del aparente caos, sino que incorporan gran utilidad práctica al momento de resolver diversos problemas de optimización, comunes en el día a día.

Luego, y como objetivo principal, se construirá un Framework en Software que incorporará los principales algoritmos de Colonias de Hormigas para la resolución de problemas de optimización. Dicho Framework se construirá bajo el paradigma de programación orientada a objetos y le proveerá al programador de interfaces fáciles de utilizar y combinar para hacer uso de algún algoritmo de Colonias de Hormigas.

Asimismo, el Framework podrá ser extendido fácilmente para incorporar nuevos algoritmos que puedan surgir en el futuro o cubrir alguna necesidad especial.

El presente documento se divide en dos capítulos. En el primer capítulo se trata el tema de los sistemas bioinspirados y específicamente, de las Colonias de Hormigas naturales. Se presenta una revisión de los experimentos y teorías formuladas por profesionales en el área de la Biología, las cuales sirvieron de inspiración para concebir los primeros modelos artificiales de Colonias de Hormigas, aplicables a la resolución de problemas de optimización. Asimismo, se introduce la metaheurística de Optimización por Colonias de Hormigas (OCH), la cual es el corazón de los algoritmos basados en Colonias de Hormigas, y se presenta el Sistema de Hormigas, un primer intento de aplicación de la metaheurística OCH.

En el segundo capítulo se detallará la construcción del Framework en Software de los algoritmos de Colonias de Hormigas previamente explicados. Finalmente, se muestran resultados de experimentos realizados sobre dicho Framework los cuales permitirán comparar el rendimiento de los diversos algoritmos de Colonias de Hormigas contra otras técnicas en optimización y entre sí. Por último, se presentan las conclusiones alcanzadas y se hacen sugerencias de mejoras que podrían realizarse al sistema en un futuro.

CAPÍTULO I: MARCO TEÓRICO

1.1 Introducción

En la naturaleza se encuentran sociedades de seres vivos que a pesar de la simplicidad de cada uno de sus miembros logran organizarse y cooperar entre sí, siendo capaces, en conjunto, de realizar tareas complejas que resultarían imposibles de llevar a cabo por uno sólo de sus miembros. Estas sociedades pueden considerarse como sistemas distribuidos, capaces de adaptarse al medio ambiente.

Los sistemas bioinspirados buscan emular el comportamiento de sociedades de seres vivos con el fin de resolver problemas computacionales complejos y de distinta índole. A diferencia de otros métodos tradicionales de la Inteligencia Artificial donde es el programador quién, a través de distintas técnicas le confiere "inteligencia" a un sistema, en los sistemas bioinspirados se definen tanto un conjunto de reglas simples como un conjunto de organismos simples que obedecen dichas reglas. A esto se le suma un método iterativo que aplica las reglas a estos organismos, y luego de varias iteraciones, típicamente surge algún tipo de comportamiento o estructura compleja.

1.2 Colonias de Hormigas (biológicas)

Las Colonias de Hormigas son un ejemplo de sociedades de seres vivos capaces de organizarse y cooperar entre sí, "que pueden considerarse como sistemas distribuidos que a pesar de la simplicidad de sus miembros, presentan una organización social altamente estructurada y compleja. Como resultado de esta organización, las Colonias de Hormigas pueden realizar tareas complejas que exceden la capacidad individual de cada hormiga" (Dorigo, 2004, p. 1).

El mecanismo por medio del cual las hormigas coordinan sus actividades se conoce en inglés como *stigmergy* (trabajo para el objetivo común), una forma indirecta de comunicación a través de pequeñas modificaciones al medio ambiente. A menudo, la comunicación a través del medio ambiente es la única forma de comunicación para algunas especies de hormigas que son ciegas o parcialmente

ciegas.

Más aún, se sabe que una hormiga en busca de alimento depositará un químico llamado feromona en el suelo, el cual introduce un sesgo que aumenta la probabilidad de que otras hormigas sigan el mismo camino que ésta (Dorigo, 2004). De esta forma, una hormiga comunica a los otros miembros de la Colonia un posible camino o vía hacia un depósito de comida y una vía de regreso al nido.

1.3 Experimento de Puente Doble

Existe un experimento diseñado y llevado a cabo por Deneubourg et al. (Deneubourg, Aron, Goss & Pasteels, 1990) de gran importancia para comprender el comportamiento de las Colonias de Hormigas en búsqueda de alimento. Dicho experimento utiliza un puente con dos ramificaciones (puente doble) que conecta un nido de hormigas con una fuente de alimento, en el cual se varía el radio entre la longitud de las dos ramificaciones del puente $(r = l_g / l_c)$, donde l_g es la longitud de la ramificación larga y l_c la longitud de la ramificación más corta (ver Figura 1.1).

Se presentan dos casos: en el primero las dos ramificaciones son de igual longitud ($r = 1$) y en el segundo la ramificación más larga es el doble que la ramificación más corta ($r = 2$).

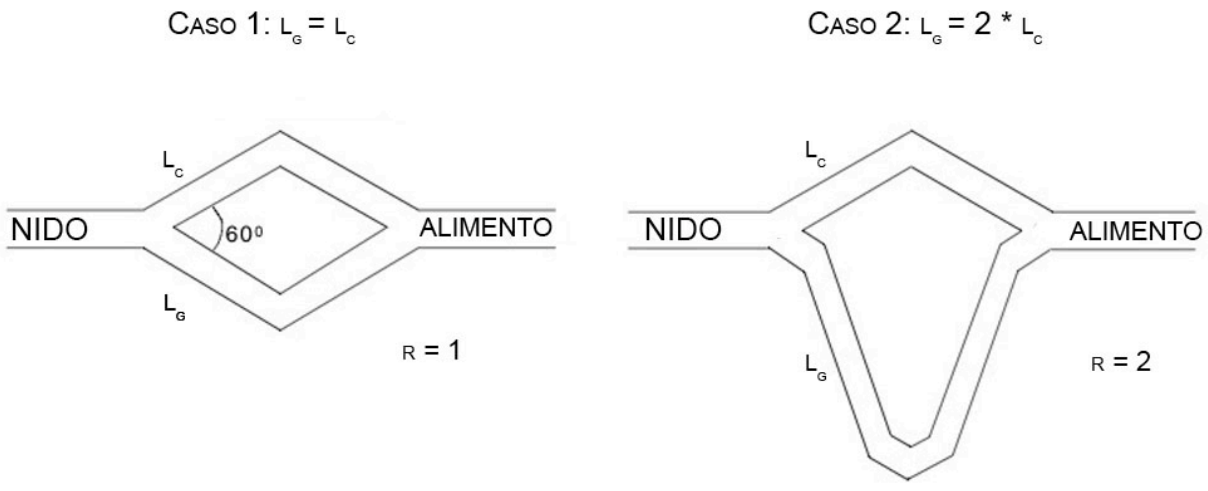

Figura 1.1. Experimento de puente doble

Se realizaron varios ensayos de este experimento en los cuales a las hormigas se les permitió moverse con libertad entre el nido y la fuente de alimento. En el primer caso con $r = 1$ se determinó que a pesar de que al inicio las hormigas tomaban decisiones aleatorias con respecto a cual ramificación seguir, eventualmente todas las hormigas utilizaron una misma ramificación. En algunos ensayos siguieron la ramificación l_g y en otros ensayos la ramificación l_c indistintamente. La explicación dada es la siguiente: cuando se inicia un ensayo no existen rastros de feromona en ninguna ramificación y sin embargo, debido a fluctuaciones aleatorias, más hormigas seleccionarán una ramificación en vez de la otra. Dado que las hormigas depositan feromonas al caminar, la ramificación con mayor número de hormigas tendrá mayor concentración de feromonas, lo cual estimula a más hormigas a seguir dicho camino. Cabe destacar que a este tipo de comportamiento se le conoce como retroalimentación positiva, y es un ejemplo de la capacidad auto organizativa de las hormigas, en donde el comportamiento global emerge como la interacción entre el comportamiento individual de cada hormiga.

Más aún, en el segundo caso con $r = 2$ en la mayoría de los ensayos después de algún tiempo, todas las hormigas deciden utilizar el camino más corto. Al igual que en el caso anterior, las hormigas inician escogiendo cualquiera de las dos ramificaciones l_c o l_g indistintamente. De esta forma, es de esperarse que en promedio la mitad de las hormigas seleccionen l_g y la otra mitad l_c. Sin embargo, dado que una ramificación es más corta que otra, aquellas hormigas que seleccionen el camino más corto son las primeras en alcanzar la fuente de alimento y emprender su viaje de vuelta al nido. Así, como resultado del proceso de retroalimentación positiva explicado anteriormente, luego de transcurrido un cierto tiempo existirá mayor concentración de feromonas en la ramificación más corta, sesgando la decisión del resto de las hormigas de la colonia hacia la utilización de dicho camino.

1.4 Modelo de Colonias de Hormigas

Los modelos computacionales inspirados en Colonias de Hormigas buscan emplear algún tipo de *stigmergy* artificial o rastros de feromona por medio de los

cuales se pueda coordinar la interacción entre los agentes artificiales (en este caso "hormigas"), con el fin de emular el comportamiento de las Colonias de Hormigas naturales.

En este punto es importante mencionar la destacada labor del científico Marco Dorigo, quién es el director de investigación del Fondo Belga para Investigación Científica, así como co-director del IRIDIA, el laboratorio de Inteligencia Artificial de la Université Libre de Bruxelles (http://code.ulb.ac.be/code.people.php?id=2). Dorigo es quién propone la metaheurística de Optimización por Colonias de Hormigas (OCH) o *Ant Colony Optimization* en inglés, dentro de la cual se enmarcan la mayoría de las implementaciones de algoritmos de Colonias de Hormigas para resolver problemas en concreto. Por tanto, muchas de las ideas de este científico serán la base del presente documento.

1.4.1 Modelo Artificial

Intuitivamente, el comportamiento derivado de la búsqueda de alimento de las Colonias de Hormigas naturales se puede modelar como un problema computacional de búsqueda en un grafo conexo, en donde hormigas artificiales harán uso del rastro de feromona artificial para tomar decisiones de acuerdo a las limitantes del problema a optimizar.

1.4.1.1 Feromona Artificial

Considerando el grafo conexo G = (*N, A*), donde *N* es el conjunto n = | *N* | de nodos y *A* es el conjunto de arcos dirigidos conectando dichos nodos, entonces para cada arco *(i, j)* del grafo G se asocia una variable τ_{ij} a la que llamaremos rastro artificial de feromona. Típicamente, este conjunto de variables se representan en el computador como una matriz M de *N* x *N* de números reales, en donde la posición M[i, j] representa el arco (i, j).

1.4.1.2 Hormiga Artificial

"Una hormiga artificial puede ser considerada como un procedimiento estocástico" (Dorigo, 2004, p. 7), que se vale de unas reglas predefinidas y los

valores contenidos en la estructura de feromona artificial, para construir posibles caminos o soluciones a un problema de búsqueda. Adicionalmente, una hormiga artificial contendrá una memoria para representar su estado actual y los distintos estados visitados para llegar al punto actual.

1.4.2 Metaheurística OCH

Los distintos algoritmos basados en Colonias de Hormigas naturales se enmarcan dentro de lo que se conoce como la metaheurística de Optimización por Colonias de Hormigas (OCH). Según Dorigo (2004), una metaheurística es un conjunto de conceptos algorítmicos que pueden ser utilizados para definir métodos heurísticos aplicables a una gran cantidad de problemas distintos.

Como se propone en (Dorigo, 2004), cualquier algoritmo concebido dentro de la metaheurística OCH puede ser interpretado como la interacción de tres procedimientos básicos: construir soluciones de hormigas, actualizar feromonas y actividades misceláneas. A continuación se explica cada uno de estos procedimientos:

- **Construir soluciones de hormigas**

Este procedimiento se encarga de manejar la Colonia de Hormigas, emulando el movimiento asíncrono y concurrente de cada hormiga artificial mientras recorren el ya mencionado grafo conexo *G,* que representa el problema a optimizar. Las hormigas artificiales se “mueven” dentro del grafo aplicando una regla de decisión que hace uso tanto de la información contenida en la matriz de feromonas, como alguna heurística particular que puede variar en distintas implementaciones.

Cuando una hormiga termina de construir una solución o mientras aún se encuentra construyéndola, hará uso del procedimiento *actualizar feromona* para decidir la cantidad de feromona a depositar (Dorigo, 2004, p. 37).

El siguiente pseudo código muestra de forma general como se implementaría el procedimiento *construir soluciones de hormigas*:

```
Siendo r = U(0,1) # Variable uniformemente distribuida entre 0 y 1.
                  # esto puede cambiar según la variante del algoritmo
                  # que se implemente
Para cada potencial camino en G hacer
# El cálculo de P_a varía según el  algoritmo que se implemente.
Calcular P_a utilizando los valores de τ_ij y de la heurística si aplica
      Si r <= P_a entonces
            Seguir el camino G;
            Romper;
      Fin si
Fin para
```

- **Actualizar feromonas**

Este procedimiento se encarga de modificar la matriz de feromona, aumentando o disminuyendo la cantidad de feromona depositada en una conexión *(i, j)* cualquiera del grafo. Incrementar la cantidad de feromona depositada en un arco *(i, j)*, aumentará la probabilidad de que otras hormigas utilicen dicho arco en la construcción de sus soluciones particulares. Luego, un arco *(i, j)* que haya sido "visitado" muchas veces será más probable de ser escogido por mayor cantidad de hormigas en el futuro, indicando así una posible buena solución.

Por otra parte, disminuir la cantidad de feromona depositada en un arco *(i, j)* cualquiera del grafo *G* es una forma útil de implementar un mecanismo de "olvido" de una solución que "evita una convergencia muy rápida del algoritmo hacia una región subóptima y de esta forma favorece la exploración de nuevas áreas del espacio de búsqueda" (Dorigo 2004).

- **Actividades misceláneas**

Este procedimiento se encarga de llevar a cabo aquellas tareas centralizadas que no puede realizar ninguna hormiga por sí sola y de forma aislada.

Este procedimiento varía sensiblemente entre distintas implementaciones de algoritmos basados en OCH. Algunos ejemplos de actividades que se ejecutarían en

este procedimiento son: recolección de información global para la toma de decisiones en conjunto, ejecución de procedimientos de optimización local, entre otros.

1.5 Algoritmos basados en OCH.

Una vez analizada la metaheurística OCH, a continuación se describen algunos de los principales algoritmos basados en la metaheurística OCH.

1.5.1 Sistema de Hormigas (SH)

- **Introducción al Sistema de Hormigas**

Este fue el primer algoritmo basado en la metaheurística OCH y se propuso en (Dorigo, Maniezzo, Colorni, 1996). Su característica principal es que el rastro de feromonas es actualizado por todas aquellas hormigas que hayan completado un camino o *tour* (http://www.scholarpedia.org/article/Ant_colony_optimization).

- **Construcción de caminos**

Como explica Dorigo este algoritmo inicia con *m* distintas hormigas artificiales ubicadas aleatoriamente en los nodos del grafo G. En cada iteración, una hormiga cualquiera *k* aplicará lo que el autor llama una regla proporcional aleatoria para decidir el próximo nodo a visitar. Esta regla viene dada por la ecuación:

$$p(k)_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathrm{N}_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \quad \text{si} \quad j \in \mathrm{N}_i^k \tag{1.1}$$

Donde $n_{ij} = 1/d_{ij}$ (información heurística), d_{ij} es el costo de trasladarse desde el nodo *i* hacia el nodo *j*, α y β son dos parámetros que controlan la influencia del rastro de feromona y de la información heurística. N_i^k es el conjunto de nodos accesibles para la hormiga *k* situada en el nodo *i*.

- **Actualización del rastro de feromona**

Este algoritmo hace uso de una forma artificial de evaporación de feromona que se implementa en el marco de las Actividades Misceláneas de OCH tal y como

sigue:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in L \tag{1.2}$$

Donde $0 < \rho \leq 1$ es la tasa de evaporación.

Una vez que ocurre la evaporación de feromonas, las hormigas depositan feromonas obedeciendo la siguiente regla (*k* representa una hormiga cualquiera):

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k},\ \forall (i,j) \in L \tag{1.3}$$

Donde *m* es el número de hormigas, $\Delta\tau_{ij}^{k}$ es la cantidad de feromona depositada por la hormiga artificial *k* en los arcos que ella ha visitado y se define como:

$$\Delta\tau_{ij}^{k} = 1/C^{k},\ \text{si el arco (i, j) pertenece a } T^{k}\text{, 0 sino}$$

Donde C^{k} es el largo del camino T^{k} construido por la k-ésima hormiga artificial y es calculado como la suma de los largos de los arcos pertenecientes a T^{k}.

Como consecuencia directa de la ecuación anterior se sabe que las hormigas depositan feromonas proporcionalmente a la calidad del camino construido. En general, aquellos arcos pertenecientes a caminos más cortos y que son utilizados por más de una hormiga, recibirán mayor cantidad de feromonas y por tanto, será más probable que otras hormigas escojan dichos arcos en futuras iteraciones.

- **Pseudo código SH**

Finalmente, Engelbretch (2007) propone el siguiente pseudo código como un resumen del algoritmo SH:

```
t = 0;
Inicializar todos los parámetros: α, β, τ0, nk
```

Ubicar a todas las hormigas artificiales $k = 1,\ldots, n_k$;

Para cada arco (i, j) del grafo *G* **hacer**

$\tau_{ij}(t) = U(0 , \tau_0)$;

Fin Para

Repetir

Para cada hormiga $k = 1,\ldots,n_k$ **hacer**

$x^k(t) = \emptyset$; #siendo x^k el vector de caminos construidos para #la hormiga k

Repetir

Desde el nodo *i*, seleccionar un posible nodo destino *j* utilizando la *regla proporcional aleatoria*:

Ecuación (1.1)

$x^k(t) = x^k(t) \cup \{(i,j)\}$;

hasta que se construya un camino completo

Calcular $f(x^k(t))$; #siendo f la función objetivo

Fin Para

Para cada arco (i, j) **hacer**

Aplicar evaporación de feromonas según la ecuación:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in L$$

Calcular $\Delta\tau_{ij}(t)$;

Actualizar el rastro de feromonas según la ecuación:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k},\ \forall (i,j) \in L$$

Fin para;

Para cada arco(i, j) **hacer**

$$\tau_{ij}(t+1) = \tau_{ij}(t);$$

Fin para;

t = *t* + 1;

hasta alcanzar condición de parada;

Retorna $x^t(t) : f(x^k(t)) = \min_{k=1,\ldots,nk}\{f(x^k(t)\}$;

- **Ejemplo práctico**

En esta sección se detalla el funcionamiento del algoritmo SH. Se mostrará una traza del algoritmo cuando se aplica a la búsqueda de la ruta más corta desde el

nido de la Colonia de Hormigas hacia una fuente de alimento. Para ello, consideremos el grafo de la Figura 1.2:

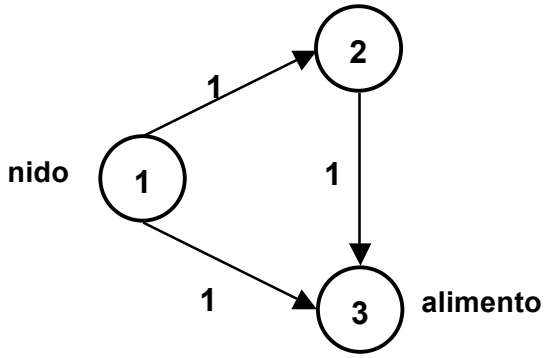

Figura 1.2. Grafo que modela el experimento de puente doble

El grafo de la Figura 1.2 es un posible modelo discreto del experimento de puente doble descrito en la Sección 1.3. En el mismo se puede observar que existen dos caminos desde el nido (nodo 1) hasta la fuente de alimento (nodo 3), a saber:

Camino corto: del nodo 1 al nodo 3. Costo del camino = 1.

Camino largo: del nodo 1 al nodo 2 al nodo 3. Costo del camino = 2.

Se puede observar que el camino largo es dos veces más costoso que el camino corto. Adicionalmente, el grafo se puede representar mediante la siguiente matriz de adyacencia.

(i, j)	1	2	3
1	∞	1	1
2	∞	∞	1
3	∞	∞	∞

Figura 1.3. Matriz de adyacencia del grafo en la Figura 1.2

La matriz de adyacencia anterior contiene la información del costo de trasladarse desde cada nodo *i* a cada nodo *j* cualquiera del grafo. Cabe destacar que en aquellas posiciones *(i, j)* donde el costo sea ∞ se considerará que no es posible trasladarse desde el nodo *i* al nodo *j*.

Para la corrida del algoritmo SH, se utilizarán los siguientes parámetros:

Parámetros	Valores
iteraciones	2
hormigas	2
α	0.5
β	2
ρ	0.5

A continuación se muestra la traza del algoritmo. Para esto, se utilizará la variable *t* como un contador del número de iteraciones.

$t = 0$. Primera iteración del algoritmo en donde se realizan las siguientes tareas básicas: se inicializan los parámetros, se ubican todas las hormigas en el nodo correspondiente al nido (nodo 1) y se inicializa el rastro o matriz de feromonas con números aleatorios uniformemente distribuidos entre [0,1]. Por ejemplo, la matriz de feromonas se puede inicializar de la siguiente forma:

0.623	0.815	0.763
0.323	0.072	0.185
0.007	0.839	0.372

Recordemos que el rastro de feromonas τ_{ij} es una matriz de igual dimensión que la matriz de adyacencia del grafo. Asimismo, cada posición *(i, j)* de esta matriz se corresponde exactamente con la posición *(i, j)* de la matriz de adyacencia. Por ejemplo, en este caso la variable $\tau_{12} = 0.815$ indica la concentración de feromonas en el arco que conecta al nodo 1 con el nodo 2.

A continuación, para cada hormiga en el grafo, se construye una solución.

Hormiga $k = 1$: Solución = $\varnothing$. Desde el nodo 1 se selecciona un nodo *j* utilizando la regla proporcional aleatoria de la ecuación 1.1:

$$p(k)_{ij} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \text{ si } j \in N_i^k$$

En este caso el conjunto de nodos accesibles es $N_1^1 = \{2,3\}$. Luego, se debe aplicar la regla proporcional aleatoria para cada uno de estos nodos hasta escoger alguno.

Del nodo 1 al nodo 2:

$$p(1)_{12} = \frac{[0.815]^{0.5}[1]^2}{[0.815]^{0.5}[1]^2 + [0.763]^{0.5}[1]^2} = 0.502$$

Del nodo 1 al nodo 3:

$$p(1)_{13} = \frac{[0.763]^{0.5}[1]^2}{[0.815]^{0.5}[1]^2 + [0.763]^{0.5}[1]^2} = 0.491$$

A continuación se genera un número aleatorio uniformemente distribuido entre [0,1] y según éste se selecciona o bien el nodo 2 o el nodo 3 para formar parte de la solución. Suponiendo que dicho número es $0.5 \geq 0.491$, se selecciona el nodo 3. Se verifica si ya se tiene una solución final, lo cual es cierto en éste caso. Luego, esta hormiga almacena el camino {1,3} como la solución obtenida. Se pasa a la siguiente hormiga.

Hormiga $k = 2$: Solución = $\varnothing$. Desde el nodo 1 se selecciona un nodo *j* igual que con la hormiga anterior. Los probabilidades son iguales que en el caso anterior. Supongamos que en este caso el número aleatorio generado para seleccionar el nodo es $0.731 \geq 0.502$, por lo que esta hormiga seleccionará el nodo 2 para formar parte de la solución = {1,2}. A continuación la hormiga debe seleccionar un nuevo nodo accesible desde el nodo 2. Sin embargo, desde el nodo 2 sólo hay un nodo accesible: el nodo 3, por lo que éste pasa inmediatamente a formar parte de la solución final {1,2,3}.

Una vez las 2 hormigas de esta Colonia terminan su trabajo, se pasa a realizar la evaporación y actualización del rastro de feromonas.

Para la evaporación de feromonas de emplea la ecuación 1.2:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in L$$

Se disminuirá el rastro de feromona por una constante, para este caso 0.5, por lo que el rastro de feromonas se convierte según la relación $\tau_{ij}(t+1) = 0.5 * \tau_{ij}(t)$

$\tau_{ij}(t)$		
0.623	0.815	0.763
0.323	0.072	0.185
0.007	0.839	0.372

$\tau_{ij}(t+1)$		
0.3115	0.4075	0.3815
0.1615	0.036	0.0925
0.0035	0.4195	0.186

Luego, se actualizará el rastro de feromonas en función de la calidad de las soluciones obtenidas por las hormigas y según la ecuación 1.3:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k},\ \forall (i,j) \in L \text{ con } \Delta\tau_{ij}^{k} = 1/C^{k},$$

La ecuación indica que se actualizarán los arcos del grafo correspondientes a las soluciones encontradas por las hormigas y se hará inversamente proporcional a la calidad del camino (largo del camino). En nuestro ejemplo las hormigas construyeron dos caminos: hormiga 1= {1,3} cuyo costo es 1 y hormiga 2= {1,2,3} cuyo costo es 2. Luego, la hormiga 1 incrementará el rastro de feromonas según la ecuación: $\tau_{13} = \tau_{13} + 1 = 1.3815$ y la hormiga 2 según $\tau_{12} = \tau_{12} + (1/2) = 0.9075$ y $\tau_{23} = \tau_{23} + (1/2) = 0.5925$. El rastro de feromonas queda así

$\tau_{ij}(t+1=1)$		
0.3115	**0.9075**	**1.3815**
0.1615	0.036	**0.5925**
0.0035	0.4195	0.186

Se observa que en la matriz anterior, existe una mayor concentración de feromonas en los arcos del camino más corto, en este caso τ_{13}. Esto a su vez

sesgará la decisión de las hormigas hacia este camino tal y como veremos en la próxima iteración

$t = 1$. Segunda iteración. El funcionamiento es exactamente igual que en la primera iteración a excepción de las tareas de inicialización las cuales se realizan sólo al principio.

Hormiga $k = 1$: Solución = $\varnothing$. Desde el nodo 1 se selecciona un nodo j.

Del nodo 1 al nodo 2:

$$p(1)_{12} = \frac{[0.9075]^{0.5}[1]^2}{[0.9075]^{0.5}[1]^2 + [1.3815]^{0.5}[1]^2} = 0.447$$

Del nodo 1 al nodo 3:

$$p(1)_{13} = \frac{[1.3815]^{0.5}[1]^2}{[0.9075]^{0.5}[1]^2 + [1.3815]^{0.5}[1]^2} = 0.552$$

Suponiendo la variable de decisión $0.89 \geq 0.447$ se selecciona el nodo 2. Sabemos entonces que la solución que construirá esta hormiga es {1,2,3}.

Hormiga $k = 2$: Solución = $\varnothing$. Iguales probabilidades que la hormiga 1. Suponiendo que se selecciona el nodo 3, la solución construida es {1,3}.

Se aplica la evaporación de feromonas:

$\tau_{ij}(1)$		
0.3115	0.9075	1.3815
0.1615	0.036	0.5925
0.0035	0.4195	0.186

$\tau_{ij}(2)$		
0.15575	0.45375	0.69075
0.08075	0.018	0.29625
0.00175	0.20975	0.093

Se aplica la actualización de feromonas. La hormiga 1 actualiza según $\tau_{12} = \tau_{12} + (1/2) = 0.95375$ y $\tau_{23} = \tau_{23} + (1/2) = 0.7962$. Por su parte, la hormiga 2 refuerza únicamente el arco $\tau_{13} = \tau_{13} + 1 = 2.1907$. El nuevo rastro de feromonas es:

$\tau_{ij}(t+1=2)$		
0.3115	**0.95375**	**2.1907**
0.1615	0.036	**0.7962**
0.0035	0.4195	0.186

Se observa que el rastro de feromona refleja las decisiones tomadas por las hormigas artificiales, y se puede concluir que estas encuentran el camino más corto {1,3} del nido a la fuente de alimento. El algoritmo continuará su funcionamiento tal y como se ha explicado antes, de lo que se puede deducir que los arcos utilizados en soluciones mantendrán una alta concentración de feromona, mientras que los demás tienden rápidamente a cero. Más aún, el arco perteneciente a la solución óptima tendrá un nivel de feromonas considerablemente mayor que cualquier otro arco. En este punto se puede ver que la Colonia de Hormigas convergerá hacia la solución óptima.

1.5.2 Variantes del Sistema de Hormigas (SH)

Si bien el SH representó una primera implementación exitosa de un algoritmo basado en la metaheurística OCH, se encontró que su rendimiento disminuía drásticamente a medida que aumentaba el tamaño del problema. Por tanto, una gran cantidad de esfuerzo en la investigación de algoritmos de Colonias de Hormigas fue dirigida a encontrar mecanismos para mejorar el desempeño del SH.

Como resultado de dicho esfuerzo de investigación, nacen tres variantes de SH: Sistema de Hormigas Élite (SHE) o *Elitist Ant System* en inglés, Sistema de Hormigas Basado en Rango (SH_{rango}) o *Rank-Based Ant System* en inglés y Sistema de Hormigas Basado en Máximos y Mínimos (SHMM) o *MAX-MIN Ant System* en inglés.

A continuación se analizarán cada una de las variantes del SH mencionadas anteriormente:

1.5.2.1 Sistema de Hormigas Élite (SHE)

Propuesto por Dorigo (1992) y Dorigo et al. (1996), este algoritmo es una primera mejora con respecto al SH. Su principal característica, como su nombre lo indica, es la implementación de una estrategia elitista.

El elitismo aplicado a algoritmos de Colonias de Hormigas artificiales consiste en reforzar los arcos correspondientes al mejor camino construido por las hormigas desde que inició la ejecución del algoritmo. Este camino se denota como T^{bs} *best-so-far tour* o el mejor camino hasta el momento.

Dicho refuerzo se implementa mediante una hormiga llamada *best-so-far ant* o mejor hormiga hasta el momento, que será aquella hormiga que en una iteración cualquiera construya el mejor camino. Sólo a esta hormiga se le permitirá depositar una cantidad adicional de feromona en los arcos de su tour.

- **Actualización del rastro de feromona en SHE**

El refuerzo adicional al T^{bs} se implementa como la suma de una cantidad $x = e * C^{bs}$ a los arcos del mismo, donde *e* es un parámetro que define la importancia o peso que se le dará al T^{bs} y C^{bs} es la longitud del tour.

Es así como la ecuación de actualización de feromonas para el SH se convierte en:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} (\Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}) \ , \tag{1.10}$$

Donde,

$\Delta\tau_{ij}^{k} = 1/C^{k}$, si el arco (i, j) pertenece a T^{k}, 0 sino

Y,

$\Delta\tau_{ij}^{bs} = 1/C^{bs}$, si el arco (i, j) pertenece a T^{bs}, 0 sino

Nótese que en este algoritmo, la evaporación de feromona se implementa de la misma forma que en el SH. Los resultados presentados por Dorigo (1992) y Dorigo et al. (1991a, 1996) sugieren que el uso de estrategias elitista con un adecuado valor para el parámetro *e*, efectivamente mejoran el rendimiento del SH. La mejora se presenta en la búsqueda de mejores tours, así como en un menor número de iteraciones.

1.5.2.2 Sistema de Hormigas basado en Rango (SH_{rango})

Propuesto por Bullnheimer et al. (1999) y como explica Engelbretch (2007), en SH_{rango} se proponen las siguientes modificaciones al SH:

1. Permitir únicamente a la hormiga con el mejor recorrido depositar feromonas en los arcos del T^{bs} (el mejor tour hasta el momento)
2. Emplear hormigas elitistas
3. Permitir a las hormigas actualizar feromonas en función de su rango

- **Actualización del rastro de feromona en SH_{rango}**

Al igual que en SHE, la mejora del algoritmo SH_{rango} con respecto a su predecesor se hace evidente en la estrategia empleada al momento de actualizar el rastro de feromona. Dicha estrategia consiste en ordenar a las hormigas por orden de longitud del tour construido en una iteración antes de depositar feromonas. Adicionalmente, la cantidad de feromonas que una hormiga depositará será una función de su rango *r*. Los empates entre hormigas de igual rango podrán ser resueltos de forma aleatoria.

Así, en cada iteración, únicamente a las *(w – 1)* hormigas de mayor jerarquía, siendo *w* un parámetro establecido a priori, y a la hormiga que produce el T^{bs} se les permitirá depositar feromona. La hormiga que produce el T^{bs} es la que mayor sesgo introduce, ya que depositará feromona según la ecuación: $1/C^{bs} * w$; mientras que la r-ésima hormiga de mayor rango en la iteración actual depositará feromonas según: $(1/C^{r}) * \max\{0, w - r\}$.

La regla de actualización de feromona para SH_{rango} queda entonces como:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}, \tag{1.11}$$

Donde,

$$\Delta\tau_{ij}^{r} = 1/C^{r} \quad \text{y} \quad \Delta\tau_{ij}^{bs} = 1/C^{bs}$$

Los resultados de una evaluación experimental llevada a cabo por Bullnheimer et al (1999) sugieren que SH_{rango} otorga resultados algo mejores que SHE y significativamente mejores que SH. (Dorigo, 2004, p. 74)

1.5.2.3 Sistema de Hormigas Basado en Máximos y Mínimos (SHMM)

Propuesto por Stützle & Hoos (1997), este algoritmo fue concebido con el fin de superar un problema específico del SH, i.e.: evitar el estancamiento prematuro en soluciones subóptimas para problemas complejos. Concretamente, el estancamiento prematuro significa que todas las hormigas siguen el mismo camino, y por lo tanto exploran sólo una sola parte del espacio de búsqueda que puede no contener la mejor solución. Esto ocurre cuando las hormigas explotan muy rápidamente las grandes concentraciones de feromonas de la matriz de feromonas, acabando con la información de búsqueda.

Como sugiere Engelbretch (2007), la gran diferencia entre el Sistema de Hormigas Máximo-Mínimo o (SHMM) y el SH, radica en que las concentraciones de feromonas se encuentran restringidas por intervalos dados. Adicionalmente, SHMM presenta las siguientes tres discrepancias con respecto al SH*:*

1. Se le permite únicamente a la mejor hormiga de la presente iteración o a la mejor hormiga global, depositar feromona. Por mejor hormiga se entiende aquella hormiga que haya producido el mejor resultado.

2. Los rastros de feromonas iniciales son acotados de forma tal que siempre se encuentren por encima de un mínimo y por debajo de un máximo.
3. Se implementa un mecanismo de modulación de feromonas, por medio del cual los rastros de feromonas son re-inicializados a un valor máximo permitido cada vez que el sistema se acerca a un estado de estancamiento.

- **Actualización del rastro de feromona en SHMM**

La estrategia de actualización del rastro de feromona en el SHMM es similar a la del SH. Una vez que todas las hormigas han construido un tour o una posible solución, el rastro de feromonas es actualizado aplicando el mismo mecanismo de evaporación que en SH:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in L \tag{1.12}$$

Seguido del depósito de feromonas según la regla:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{mejor} \tag{1.13}$$

Donde, $\Delta\tau_{ij} = 1/C^{mejor}$, si a la hormiga que se le permite depositar la feromona es la mejor hormiga global, o $\Delta\tau_{ij} = 1/C^{iteracion}$ si a la hormiga que se le permite depositar feromona es la mejor de la presente iteración; siendo $C^{iteracion}$ el largo del mejor tour de la presente iteración.

- **Re-inicialización del rastro de feromonas**

Como característica distintiva del SHMM, se tiene la re-inicialización del rastro de feromonas en el caso de que el algoritmo presente señales de estancamiento prematuro.

Para determinar el punto de estancamiento del algoritmo, se utiliza el factor de ramificación λ con λ = *0.05*. Gambardella y Dorigo (1996) definieron la media del

factor de ramificación λ como un indicador de la dimensión del espacio de búsqueda. A medida que transcurre el tiempo de ejecución del algoritmo, la media de este factor decrece hacia un valor pequeño, indicando un punto de estancamiento del algoritmo. El factor de ramificación λ_i se define como el número de conexiones partiendo del nodo *i* con valores de τ_{ij} mayores que:

$$\lambda(\tau_{\min} - \tau_{\max}) + \tau_{\min},$$

(1.14)

Donde,

$\tau_{i,\min} = \min_{j \in N_i}\{\tau_{ij}\}$, $\tau_{i,\max} = \max_{j \in N_i}\{\tau_{ij}\}$ y N_i es el conjunto de todos los nodos conectados con el nodo *i.*

De esta forma se tiene que, si se cumple:

$$\frac{\sum_{i \in V} \lambda_i}{n_G} < \varepsilon$$

(1.15)

Donde ε es un número positivo pequeño, se dice que el algoritmo se encuentra en un estado de estancamiento y se re-inicializa el rastro de feromona a un valor proporcional a la diferencia con el máximo valor permitido para las feromonas, específicamente:

$$\Delta\tau_{ij}(t) \propto (\tau_{\max}(t) - \tau_{ij}(t))$$

(1.16)

Esta estrategia se conoce como modulación del rastro de feromonas y permite que aquellas conexiones donde exista una mayor concentración de feromona reciban menos feromonas que aquellas conexiones con menor concentración. De esta forma, se aumenta la probabilidad de que conexiones más débiles participen en el proceso de construcción de soluciones y por tanto, se aumenta la capacidad de búsqueda general del algoritmo.

- **Límites máximos y mínimos de τ_{ij}**

Stützle & Hoos (1997) demostraron formalmente que la máxima cantidad de feromonas para un arco *(i, j)* cualquiera del grafo *G* se encuentra limitada o acotada asintóticamente según:

$$\lim_{t\to\infty} \tau_{ij}(t) = \tau_{ij} \leq \frac{1}{1-\rho}\frac{1}{f^*} \tag{1.17}$$

Donde f^* es el costo de la solución óptima teórica. Sin embargo, como el costo de la solución teórica es un valor desconocido, se utiliza un estimado de $f^* \approx f(\hat{x}(t))$ donde $f(\hat{x}(t))$ es el costo de la mejor solución global hasta el momento. Por tanto, se sabe que la máxima concentración de feromona cambia en cuanto se encuentre una nueva mejor solución global.

Más aún, el límite superior de las concentraciones de feromonas es una función dependiente del tiempo como se muestra a continuación:

$$\tau_{\max}(t) = (\frac{1}{1-\rho})\frac{1}{f(\hat{x}(t))} \tag{1.18}$$

Para el caso contrario, el límite inferior, se tiene:

$$\tau_{\min}(t) = \frac{\tau_{\max}(t)(1-\sqrt{\hat{p}n_G})}{(n_G/2-1)\sqrt{\hat{p}n_G}} \tag{1.19}$$

Donde 0 < $\hat{p}$ < *1* es la probabilidad de construir la mejor solución. Esto asegura que $\tau_{\min}(t) > 0$ en todo momento.

Para finalizar, cabe destacar que a pesar de ser una variante del algoritmo SH, este algoritmo incluye varias mejoras fundamentales que le confieren una estructura más compleja que la de su predecesor. En el anexo 1 se encuentra una implementación de este algoritmo en pseudo código.

1.5.3 Extensiones del SH

Los algoritmos basados en la metaheurística OCH explicados anteriormente, parten del algoritmo SH y lo que hacen es introducir modificaciones en la estructura general de este algoritmo para obtener mejoras importantes en su rendimiento.

Sin embargo, se han propuesto algoritmos basados en la metaheurística OCH pero que difieren sustancialmente del algoritmo SH, al presentar nuevas ideas y mecanismos que permiten conseguir un mayor rendimiento que el de dicho algoritmo o cualquiera de sus variantes. Algunos de estos algoritmos se describen a continuación.

1.5.3.1 Sistema de Colonias de Homigas (SCH)

El Sistema de Colonias de Hormigas (SCH) o *Ant Colony System* en inglés, fue desarrollado por Gambardella y Dorigo (1996) con el objetivo de mejorar el desempeño de su antecesor SH. Según Engelbretch (2007), el SCH difiere del SH en cuatro aspectos fundamentales:

1. Se utiliza una regla proporcional aleatoria distinta que aprovecha mejor la experiencia de búsqueda acumulada por las hormigas.
2. Se utiliza una regla de actualización de feromonas distinta en donde la evaporación y depósito de feromonas ocurre sólo en los arcos pertenecientes al mejor camino hasta el momento T^{bs}.
3. Se introducen actualizaciones de feromonas locales cuando una hormiga utiliza un arco *(i, j)* para moverse del nodo *i* al nodo *j*. Dicha hormiga disminuirá o evaporará la feromona de dicho arco para aumentar la probabilidad de explorar caminos alternativos.
4. Se utilizan listas de candidatos para favorecer nodos específicos.

- **Construcción de caminos**

Según Gambardella (1996) la construcción de caminos o tours, por parte de una hormiga *k* se realiza de acuerdo con la siguiente regla proporcional aleatoria:

$$j = \arg\max_{l \in N_i^k} \{\tau_{il}[n_{il}]^\beta\}, \text{ si } q \leq q_0; \quad (1.20)$$

$$j = J, \text{ sino; con } J \in N_i^k$$

Donde q es una variable aleatoria uniformemente distribuida en el intervalo [0, 1], q_0 es un parámetro definido a priori que toma valores en el rango [0, 1] y J es un nodo seleccionado de acuerdo a una variable aleatoria cuya distribución viene dada por la siguiente ecuación:

$$p(k)_{ij} = \frac{\tau_{ij}[\eta_{ij}]^\beta}{\sum_{l \in \mathrm{N}_i^k} \tau_{il}[\eta_{il}]^\beta}, \text{ si } j \in \mathrm{N}_i^k \quad (1.21)$$

Analizando las ecuaciones anteriores se tiene que, con probabilidad q_0, una hormiga hace el mejor movimiento posible como lo indican los rastros de feromonas y la información heurística, según la ecuación (1.20). Para este caso, la hormiga está explorando el conocimiento adquirido a través de las iteraciones anteriores, creando un sesgo hacia aquéllos nodos conectados a través de arcos con poco peso y con gran cantidad de feromonas. Por otra parte, y con probabilidad $(1 - q_0)$ la hormiga explora los arcos del grafo de forma sesgada según lo indica el rastro de feromonas.

Por lo tanto, el parámetro q_0 se utiliza o bien para aprovechar el conocimiento adquirido o para favorecer la exploración de nuevos caminos. A medida que el parámetro q_0 es más pequeño, se explota menos la información contenido en los mejores arcos hasta el momento y se favorece más la exploración de nuevos caminos o tours. Cabe destacar que cuando se cumple que $q > q_0$, la regla proporcional aleatoria es la misma que la del algoritmo SH con $\alpha = 1$.

- **Actualización del rastro de feromona global en el SCH**

A diferencia de SH, en éste algoritmo se le permitirá únicamente a aquella hormiga que haya construido el camino más corto depositar feromona en los arcos de dicho camino luego de cada iteración según la ecuación:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \quad \forall(i,j) \in T^{bs} \tag{1.22}$$

Donde, $\Delta\tau_{ij}^{bs} = 1/C^{bs}$

El parámetro ρ indica, como ya es usual, la tasa de evaporación de feromona. A diferencia de las ecuaciones utilizadas para actualizar las feromonas en el SH, en el SCH la cantidad de feromona depositada es reducida por el factor *p* lo que da por resultado que la cantidad de feromona depositada sea un promedio ponderado entre la cantidad de feromona que se encontraba anteriormente y la cantidad de feromona a ser depositada.

- **Actualización del rastro de feromona local en el SCH**

Adicionalmente a la actualización de feromonas globales, este algoritmo hace uso de una estrategia de actualización de feromonas locales. Esta estrategia consiste en que una hormiga, inmediatamente luego de cruzar un arco *(i, j)* durante la construcción de un camino, actualiza las feromonas según la ecuación:

$$\tau_{ij} = (1-\xi)\tau_{ij} + \xi\tau_0 \tag{1.23}$$

Donde, $\xi \in (0,1)$ y $\tau_0 < \varepsilon$, con ε una constante pequeña positiva.

El efecto esta regla de actualización de feromonas locales se puede resumir como la reducción del rastro de feromona τ_{ij} cada vez que una hormiga "camina" sobre el arco *(i, j)*. De esta forma, se disminuye la probabilidad de que dicho arco sea seleccionado por otra hormiga en el futuro, favoreciendo la exploración de nuevos caminos que aún no hayan sido visitados por la Colonia de Hormigas. Adicionalmente, Gambardella y Dorigo explican que mediante este mecanismo, se evita que el algoritmo se estanque en soluciones parciales y subóptimas.

- **Lista de nodos candidatos**

Sea N_i^k la lista de nodos candidatos y $n_l < |N_i^k|$ la cantidad de nodos en dicha lista (n_1 es un parámetro establecido a priori). Los n_1 nodos más cercanos, ya sea en distancia o costo al nodo *i*, serán incluidos en la lista de candidatos N_i^k y ordenados descendentemente por distancia. De esta forma, cuando una hormiga seleccione el próximo nodo a visitar, se seleccionará al primer candidato de la lista que representa la alternativa más prometedora.

Finalmente, en el anexo 2 se encuentra una propuesta de pseudo código para el algoritmo SCH.

1.5.3.2 Hormiga-Q

Hormiga-Q o *Ant-Q* en inglés, es una versión anterior del SCH basada en ideas de otras ramas de la Inteligencia Artificial, i.e.: aprendizaje-Q (*Q-learning*), que es una técnica de aprendizaje dirigida, que funciona cuando un agente "aprende" valores de tuplas (estado, acción). Así, el valor *Q(e,a)* se define como el valor esperado de la suma de futuros beneficios obtenidos al realizar una acción *a* desde el estado *e*, e inmediatamente siguiendo una política óptima. Una vez aprendidos dichos valores, la acción óptima a realizar por el agente desde un estado cualquiera es aquella con mayor valor Q.

Para el caso de las Colonias de Hormigas, el algoritmo Hormiga-Q, propuesto por Gambardella & Dorigo (1995) deja de lado la noción de feromonas y la reemplaza con valores Hormiga-Q o valores-HQ. El objetivo entonces es el aprendizaje de dichos valores-HQ, de forma tal que el descubrimiento de buenas soluciones se vea favorecido probabilísticamente.

- **Regla proporcional aleatoria para Hormiga-Q**

Similar al SCH, la regla de decisión para los movimientos de las hormigas se define como:

$$j = \arg\max_{u \in N_i^k(t)} \{\mu_{iu}^{\alpha}(t) n_{iu}^{\beta}(t)\}, \text{ si } q \leq q_0; \tag{1.24}$$

$$j = J, \text{ sino; con } J \in N_i^k$$

Los parámetros α, β indican la importancia de los valores-HQ aprendidos, es decir, de μ_{ij}; mientras que n_{ij} provee información heurística. Es claro entonces que los valores-HQ expresan cuán útil u óptimo es moverse al nodo *j* a partir del nodo *i*. El valor *J* se selecciona con la ya familiar regla aleatoria proporcional:

$$p_{ij}^k(t) = \frac{\mu_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{u \in \mathrm{N}_i^k(t)} \mu_{iu}^{\alpha}(t) n_{iu}^{\beta}(t)}, \text{ si } j \in \mathrm{N}_i^k(t), \tag{1.25}$$

0 sino

- **Aprendizaje de los valores Hormiga-Q**

  El aprendizaje de los valores-HQ se implementa utilizando la siguiente regla:

$$\mu_{ij}(t+1) = (1-\rho)\mu_{ij}(t) + \rho\left(\Delta\mu_{ij}(t) + \gamma \max_{u \in N_j^k(t)} \{\mu_{ju}(t)\}\right) \tag{1.26}$$

Donde ρ se refiere al factor de reducción, análogo a la evaporación de feromonas, y γ es la velocidad de aprendizaje. Cabe destacar que en el caso donde $\gamma = 0$, la regla se reduce a la ecuación de actualización global de feromonas del algoritmo SCH. Esta regla se aplica a cada hormiga *k* inmediatamente después de seleccionar un nuevo nodo *j*.

En el presente Capítulo se describen algunos algoritmos derivados del comportamiento de Colonias de Hormigas en búsqueda de alimento. Cabe destacar que existen otros algoritmos que no fueron incluidos en el presente trabajo. En el próximo Capítulo se planteará un Framework para centraliza el funcionamiento de los algoritmos descritos en este Capítulo. Por último, se comparará la eficiencia de los algoritmos del Framework contra otros tipos de algoritmos en optimización.

CAPÍTULO II: MARCO APLICATIVO

2.1 Introducción

En este capítulo se presenta en detalle el diseño y funcionamiento del Framework basado en algoritmos de Colonias de Hormigas, objeto del presente trabajo. Se explica la motivación para crear dicho Framework, los objetivos planteados en su realización, la metodología, técnicas y herramientas de programación empleadas durante su construcción.

Luego, se presentarán una serie de experimentos o pruebas de desempeño que permitirán evaluar el funcionamiento del software elaborado. A través de dichas pruebas se podrá comparar la calidad de las soluciones obtenidas mediante los algoritmos de Colonias de Hormigas contra otras soluciones obtenidas a través de métodos tradicionales como *backtracking,* y métodos estocásticos como los algoritmos genéticos.

2.2 Planteamiento del Problema

Una vez realizada la investigación en profundidad sobre los modelos computacionales inspirados en Colonias de Hormigas naturales y los algoritmos de optimización derivados de dichos modelos, se realizó una investigación acerca de las herramientas que permiten hacer uso de estos algoritmos en algún ambiente de programación, sin encontrarse ninguna solución que involucre la utilización de todos los algoritmos presentados en este trabajo. Se plantea entonces el deseo de centralizar todo el conocimiento adquirido sobre modelos de optimización por Colonias de Hormigas en un solo Framework, que sirva de herramienta tanto para el trabajo de investigación de la comunidad científica como para la enseñanza en el salón de clase.

2.3 Objetivos

2.3.1 Objetivo General

- Analizar los modelos computacionales derivados del comportamiento de las Colonias de Hormigas naturales con el fin de construir una herramienta basada en software libre, que incorpore dichos modelos en la resolución de problemas de optimización numérica.

2.3.2 Objetivos Específicos

- Evaluar el desempeño de los modelos de Colonias de Hormigas en la resolución de distintos problemas de optimización numérica.
- Construir un Framework en software libre basado en modelos de Colonias de Hormigas naturales, que integre los distintos algoritmos de optimización y que pueda ser utilizado por la comunidad científica en general.

2.4 Propuesta de Framework

En esta sección se explica, a grandes rasgos, la propuesta del Framework objeto del presente trabajo. En posteriores secciones se detallará su estructura y funcionamiento. Cabe destacar que el software aquí descrito puede ser descargado libremente por cualquiera a través de la siguiente URL en Internet: http://sourceforge.net/projects/libai/

2.4.1 Consideraciones de construcción y funcionamiento

El Framework será construido utilizando el paradigma de programación orientada a objetos, y le proveerá al programador de un *API* (*Application Programming Interface o* Interfaz de Programación de Aplicación) en el cual se encontrarán implementados los principales algoritmos basados en la metaheurística OCH y sus variantes más importantes explicadas en el Capítulo I. El Framework poseerá, como es natural en éste tipo de Software, una librería con funcionalidades básicas y comunes a los algoritmos de Colonias de Hormigas anteriormente revisados. Dichas estructuras básicas serán el *core* o corazón del sistema, el cual no debe ser modificado directamente por el usuario sino que el Framework permitirá, por medio de diversos mecanismos, extender su comportamiento natural para adaptarse a necesidades particulares.

Por lo tanto, el Framework deberá ser fácilmente extensible y adaptable con el fin de incorporar nuevos algoritmos o variantes basadas en OCH que puedan surgir en el futuro, por lo que el mismo se diseñará utilizando herramientas y técnicas de la Ingeniería de Software considerada como estado del arte.

Por último cabe destacar que las distintas implementaciones de los algoritmos de Colonias de Hormigas contenidas en el Framework basarán su funcionamiento en la construcción de caminos desde un nodo origen hacia un nodo destino, en analogía con la búsqueda de rutas entre un nido y una fuente de alimento, tal y como se explicó en el Capítulo I.

2.4.2 Consideraciones técnicas

Como consideración técnica principal cabe destacar que el Framework se construirá sobre la última versión disponible a la fecha del lenguaje de programación orientado a objetos JAVA, en su versión estándar, en concreto: JAVA SE 1.6. El Framework no requerirá ninguna extensión o librería externa que no se incluya con la distribución estándar del lenguaje tal y como se indica en el sitio web http://java.sun.com/.

2.5 Diseño del Framework

En esta sección se detalla la arquitectura del Framework, sus principales componentes y funcionamiento general. Es importante destacar que se empleará el lenguaje UML (*Unified Modeling Language* o Lenguaje de Modelado Unificado por sus siglas en inglés), para la construcción de los diagramas de estructura y comportamiento que se consideren necesarios.

2.5.1 Arquitectura del Framework

La arquitectura del Framework se divide en dos paquetes de clases, a saber: un paquete de clases principales o *Algoritmos* del Framework y un paquete de clases auxiliares o *Comunes*. Dentro del lenguaje de programación Java, esta estructura junto con sus clases correspondientes, se organizan en dos paquetes o *packages* como se muestra en la Figura 2.1.

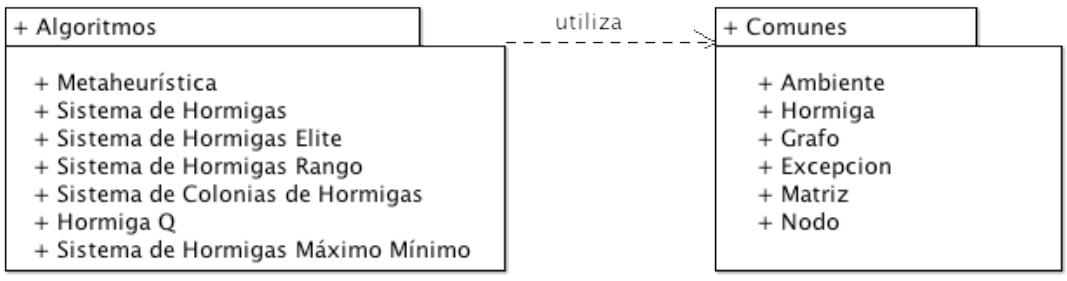

Figura 2.1. Diagrama de paquetes del Framework

A continuación se detallan cada uno de los paquetes anteriormente mencionados.

2.5.1.a *Algoritmos* del Framework

En el paquete *Algoritmos* del Framework se define la Metaheurística OCH como una clase abstracta, y cada uno de los algoritmos de Colonias de Hormigas como una clase concreta que extiende dicha clase abstracta. Esta estructura se presenta en el diagrama de clases de la Figura 2.2.

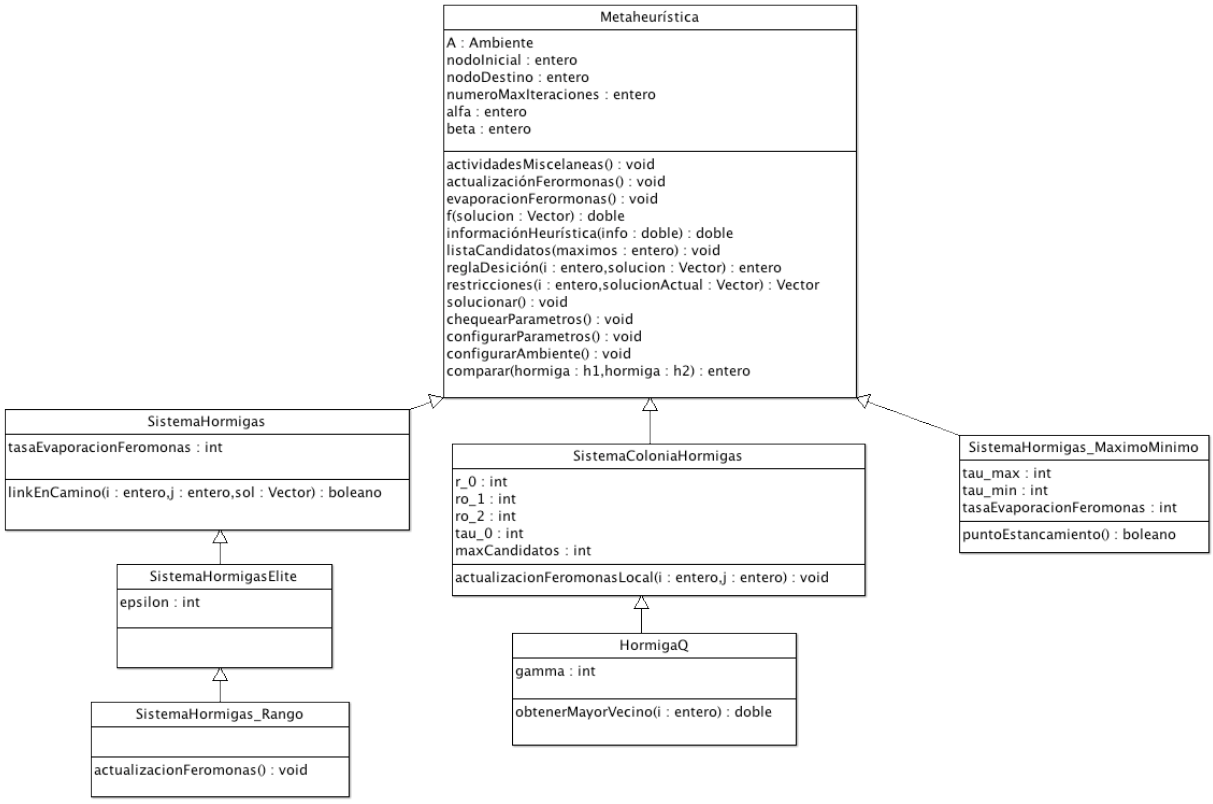

Figura 2.2. Arquitectura del Framework: *Algoritmos* del Framework

Se observa que en el diagrama de clases de la Figura 2.2, el Framework se compone de una clase abstracta principal llamada *Metaheurística*. Dicha clase representa el mayor nivel de abstracción del Framework, y pretende ser la base de su funcionamiento. Para tal fin la clase contiene los siguientes atributos:

- *Ambiente*: instancia de la clase Ambiente perteneciente al conjunto de clases auxiliares que se explican en la próxima sección.

- *Nodo Inicial*: indica el índice del nodo desde donde se comenzarán a construir soluciones.

- *Nodo Destino*: Indica el índice del nodo destino, a dónde se debe llegar para considerar que un camino representa una solución.

- *Número Máximo de Iteraciones*: indica el máximo número de iteraciones permitidas hasta la parada de un algoritmo. Éste será el criterio principal de parada utilizado por los algoritmos del Framework.

- *Alfa y Beta*: son parámetros que se utilizan al momento de construir soluciones, específicamente, al momento de seleccionar un nodo a visitar. *Alfa* representa la importancia que se le dará a la información contenida en el rastro de feromonas, mientras que *Beta* representa la importancia que se le dará a la información heurística.

Los atributos anteriores se encuentran en el máximo nivel de abstracción ya que las distintas implementaciones concretas de los algoritmos de Colonias de Hormigas necesitarán, sin importar cual sea el problema a resolver, hacer uso de éstos para su correcto funcionamiento.

Adicionalmente, la clase abstracta *Metaheurística* define los siguientes métodos abstractos:

- *evaporaciónFeromonas*: se encarga de implementar el mecanismo de evaporación del rastro de feromonas.

- *actualizaciónFeromonas:* se encarga de implementar el mecanismo de actualización del rastro de feromonas.

- *reglaDecisión(entero i)* $\rightarrow$ *entero:* lo utilizan las hormigas artificiales para decidir, a partir de un nodo *i*, el próximo nodo a visitar. Cabe destacar que éste método implementa la lógica más importante de cualquier algoritmo, ya que

define como una hormiga construirá soluciones y por lo tanto, la calidad de las mismas.

- *solucionar*: éste es el cuerpo del algoritmo. Aquí se itera sobre el grafo problema y se despliegan las hormigas sobre el mismo en búsqueda de soluciones. Adicionalmente, se implementa la lógica necesaria para sincronizar las llamadas a los distintos métodos: *evaporaciónFeromonas*, *actualizaciónFeromonas*, etc.

- *listaCandidatos*: genera, por cada nodo en el grafo, una lista de candidatos de acuerdo a una heurística específica del problema.

- *actividadesMiscelaneas*: esta función se llama en cada iteración y provee un puente para implementar actividades centralizadas dependientes de cada problema.

- *informaciónHeurística*: calcula la información heurística. Este método debe ser implementado a nivel del problema.

- *chequearParametros*: esta función se encarga de chequear que existan todos los parámetros necesarios para la ejecución de un algoritmo. En caso de que existan, permite la ejecución del mismo. En caso contrario, arroja una excepción que debe ser controlada por el código cliente.
- *restricciones:* para cada nodo *i* del grafo, esta función genera una lista con aquellos nodos que sean posible visitar partiendo del nodo *i*, según las restricciones del problema a resolver.

Esta clase también provee los siguientes métodos concretos:

- *configurarParametros*: esta función se invoca al principio de la ejecución de cualquier algoritmo y su responsabilidad es configurar los parámetros particulares del algoritmo en cuestión. Por ejemplo, para el algoritmo de Sistema de Hormigas, esta función debe configurar valores para los parámetros alfa y beta.

- *configurarAmbiente*: configura el ambiente de ejecución: hormigas, feromonas y grafo.
- *f*: recibe un vector solución y devuelve su longitud.
- *comparar*: recibe dos hormigas y compara sus soluciones según la función *f.*

Los métodos explicados anteriormente surgen de la metaheurística OCH explicada en el Capítulo I y componen el corazón del sistema. Cualquier algoritmo del Framework deberá implementar cada uno de estos métodos al extender la clase abstracta *Metaheurística*.

Específicamente, el Framework implementa los siguientes algoritmos de Colonias de Hormigas, los cuales fueron explicados con detalle en el Capítulo I y que de aquí en adelante se denominarán como implementaciones de algoritmos concretas:

- Sistema de Hormigas
    - Sistema de Hormigas Élite
    - Sistema de Hormigas basadas en Rango
- Sistema de Hormigas Máximo-Mínimo.
- Sistema de Colonias de Hormigas
    - Hormiga Q

2.5.1.b Clases auxiliares

Adicionalmente, el Framework establece un conjunto de clases auxiliares y comunes a todos los algoritmos, las cuales se encargarán de llevar la información del entorno al momento de ejecución de un algoritmo cualquiera. Estas clases son

responsables de la información relacionada al problema a resolver, las hormigas artificiales y el rastro de feromonas.

El siguiente diagrama de clases muestra la estructura de dichas clases auxiliares:

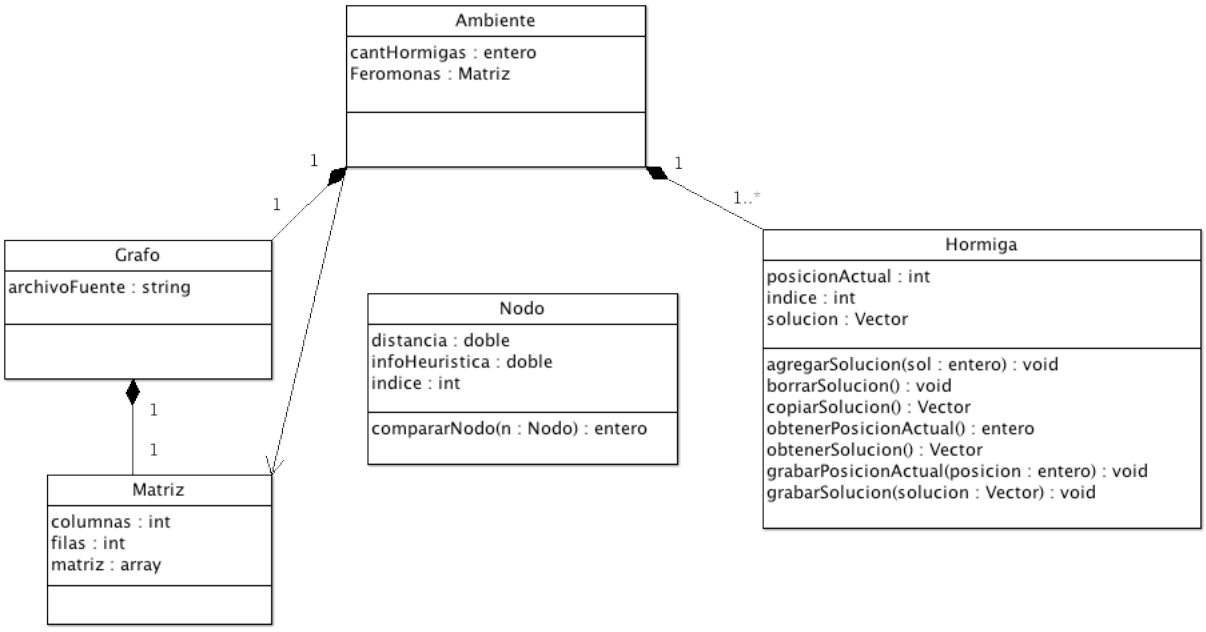

Figura 2.3. Arquitectura del Framework: Clases Auxiliares

Como se puede observar en el diagrama anterior, las clases encargadas de llevar información del entorno son las siguientes:

- *Ambiente*: esta clase contiene un arreglo de objetos tipo Hormiga, un Grafo y una matriz de Feromonas.

- *Hormiga*: esta clase implementa a una hormiga artificial. Se encarga de llevar el estado de la hormiga, específicamente: el índice del nodo en el que se encuentran en un instante cualquiera y un vector con los del grafo correspondientes a una solución, ya sea parcial o total de esta hormiga.

- *Grafo*: implementa las restricciones de un problema particular en una matriz, típicamente cargada desde un archivo plano en memoria.

- *Matriz*: es una clase que implementa un arreglo bidimensional o matriz como un arreglo unidimensional, ya que un arreglo de éste tipo es mucho más eficiente que implementar una matriz directamente tal y como la provee el lenguaje de programación. Esta clase se utiliza dentro de la clase Grafo para almacenar la información del problema a resolver, y dentro de la clase ambiente para modelar la matriz de feromonas. Adicionalmente, provee métodos especializados para cargar una matriz de adyacencia desde un archivo de texto plano y prepararla para su posterior utilización por parte de los algoritmos de Colonias de Hormigas.

- *Nodo*: representa un nodo particular del grafo. Esta clase se utiliza principalmente al momento de construir la lista de candidatos, ya que permite comparar un nodo contra otro de forma muy eficiente y sencilla de implementar.

Cabe destacar que la clase abstracta *Metaheurística*, explicada en la sección 2.5.1.b, poseerá siempre una instancia de la clase *Ambiente*. De esta forma, un algoritmo cualquiera tendrá acceso a la información necesaria para la resolución del problema en cuestión.

Adicionalmente, esta estructura permite mantener la información del problema a resolver separada de la lógica de resolución; esto redunda en una mayor abstracción de los componentes del sistema facilitando su posterior extensión.

2.5.1.c Extensión del Framework

Como se mencionó anteriormente en la propuesta de Framework (Sección 2.4), el mismo será fácilmente extensible para incorporar alguna nueva variante de los algoritmos de Colonias de Hormigas que pueda surgir en el futuro. La metodología para extender el Framework se describe a continuación.

- **Incorporación de un nuevo algoritmo**

Si el algoritmo a implementar difiere sustancialmente de algún algoritmo concreto dentro del Framework, se debe extender la clase abstracta *Metaheurística* tal y como se indica en la Figura 2.6.

Como se muestra en el diagrama, la *Clase_Ejemplo* implementa un nuevo algoritmo de Colonias de Hormigas. En dicha clase se tiene la libertad de implementar cualquier cantidad de atributos y métodos que se consideren necesarios para el buen funcionamiento del algoritmo. En el diagrama dichos atributos se denotan como: atributo_1, atributo_2,…, atributo_n; y los métodos de igual forma: metodo_1, metodo_2,…, metodo_n.

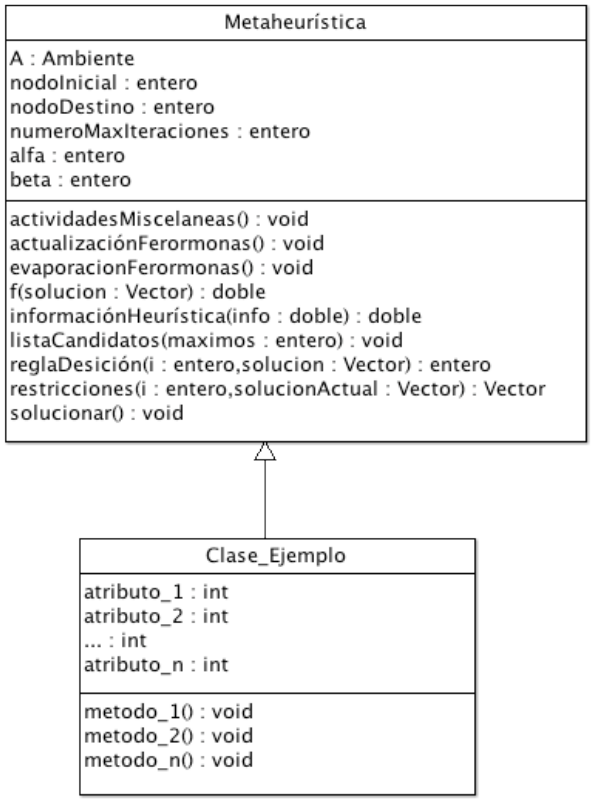

Figura 2.6. Extensión del Framework: Incorporación de un nuevo algoritmo

Sin embargo, y como ya se mencionó anteriormente, se deben implementar obligatoriamente los siguientes métodos: *evaporaciónFeromonas, actualizaciónFeromonas, reglaDesición* y *solucionar.* Opcionalmente, y de acuerdo a las características del algoritmo a implementar, los siguientes métodos se pueden

implementar o declarar como finales: *listaCandidatos, actividadesMiscelaneas, informaciónHeurística*. Todos los métodos anteriores se explican en la sección 2.5.1.1 del presente documento.

- **Extensión de un algoritmo concreto**

Si la variante a implementar parte de algún algoritmo concreto del Framework, se debe extender este algoritmo y sobrescribir los métodos que sean necesarios. El diagrama de la Figura 2.7 muestra el caso hipotético en que el algoritmo a extender sea el de Sistema de Hormigas.

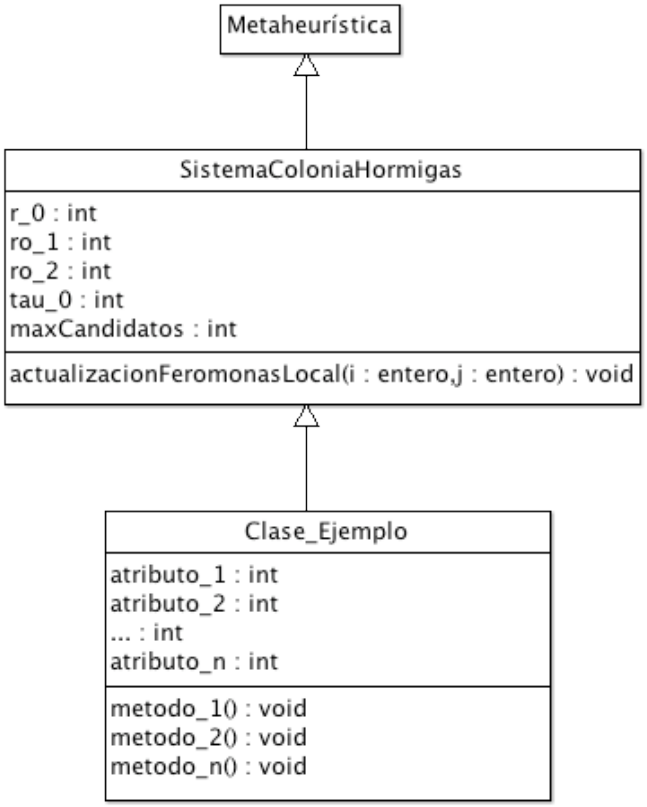

Figura 2.7. Extensión del Framework: Incorporación de un nuevo algoritmo a partir de un algoritmo existente

Al igual que en el caso anterior, la *Clase_Ejemplo* puede implementar cualesquiera atributos y métodos que se consideren necesarios, con la diferencia de

que esta clase hereda todas aquellos métodos y atributos de la clase que se está extendiendo. Este mecanismo se empleó en la construcción del Framework. Por ejemplo, el algoritmo de Sistema de Hormigas Élites es una extensión del algoritmo de Sistema de Hormiga (ver figura 2.2).

Es importante destacar que tanto para el caso de la construcción de un nuevo algoritmo o la extensión de un algoritmo concreto, se debe respetar el uso de los objetos de la clases comunes y utilizarlos para representar la información del problema a optimizar, de forma tal de mantener la consistencia entre distintas implementaciones.

2.5.2 Funcionamiento del Framework

Una vez explicadas las estructuras principales del Framework, en esta sección se detalla como éstas se combinan e interactúan para resolver un problema de optimización cualquiera.

2.5.2.a Diagrama de Colaboración

Un diagrama de colaboración ilustra las relaciones e interacciones entre los objetos de software de un sistema en el lenguaje UML. El diagrama de colaboración del Framework se presenta en la siguiente figura.

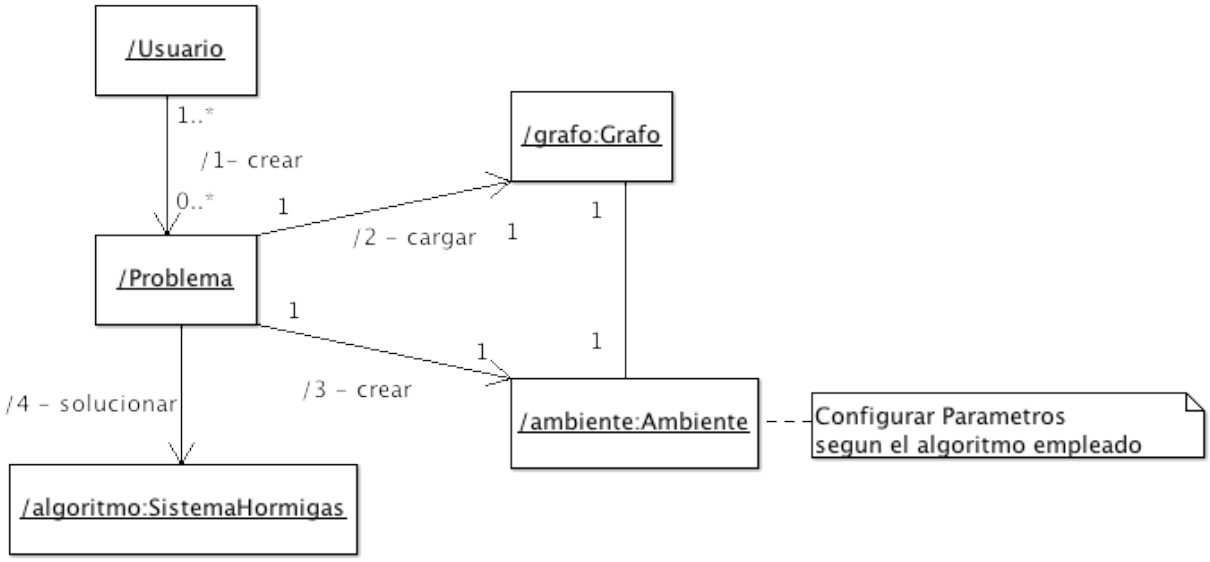

Figura 2.8. Diagrama de Colaboración del Framework

En la Figura 2.8 se observan los siguientes objetos: Usuario, Problema, Grafo, Ambiente y Algoritmo. Además se entiende que un objeto Usuario se compone de 0 o más objetos Problema, y un objeto Ambiente se compone exactamente de un objeto Grafo. Cabe destacar que el objeto Usuario no se corresponde con un objeto particular dentro del Framework, ya que se utiliza en este diagrama como un objeto virtual con el único fin de ilustrar que es un usuario quién debe inicializar el proceso de resolución de uno o más problemas. Por otra parte, el objeto Problema es aquél que contiene las restricciones y la lógica propia del problema a optimizar.

Por otra parte, el flujo de mensajes se explica de la siguiente manera:

- El objeto Usuario envía el mensaje */1-crear* al Problema, inicializando la resolución de un problema particular.
- El objeto Problema crea y carga un objeto Grafo, lo cual se denota en el diagrama a través del mensaje */2-cargar*.
- Nuevamente, el objeto Problema crea un objeto Ambiente a través del mensaje */3-crear*.
- Una vez el objeto Problema ha creado un Grafo y un Ambiente, el mismo está listo para ser resuelto, por lo tanto envía un mensaje */4-solucionar* al objeto Algoritmo.

Es importante resaltar que el diagrama de colaboración anterior permanece inalterable ante la instancia particular del algoritmo de Colonias de Hormigas que se emplee. Por lo tanto, el objeto Algoritmo se puede cambiar por cualquiera de los siguientes: Sistema de Hormigas, Sistema de Hormigas Élite, Sistema de Hormigas Rango, Sistemas de Colonias de Hormigas, Hormiga Q, y Sistema de Hormigas Máximo-Mínimo. Más aún, éste objeto también representa un potencial nuevo algoritmo que extienda los algoritmos anteriores.

2.5.2.b Diagramas de Secuencia

En el lenguaje UML un diagrama de secuencia ilustra la secuencia de acciones que ocurren en un sistema. El diagrama de secuencia del Framework se presenta en la Figura 2.9.

Como se observa en el diagrama 2.9, el Usuario inicia el Framework creando un Problema. Luego éste cargará un Grafo (bien sea de memoria o directamente desde código) e instanciará un objeto Ambiente, el cual contendrá al Grafo antes mencionado y los demás parámetros necesarios para la ejecución del Algoritmo.

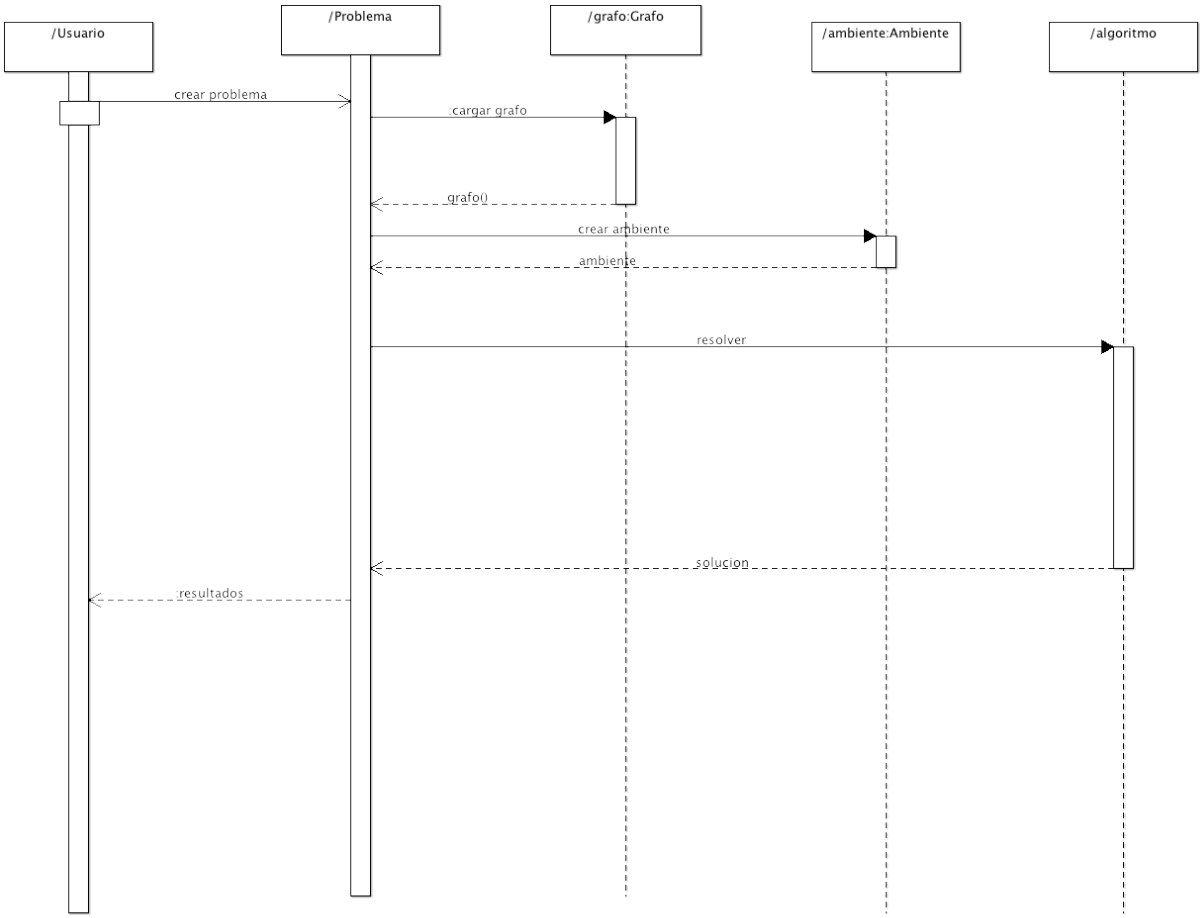

Figura 2.9. Diagrama de Secuencia del Framework

Del diagrama se puede concluir que las actividades de inicialización del Problema, carga del Grafo y Ambiente son las que consumen una menor cantidad del tiempo total, si bien la carga de un grafo será proporcional al tamaño del mismo.

Como es de esperarse, la ejecución del algoritmo *per se* será la que consuma mayor parte del tiempo total de vida del sistema, aunque al igual que en la carga de un grafo, la ejecución del algoritmo será una función del espacio de búsqueda del problema y los parámetros del mismo, por ejemplo: cantidad total de hormigas, número de iteraciones, etc.

- **Diagramas de Secuencia de Algoritmos Concretos**

A continuación se presentan los diagramas de secuencia de los algoritmos concretos contenidos en el Framework. Con éstos se pretende ilustrar detalladamente la división de responsabilidades entre el código cliente o Problema y el código del Framework.

La Figura 2.10, muestra el diagrama de secuencia del algoritmo de Sistema de Hormigas. Como se puede observar, las siguientes actividades son responsabilidad del código cliente o Problema:

1. Configuración de parámetros (en este caso alfa y beta).
2. Invocar al método de resolución del algoritmo.
3. Calcular los nodos permitidos.
4. Enviar información heurística de un nodo.
5. Evaluar una posible solución y recibir el resultado final.

Por su parte y como ya se ha mencionado anteriormente, el objeto del algoritmo de Sistema de Hormigas se encargará de construir soluciones y sincronizar los eventos de evaporación y actualización de feromonas. En general, la división de responsabilidades antes descrita aplicará para todos los algoritmos, a excepción de algunos casos particulares.

El diagrama de la Figura 2.11 muestra una extensión del funcionamiento del Sistema de Hormigas, a saber: el Sistema de Hormigas Élite. Como se puede observar en el diagrama, el Sistema de Hormigas Élite reutiliza la mayor parte de la

lógica del Sistema de Hormigas. La diferencia fundamental en éste caso es el reforzamiento de los arcos del mejor *tour* o camino construido hasta el momento. Esta actividad se lleva a cabo en el procedimiento de actividades misceláneas del objeto Sistema de Hormigas Élite.

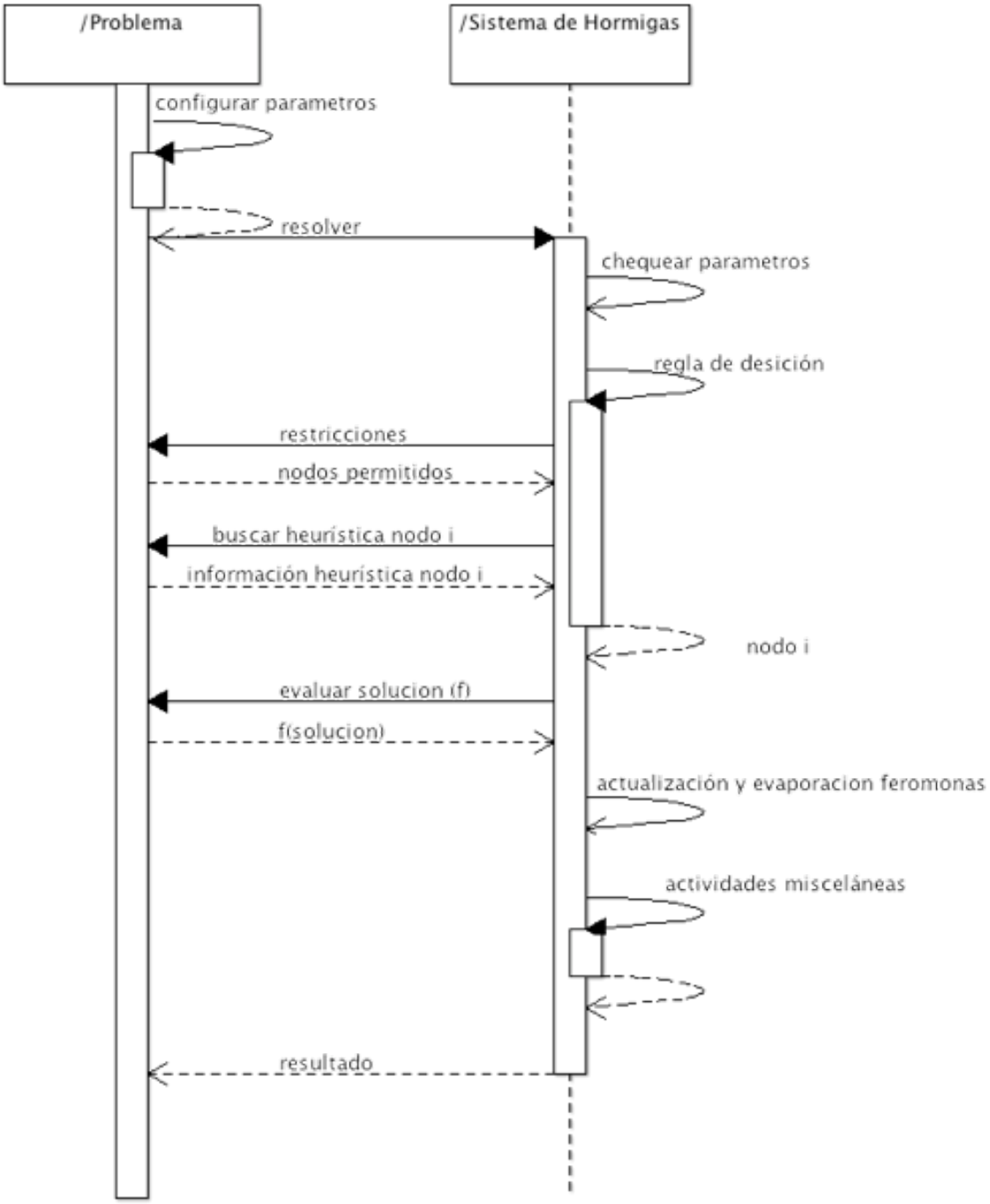

Figura 2.10. Diagrama de Secuencia del Sistema de Hormigas

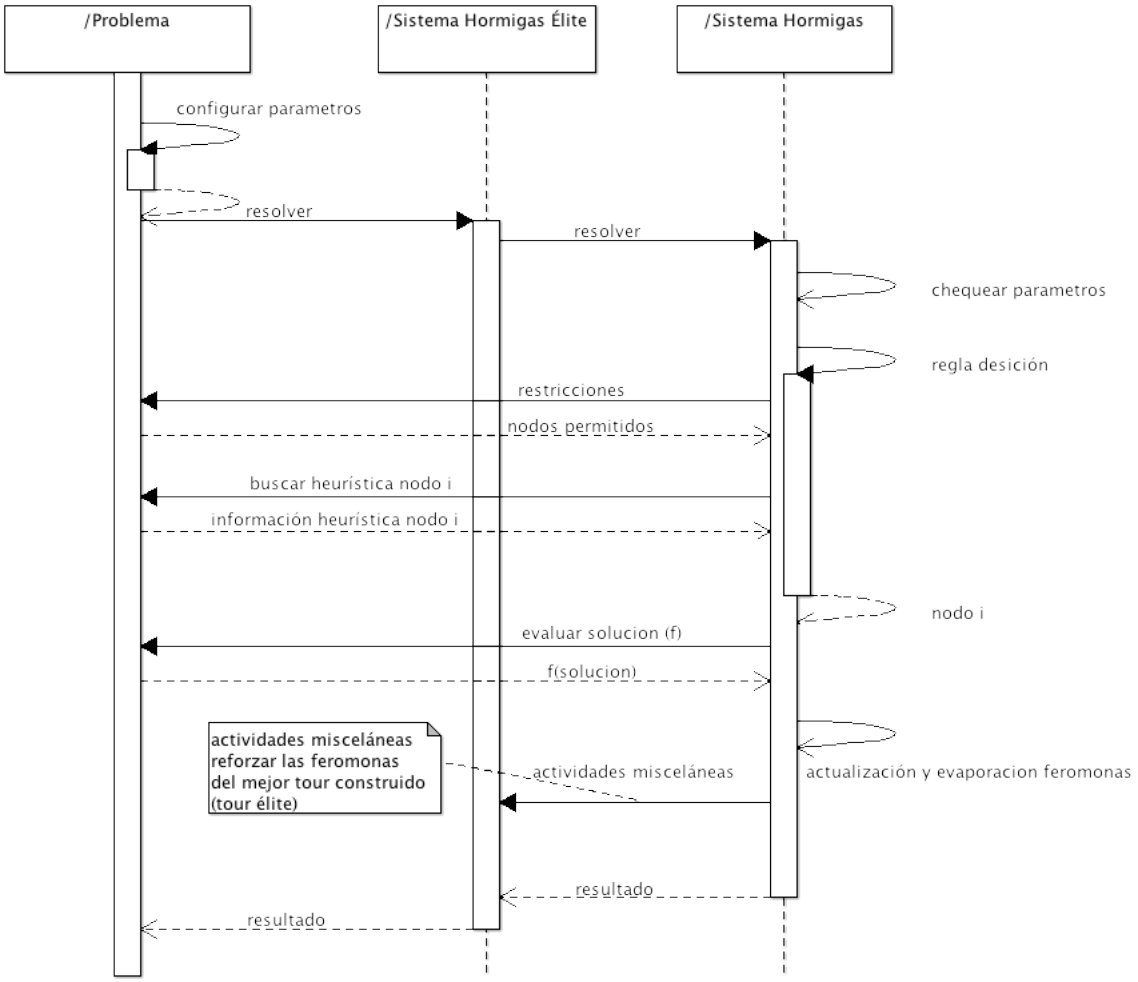

Figura 2.11. Diagrama de Secuencia del Sistema de Hormigas Élite

Es importante mencionar que un diagrama similar al de la Figura 2.11 aplica para el caso del Sistema de Hormigas Rango, con la diferencia fundamental que éste extiende el comportamiento del Sistema de Hormigas Élite y varía únicamente en la implementación del procedimiento de actualización de feromonas, tal y como se explicó en el Capítulo I.

La Figura 2.12 presenta el diagrama de secuencia del Sistema de Colonias de Hormigas. La única diferencia con los algoritmos anteriores reside en el hecho de que éste invoca una función para la construcción de una lista de nodos candidatos del lado del Problema.

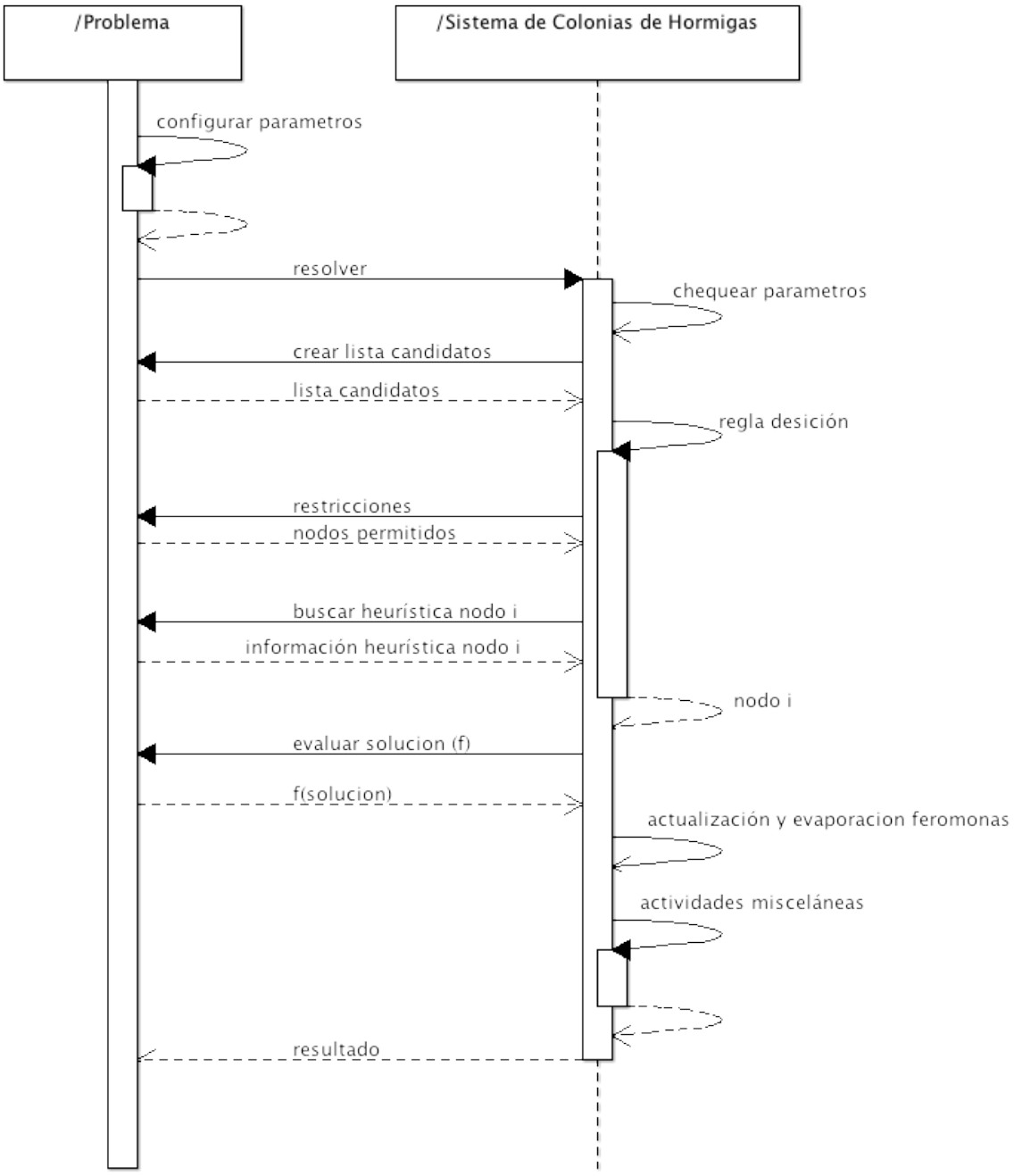

Figura 2.12. Diagrama de Secuencia del Sistema de Colonias de Hormigas

Es importante destacar que el diagrama anterior aplica igual para el algoritmo HormigaQ, ya que éste extiende del Sistema de Colonias de Hormigas y modifica únicamente la forma en que se hace la actualización de feromonas.

Por último se presenta el diagrama de secuencia del SH Máximo Mínimo en la Figura 2.13

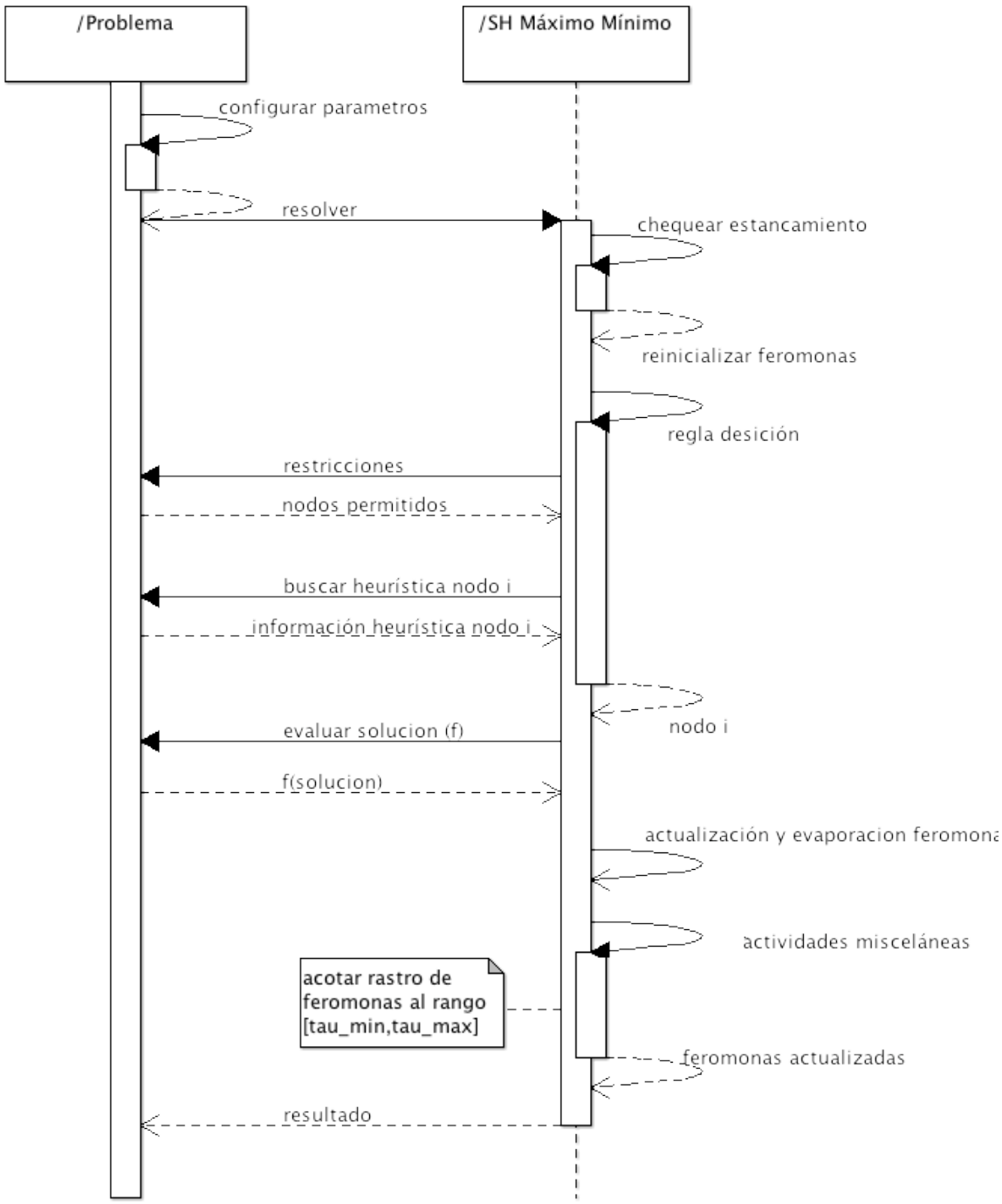

Figura 2.13. Diagrama de Secuencia del Sistema de Hormigas Máximo Mínimo

El algoritmo descrito en la Figura 2.13 actúa de forma similar a los algoritmos anteriores pero introduce nuevas funciones como el chequeo de estancamiento, reinicialización y acotación de feromonas. Estas funciones se explicaron con detalle en el Capítulo I del presente documento.

2.6 Aplicación Ejemplo

Como ejemplo práctico del funcionamiento del Framework, se describe en esta sección una aplicación que provee una interfaz al usuario y que hace uso de las librerías del Framework para resolver el problema de encontrar la ruta más corta entre el nido y la fuente de alimento en un grafo conexo cualquiera. Es importante destacar que esta aplicación sirve tan sólo como una interfaz que le permite al usuario no programador hacer uso de los algoritmos contenidos en el Framework para resolver únicamente el problema de las rutas más cortas. Para resolver otros problemas se debe hacer uso del Framework como se explica en la sección 2.5. En la Figura 2.14 se muestra dicha aplicación.

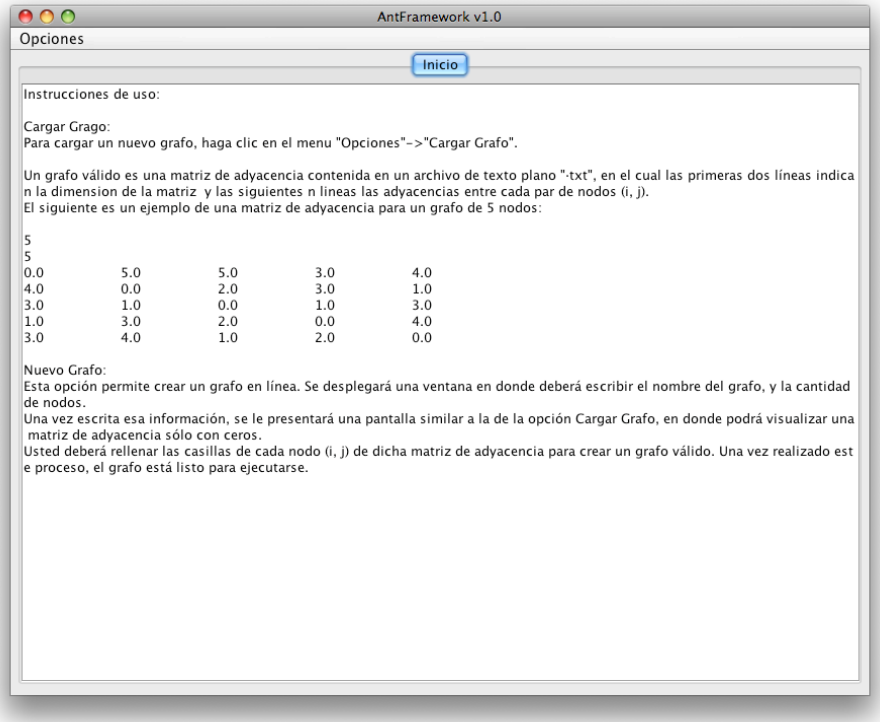

Figura 2.14. Aplicación Ejemplo

La aplicación permite al usuario cargar un grafo almacenado en memoria o crear un nuevo grafo. Un grafo almacenado en memoria es simplemente un archivo de texto plano que contiene la matriz de adyacencia de dicho grafo. El siguiente es un ejemplo de una matriz de adyacencia almacenada en memoria:

```
5
5
0   5   5   3   4
4   0   2   3   1
3   1   0   1   3
1   3   2   0   4
3   4   1   2   0
```

Las primeras dos líneas indican la dimensión de la matriz de adyacencia, en este caso 5 filas por 5 columnas. En general los algoritmos de CH utilizarán matrices de adyacencias cuadradas ya que es necesario representar las conexiones entre cada par de nodos (i, j) del grafo.

Por otra parte, el sistema permite crear nuevos grafos. En este caso se le pedirá al usuario introducir el número de nodos (ver Figura 2.15).

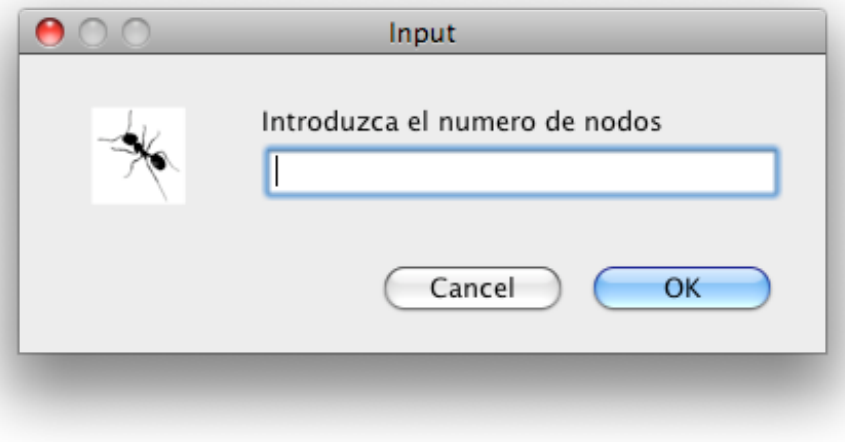

Figura 2.15. Crear un nuevo grafo

Indiferentemente de que el usuario cargue un grafo de memoria o cree un nuevo grafo, el sistema desplegará la pantalla de la Figura 2.16, en donde el usuario podrá configurar los parámetros de ejecución del algoritmo.

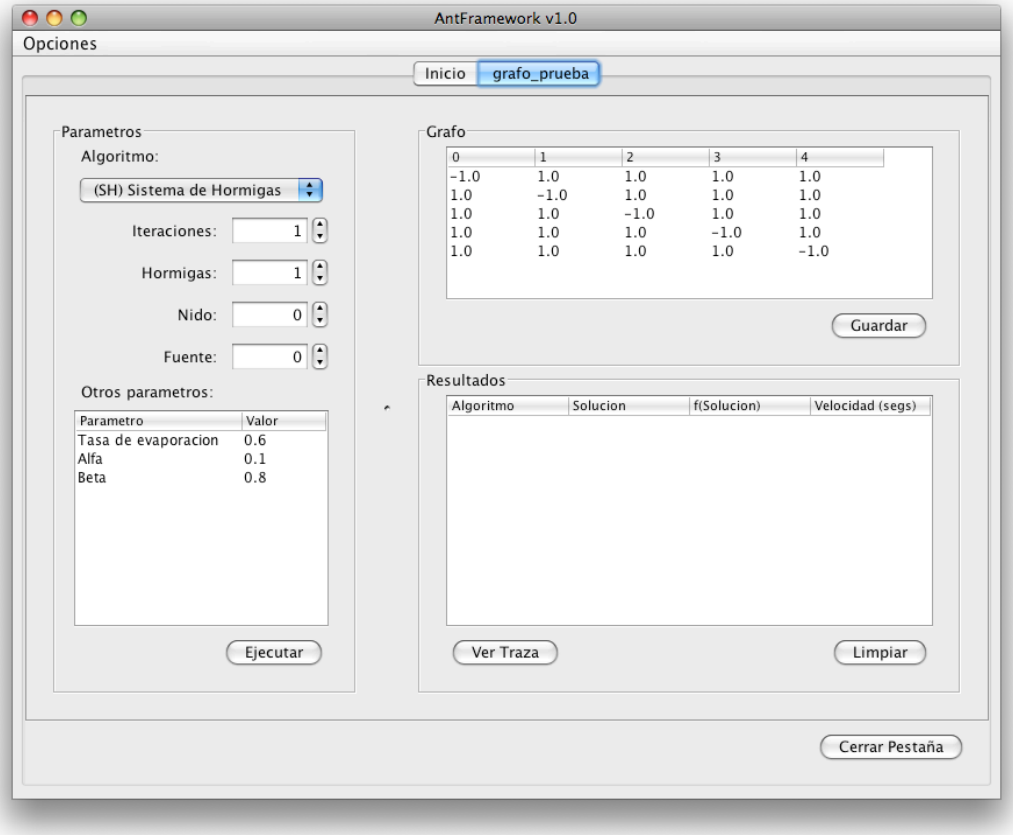

Figura 2.16. Ejecución del algoritmo

En esta pantalla el usuario puede configurar los parámetros de ejecución: nodo nido, nodo fuente, cantidad de hormigas y cantidad de iteraciones; además de los parámetros propios de cada algoritmo particular. También puede ver el grafo sobre el que se está trabajando y modificarlo *in situ*. Por último, podrá iniciar la ejecución de un algoritmo al presionar el botón ”Ejecutar” y observar los resultados en el panel situado en la parte inferior derecha.

Es importante destacar que la aplicación provee una representación gráfica del rastro de feromonas. Un ejemplo de dicha representación se puede observar en la Figura 2.17, en donde se observa una cuadrícula de *N x N* posiciones en donde *N* es el número de nodos del grafo. Cada posición se rellenará con un color de la escala de grises, en donde el negro representa una alta concentración de feromonas y blanco ninguna concentración. La aplicación permitirá ver el estado de la matriz de

feromonas en distintas iteraciones del algoritmo, lo cual a su vez permitirá hacer un análisis gráfico del comportamiento de un algoritmo.

Figura 2.17. Representación gráfica del rastro de feromonas

Por último, a continuación se presenta el código empleado por la aplicación y que hace uso del Framework para resolver el problema de la ruta más corta.

```
	public class SP_ACS extends antframework.algorithms.AntColonySystem {

public SP_ACS(String graphLocation, int problemInitialNode, int
problemDestinationNode) throws Exception{

	Graph G = new Graph(graphLocation);

	Enviroment Env = new Enviroment(G, true);

	this.setParam(AntColonySystem.initialNode  , problemInitialNode);

	this.setParam(AntColonySystem.destinationNode , problemDestinationNode);

	this.setParam(AntColonySystem.maxNumIterations , 5);

	this.setParam(AntColonySystem.beta , 0.25);

	this.setParam(AntColonySystem.maxCandidates , Integer.MAX_VALUE);

	this.setParam(AntColonySystem.r_0 , 0.5);

	this.setParam(AntColonySystem.ro_1 , 0.1);
```

```
        this.setParam(AntColonySystem.ro_2 , 0.9);
        this.setParam(AntColonySystem.tau_0 , 0.1);
        Env.setAnts(5);
        this.setE(Env);
}
public Vector<Integer> restriccions(int i, Vector<Integer> currentSolution){
        int cols = this.Graph.getM().getColumns();
        Vector<Integer> adjacents = new Vector<Integer>();
        for(int j=0; j < cols;j++){
             if(this.Graph.getM().position(i, j) < Integer.MAX_VALUE){
                          adjacents.add(j);
             }
        }
        return adjacents;
}
public void candidateList(int max){
         Matrix G = this.Graph.getM();
         int rows = G.getRows() , cols = G.getColumns() , cont;

         for(int i = 0; i<rows ; i++){
             cont = 0;
             Vector<Node> nodes = new Vector<Node>();
             for(int j = 0; j<cols; j++){
                 if((G.position(i, j) < Integer.MAX_VALUE)){
                     Node nodeTmp = new Node(j,G.position(i, j),1/G.position(i, j));
                     nodes.add(nodeTmp);
                 }
             }
              Collections.sort(nodes);
              if(nodes.size() > max){
             nodes.setSize(max);
        }
```

```
                AntColonySystem.candidates.put(i, nodes);
            }
        }
        public double heuristicInfo(double number){
            return 1/number;
        }
        public double f(Vector<Integer> Solution){
            int numberSolutionNodes = Solution.size();
            if(numberSolutionNodes != 0 && Solution.elementAt((numberSolutionNodes - 1))
           != this.Parameters.get(AntColonySystem.destinationNode).intValue()){
                return Double.MAX_VALUE;
            }else{
                return super.f(Solution);
            }
        }
    }
```

Del código anterior se destacan las siguientes funciones: constructor de la clase, restricciones, lista de candidatos, información heurística y f. La funcionalidad de cada una de éstas se explica a continuación.

- El constructor de la clase es responsable de inicializar el ambiente (hormigas y grafo) y configurar los parámetros de ejecución. En este caso se está cargando un grafo G en memoria y utilizando 5 hormigas.

- La función de restricciones se encarga de calcular, para cada nodo *i* del grafo, una lista con aquellos nodos que sean posible visitar partiendo del nodo *i*, según las restricciones del problema a resolver. Para el caso ejemplo, esta función simplemente devuelve todos los nodos conectados al nodo *i*.

- La función de lista de candidatos se encarga de generar, para cada nodo *i* del grafo, una lista de nodos candidatos para visitar a partir del nodo i. Esta lista estará ordenada según el potencial de los candidatos, con el candidato más prometedor de primero. En el caso ejemplo se ordenan los nodos candidatos

ascendentes según la distancia al nodo i. Así, el nodo más prometedor será el que se encuentre a menor distancia.

- La información heurística recibe un nodo y devuelve, según una heurística particular del problema a resolver, un número que representa el beneficio potencial de seleccionar dicho nodo. Esta función puede variar en distintas implementaciones de un mismo problema de optimización y afecta sensiblemente la calidad y velocidad de las soluciones obtenidas. En este caso simplemente 1/(longitud del camino).
- La función f evalúa una solución construida por una hormiga. En este caso recibe una solución o camino y calcula la longitud total del mismo, nodo por nodo.

2.7 Experimentos de Verificación

En esta sección se detallan los experimentos realizados para verificar el correcto funcionamiento y eficiencia de las distintas implementaciones de los algoritmos de Colonias de Hormigas dentro del Framework

- **Consideraciones Generales**

En general, los experimentos de validación tomarán en cuenta dos dimensiones: calidad y velocidad de la solución, contra algún otro algoritmo determinista y/o estocástico. Para medir la calidad de las soluciones se utilizó como métrica el error absoluto *(e_{abs})*, el cual se calculó como la diferencia absoluta (valor positivo) entre el valor de control (v_c) y el valor obtenido (v_o) por algún algoritmo de Colonias de Hormigas.

Por otra parte, la velocidad *(v)* de las soluciones se obtiene como una medida relativa del tiempo de ejecución de un algoritmo. Al iniciarse la ejecución de un algoritmo se toma una muestra del tiempo del sistema en milisegundos *(t_i)*. Este proceso se repite al final de la ejecución de un algoritmo *(t_f)*. Matemáticamente, la velocidad de un algoritmo se calcula según la siguiente ecuación:

$$v = t_f - t_i \tag{2.1}$$

Por último, para obtener el tiempo y error promedio para cada grafo y conjunto de parámetros, se ejecutó cada algoritmo de Colonias de Hormigas 10 veces. En el caso del mejor tiempo y mejor error, se seleccionó el menor resultado de los 10 posibles. El mismo procedimiento se utilizó para los algoritmos de control.

La calidad de una solución dependerá de la naturaleza del problema de optimización a resolver. En general una solución será de mayor calidad que otra si es menor numéricamente y el problema planteado es de minimización. Lo contrario también es cierto: una solución será de mayor calidad que otra si es mayor numéricamente y el problema planteado es de maximización. Por velocidad de solución se entiende que será mejor aquélla solución que se consiga en un menor

tiempo o, en otras palabras, un algoritmo será más veloz que otro si consigue soluciones en un menor tiempo de ejecución. Para este caso, la medida de tiempo utilizada serán milisegundos (segundos / 1.000).

- **Grafos de Prueba**

Se emplearon 3 estructuras de grafos distintas para la realización de las pruebas: grafos completos, grafos incompletos o poco densos y grafos para el recorrido de caballo. Un grafo completo es aquél en donde todas sus aristas conectan cada par de nodos, mientras que un grafo incompleto es aquél en donde no existe conexión en al menos un par de nodos distintos. Un grafo para el recorrido de caballo es una es una representación de un posible recorrido de caballo tal y como se explica en la sección del Problema de Caballo. Adicionalmente, cabe destacar que independientemente de la estructura de un grafo cualquiera, en aquéllas posiciones en donde se cumpla que i = j (la misma fila y columna), no existirá una conexión válida, de forma tal de evitar posibles bucles al momento de construir soluciones.

En general, la estructura de un grafo podrá ser fácilmente identificada por el nombre del mismo según el siguiente patrón:

<cg | g | kg><# de nodos>x<# de nodos>[parámetros del grafo].graph

En donde cg = grafo completo, g = grafo incompleto, kg = grafo de caballo. Si un grafo es completo, el nombre puede terminar en <_0 | _1>. Si termina en _0, dicho grafo poseerá únicamente conexiones con valor 1 en sus aristas. En caso contrario en donde termina en _1, el grafo tendrá conexiones con valores enteros aleatorios dentro del rango [1, # de nodos]. Un grafo incompleto sólo tendrá conexiones aleatorias similares al caso del grafo completo que termina en _1. Adicionalmente, estos grafos contendrán un número real indicando la densidad de los arcos, a menor número mayor densidad del grafo. Si es un grafo de recorrido de caballo, el número de nodos en el identificador representa las dimensiones del tablero y contiene una cantidad de nodos igual a su multiplicación, las conexiones válidas tendrán sólo valores iguales a 1.

Por ejemplo, el siguiente nombre de grafo cg20x20_1.graph, identifica a un grafo completo con 20 nodos y conexiones aleatorias en el rango [1, 20]. Otro ejemplo sería g10x10_0.3[2].graph, cuya matriz de adyacencia se muestra en la figura 2.18 y que identifica a un grafo incompleto de 10 nodos. Por último el grafo kg6x6.graph es un grafo de caballo para un tablero de 6 x 6 conteniendo 36 nodos. Más adelante se nombrarán explícitamente los grafos utilizados en cada caso, según sea el caso. Los grafos analizados en el presente documento se podrán encontrar en el anexo 3.

0	5	7	0	0	5	4	0	9	0
0	0	1	1	8	9	3	5	6	0
2	8	0	8	10	0	5	4	0	9
5	8	2	0	0	0	9	8	8	1
10	5	3	8	0	9	0	9	0	6
2	6	2	0	0	0	4	7	0	0
0	0	6	0	6	1	0	9	4	0
0	0	0	8	2	1	7	0	6	0
4	0	4	3	0	1	0	10	0	9
7	10	3	0	3	7	6	10	0	0

Figura 2.18. Matriz de Adyacencia del grafo g10x10_0.3[2]

Se plantearon 3 experimentos: el de conseguir la ruta más corta de un grafo, el del recorrido de Caballo y el del Problema del Agente Viajero. Cada uno de estos se explican a continuación:

2.7.1 Ruta más Corta

Este experimento fue concebido como una prueba de control, con el fin de comprobar el correcto funcionamiento de los algoritmos. Por correcto funcionamiento de los algoritmos de Colonias de Hormigas se entiende que éstos sean capaces de encontrar un camino válido entre cualesquiera par de nodos *(i,j),* suponiendo que éste exista. Un camino legítimo se construye al respetar la estructura del grafo y seguir, a partir de un nodo *i*, un nodo *j* adyacente con conexión válida.

2.7.1.a Definición

Dado un grafo cualquiera *G*, este experimento consiste en encontrar la ruta más corta entre cada par de nodos *(i, j),* para todos los nodos i != j.

2.7.1.b Objetivos

Se desea contrastar la velocidad y calidad de la solución contra el algoritmo de Dijkstra para encontrar rutas mínimas. Dicho algoritmo es el estándar *de facto* para conseguir las rutas más cortas dado un nodo origen al resto de los nodos de un grafo, y funciona como un algoritmo *greedy* o voraz altamente eficiente.

2.7.1.c Parámetros

Las siguiente tabla muestra el conjunto de parámetros utilizados por los diferentes algoritmos de Colonias de Hormigas para resolver el problema de la ruta más corta:

Algoritmo	Conjunto de Parámetros		
Sistema de Hormigas	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
alfa	0.5	0.85	0.85
beta	0.75	0.75	0.85
tasa evaporacion	0.75	0.7	0.65
Sistema de Hormigas Élite	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
alfa	0.5	0.5	0.5
beta	0.25	0.25	0.25
tasa evaporacion	0.75	0.7	0.65
epsilon	2	2	2
Sistema de Hormigas Rango	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
alfa	0.5	0.5	0.5
beta	0.75	0.5	0.5
tasa evaporacion	0.75	0.7	0.65
epsilon	5	7	10
Sistema de Colonias de Hormigas	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
beta	1	0.8	0.5

candidatos	50	∞	50
r_0	0.4	0.8	0.5
ro_1	0.1	0.25	0.1
ro_2	0.9	0.9	0.9
tau_0	0.1	0.1	0.1
Hormiga Q	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
alfa	0.5	0.5	0.5
beta	0.75	0.75	0.75
candidatos	40	40	40
r_0	0.35	0.35	0.35
ro_1	0.5	0.5	0.5
ro_2	0.5	0.5	0.5
tau_0	0.1	0.1	0.1
gamma	0.75	0.75	0.75
Sistema de Hormigas Máximo Mínimo	**1**	**2**	**3**
iteraciones	50	100	200
hormigas	5	10	15
tasa evaporacion	0.01	0.01	0.01
alfa	0.8	0.8	0.8
beta	1	1	1
tau_max	1.75	1.75	1.75
tau_min	0.75	0.75	0.75

Tabla 2.1. Parámetros para la resolución del problema de las rutas más cortas

Por cada algoritmo se realizaron cuatro conjuntos de prueba, con valores distintos para cada parámetro dentro de cada conjunto. Como se puede observar en la tabla anterior, se aumentaron progresivamente el número de iteraciones y hormigas utilizadas en cada conjunto de pruebas. Así, el conjunto 0 iteró 5 veces y se emplearon 5 hormigas, mientras que para el conjunto 3 se utilizaron 200 iteraciones y 15 hormigas. De esta forma se pudo observar el rendimiento de los algoritmos en función de un mayor número de iteraciones y de agentes artificiales. En general, este mecanismo se utilizará para todos los experimentos.

2.7.1.d Grafos utilizados

Las siguiente tabla muestra los grafos utilizados para éste experimento fueron los siguientes:

Grafos Completos	Grafos Incompletos
cg10x10_0.graph	g10x10_0.3[0].graph
cg10x10_1.graph	g10x10_0.3[1].graph
cg15x15_1.graph	g10x10_0.3[2].graph
cg18x18_0.graph	g10x10_0.7[0].graph
cg18x18_1.graph	g10x10_0.7[1].graph
cg20x20_0.graph	g10x10_0.7[2].graph
cg20x20_1.graph	g15x15_0.3[0].graph
cg25x25_0.graph	g15x15_0.3[1].graph
cg25x25_1.graph	g15x15_0.3[2].graph
cg5x5_0.graph	g15x15_0.7[0].graph
cg5x5_1.graph	g15x15_0.7[1].graph
	g15x15_0.7[2].graph
	g20x20_0.3[0].graph
	g20x20_0.3[1].graph
	g20x20_0.3[2].graph
	g20x20_0.7[0].graph
	g20x20_0.7[1].graph
	g20x20_0.7[2].graph
	g50x50_0.3[0].graph
	g50x50_0.3[1].graph
	g50x50_0.3[2].graph
	g50x50_0.7[0].graph
	g50x50_0.7[1].graph

Tabla 2.2. Grafos para la resolución del problema de las rutas más cortas

2.7.1.e Resultados

En general se observó que los algoritmos de Colonias de Hormigas son capaces de obtener una ruta entre para cada par de nodos en un grafo cualquiera. Sin embargo, al comparar los resultados obtenidos por dichos algoritmos contra el algoritmo de Dijkstra, se puede concluir que éste último obtendrá resultados de mayor calidad en menor tiempo. Esta situación se ilustra en la Figura 2.19.

En cuanto a la calidad de las soluciones, de la tabla anterior se desprenden dos conclusiones: los algoritmos son capaces de conseguir rutas y el error de las mismas disminuye al aumentar el número de iteraciones y hormigas.

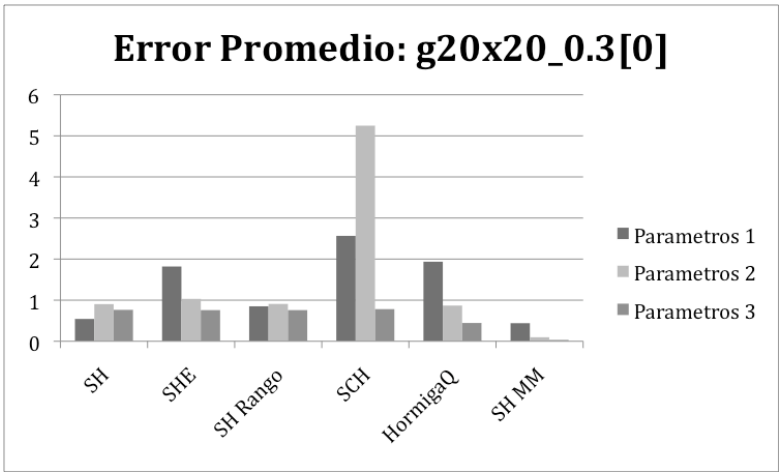

Figura 2.19. Error Promedio Rutas Mas Cortas, g20x20_0.3[0]

Algoritmo	Parámetros 1	Parámetros 2	Parámetros 3
SH	0.5436	0.9028	0.7642
SHE	1.8202	1.0306	0.7573
SH Rango	0.8505	0.9073	0.7576
SCH	2.5647	5.2478	0.7813
HormigaQ	1.9355	0.8686	0.4452
SHMM	0.4397	0.0971	0.0315

Tabla 2.7. Error Promedio Rutas Mas Cortas, g20x20_0.3[0]

En cuanto a la velocidad de las soluciones, las siguientes gráficas demuestran como al aumentar la cantidad de iteraciones y hormigas, aumenta en forma lineal el tiempo de ejecución del algoritmo.

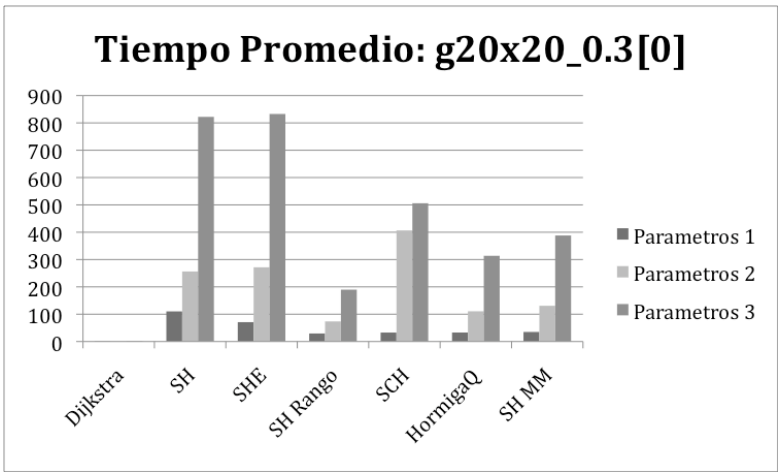

Figura 2.20. Tiempo Promedio Rutas Mas Cortas, g20x20_0.3[0]

Algoritmo	Parámetros 1	Parámetros 2	Parámetros 3
Dijkstra	0,0355	0,0347	0,04
SH	110,2747	256,0902	822,1355
SHE	70,8447	271,4952	832,8039
SH Rango	29,3131	73,7676	189,8355
SCH	32,8157	406,6060	506,0210
HormigaQ	32,9047	110,6436	313,7468
SHMM	32,0005	130,7763	388,1536

Tabla 2.8. Tiempo Promedio Rutas Mas Cortas, g20x20_0.3[0]

Del experimento de las rutas más cortas se puede concluir que los algoritmos de Colonias de Hormigas son capaces de conseguir soluciones, y dependiendo de los parámetros empleados (en especial la cantidad de hormigas e iteraciones), estas soluciones se acercaran a las soluciones óptimas. Sin embargo, dichas soluciones se encontrarán en un tiempo considerablemente mayor al empleado por el algoritmo de Dijkstra.

2.7.2 Recorrido de Caballo

2.7.2.a Definición

Dado un tablero cuadrado de *n x n* celdas que representan n^2 nodos, se desea visitar todos los nodos del grafo una sola vez, partiendo desde cualquier posición (i, j) y realizando sólo movimientos en *L,* similares a los que se le permiten realizar a la ficha del Caballo en el juego del Ajedrez y permitiendo en el último movimiento volver a la casilla de origen.

2.7.2.b Objetivos

Contrastar la velocidad de las soluciones obtenidas mediante los algoritmos de Colonias de Hormigas contra un algoritmo NP-completo con heurísticas altamente eficientes.

2.7.2.c Parámetros

Las siguiente tabla muestra el conjunto de parámetros utilizados por los diferentes algoritmos de Colonias de Hormigas para resolver el problema del recorrido de Caballo:

Algoritmo	Conjunto de Parámetros		
Sistema Hormigas	**1**	**2**	**3**
iteraciones	100	200	250
hormigas	15	20	25
alfa	0.25	0.15	0.1
beta	1.25	1.75	2
tasa evaporacion	0.2	0.15	0.1
Sistema de Hormigas Élite	**1**	**2**	**3**
iteraciones	100	200	250
hormigas	15	20	25
alfa	0.25	0.15	0.1
beta	1.25	1.75	2
tasa evaporacion	0.2	0.15	0.1
epsilon	1.5	2	2.5
Sistema de Hormiga Rango	**1**	**2**	**3**
iteraciones	100	200	250
hormigas	15	20	25
alfa	0.25	0.15	0.1
beta	1.25	1.75	2
tasa evaporacion	0.2	0.15	0.1
epsilon	5	7	10
Sistema de Colonias de Hormigas	**1**	**2**	**3**
iteraciones	100	200	250
hormigas	15	20	25
beta	1.25	1.75	2
candidatos	Ng / 2	Ng / 3	Ng / 3
r_0	0.3	0.25	0.2
ro_1	0.2	0.15	0.1
ro_2	0.2	0.15	0.1
tau_0	0.1	0.1	0.1
Hormiga Q	**1**	**2**	**3**
iteraciones	100	200	250
hormigas	15	20	25
alfa	0.25	0.15	0.1
beta	1.25	1.75	2
candidatos	Ng / 2	Ng / 3	Ng / 3
r_0	0.3	0.25	0.2
ro_1	0.2	0.15	0.1
ro_2	0.2	0.15	0.1
tau_0	0.1	0.1	0.1

gamma	0.75	1	1.25

Sistema de Hormigas Máximo Mínimo	1	2	3
iteraciones	100	200	250
hormigas	15	20	25
tasa evaporacion	0.2	0.15	0.1
alfa	0.25	0.15	0.1
beta	1.25	1.75	2
tau_max	1.75	1.75	1.75
tau_min	0.75	0.75	0.75

Tabla 2.3. Parámetros utilizados para la resolución del Recorrido de Caballo

En la tabla anterior, *Ng* significa números de nodos en el grafo. Así, si se experimenta con un grafo de 50 nodos, la lista de candidatos será de 25 miembros (*Ng* / 2 = 50 / 2 = 25).

2.7.2.d Grafos utilizados

Las siguiente tabla muestra los grafos utilizados para éste experimento.

Grafos de Caballo
kg10x10.graph
kg15x15.graph
kg20x20.graph
kg30x30.graph
kg3x3.graph
kg5x5.graph
kg6x6.graph
kg8x8.graph

Tabla 2.4. Grafos utilizados para la resolución del Recorrido de Caballo

2.7.2.e Resultados

En este experimento se comparó en la velocidad de las soluciones obtenidas por los algoritmos de Colonias de Hormigas contra un algoritmo NP-completo con heurísticas altamente eficientes. Es importante destacar que los algoritmos de Colonias de Hormigas empleados para la resolución de este problema son exactamente igual a los empleados para la resolución del problema del agente viajero, utilizando como nodo de partida y llegada el nodo ubicado en la mitad del tablero. Todos los grafos empleados para la resolución de este problema tendrán

conexiones únicamente en aquellos nodos que cumplan con los movimientos del caballo tal y como se explica en la Sección 2.6.2.

La figura a continuación presenta uno de los resultados obtenidos.

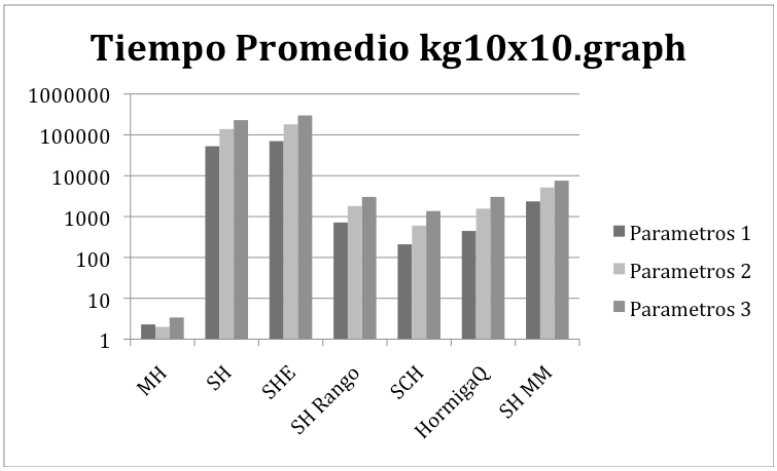

Figura 2.21. Tiempo Promedio Recorrido de Caballo, kg10x10, escala logarítmica

Algoritmo	Parámetros 1	Parámetros 2	Parámetros 3
MH	2,3	2	3,4
SH	52.859,9	138.697,4	229.666,6
SHE	70.467,7	181.152,8	299.172
SH Rango	717,4	1.807,2	3.025,1
SCH	210	602,4	1.363,8
HormigaQ	445,5	1.569,3	3.039,3
SHMM	2.358,3	5.145,1	7.586,1

Tabla 2.9. Tiempo Promedio Recorrido de Caballo kg10x10

Para el caso del grafo kg10x10, el algoritmo NP-completo consigue una solución entre 2 y 3 milisegundos, por lo que se puede establecer que éste es mucho más veloz que los algoritmos de Colonias de Hormigas en general. Por ejemplo, sabemos que para el conjunto de parámetros 3 se utilizan 250 iteraciones y 15 hormigas (ver sección 2.6.2.3), lo cual se refleja en los elevados tiempos de ejecución en relación con el algoritmo NP-completo (unas 1000 veces más rápido). Una situación similar a la detallada anteriormente ocurre para el grafo kg20x20. A continuación se presentan las gráficas para este grafo:

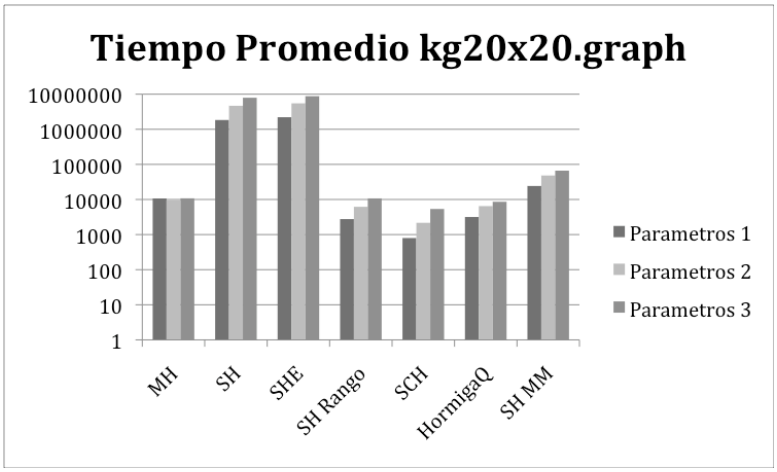

Figura 2.22. Tiempo Promedio Recorrido de Caballo, kg20x20, escala logarítmica

Algoritmo	Parámetros 1	Parámetros 2	Parámetros 3
MH	10.711	9.616	10.785
SH	1.838.802	4.678.252,25	7.929.058
SHE	2.211.871	5.480.483,75	8.800.862
SH Rango	2.767	6211	10.727
SCH	799	2.162,25	5.380
HormigaQ	3172	6.487,25	8.629
SHMM	24.290	48.196,5	66.924

Tabla 2.10. Tiempo Promedio Recorrido de Caballo kg20x20

De igual forma que ocurrió con el grafo kg10x10, las velocidades obtenidas para el grafo kg20x20 aumentaron proporcionalmente al conjunto de parámetros empleado, siendo el conjunto de parámetros 3 el que mayor cantidad de tiempo tardó en obtener una solución (hasta unas 10 veces más que para los otros parámetros). Sin embargo, para este grafo el algoritmo NP-completo consumió 10.785 milisegundos (10,785 segundos) antes de obtener resultados. Esta disminución de las velocidades de ejecución se atribuye al incremento del espacio de búsqueda de 10x10=100 nodos a 20x20=400 nodos.

Adicionalmente, se puede observar que para el caso del algoritmo NP-completo, un incremento de 300 nodos en el espacio de búsqueda resulta en un incremento en el tiempo necesario para obtener una solución de 10 segundos aproximadamente. Sin embargo, y pese a que se trata de un paradigma de resolución

de problemas diferente, esta misma relación se cumple para los algoritmos de Colonias de Hormigas. Por ejemplo, para el caso del grafo 10x10 y conjunto de parámetros 3, SHMM tardo unos 7 segundos aproximadamente en obtener solución, mientras que éste mismo algoritmo para el grafo 20x20 y conjunto de parámetros 3 tardó 70 segundos.

De éste experimento se puede concluir que a pesar de que los algoritmos de Colonias de Hormigas pueden emplearse para solucionar este problema, se debe idear una heurística apropiada para asistir el proceso de construcción de soluciones y descartar rápidamente aquéllos caminos poco prometedores, de forma tal de disminuir la cantidad de tiempo necesaria para obtener soluciones.

2.7.3 Problema del Agente Viajero

Este experimento es el más importante del presente trabajo, ya que el mismo es utilizado por una gran cantidad de trabajos de investigación en el área de optimización, lo que lo convierte en el estándar *de facto*.

2.7.3.a Definición

El problema del agente viajero se define como: “un vendedor tiene que visitar n + 1 ciudades, cada una exactamente una vez. La distancia entre cada par de ciudades viene dada por d_{ij} (en general $d_{ij} \neq d_{ji}$). El problema es encontrar el recorrido (tour) que comience y termine en la misma ciudad y minimice la distancia total recorrida”. Por conveniencia se suele etiquetar la ciudad origen como 0 y también como n + 1.

Formalmente, el problema del agente viajero puede ser representado por un grafo conexo y totalmente pesado $G = (N , A)$, siendo N el conjunto de nodos que representa las ciudades a ser visitadas por el agente y A el conjunto de arcos (i, j) que representan el costo de trasladarse de la ciudad i a la ciudad j. Adicionalmente, se sabe que en caso de que el grafo G no esté completo o falte algún arco entre una ciudad y otra, siempre se podrá construir un grafo $G´$ con valores tan altos en los arcos faltantes tal que sea prácticamente imposible que dichos arcos sean utilizados

en cualquier solución.

2.7.3.b Objetivos

Comparar el rendimiento (calidad y velocidad de solución) de los algoritmos de Colonias de Hormigas contra otros 2 algoritmos, específicamente: contra un algoritmo NP-completo sin heurísticas y contra un algoritmo estocástico. El primero será un algoritmo *backtracking* completo y el segundo un algoritmo genético.

2.7.3.c Parámetros

Las siguiente tabla muestra el conjunto de parámetros utilizados por los diferentes algoritmos de Colonias de Hormigas para resolver el problema del recorrido de Caballo:

Sistema de Hormigas	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000
alfa	0.5	0.25	0.15
beta	1	1.25	1.75
tasa evaporacion	0.25	0.2	0.15

Sistema de Hormigas Élite	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000
alfa	0.5	0.25	0.15
beta	1	1.25	1.75
tasa evaporacion	0.25	0.2	0.15
epsilon	1	1.5	2

Sistema de Hormigas Rango	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000
alfa	0.5	0.25	0.15
beta	1	1.25	1.75
tasa evaporacion	0.25	0.2	0.15
epsilon	2	5	7

Sistema de Colonias de Hormigas	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000

beta	1	1.25	1.75
candidatos	Ng / 2	Ng / 2	Ng / 3
r_0	0.35	0.3	0.25
ro_1	0.25	0.2	0.15
ro_2	0.25	0.2	0.15
tau_0	0.1	0.1	0.1
Hormiga Q	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000
alfa	0.5	0.25	0.15
beta	1	1.25	1.75
candidatos	Ng / 2	Ng / 2	Ng / 3
r_0	0.35	0.3	0.25
ro_1	0.25	0.2	0.15
ro_2	0.25	0.2	0.15
tau_0	0.1	0.1	0.1
gamma	0.5	0.75	1
Sistema de Hormigas Máximo Mínimo	**0**	**1**	**2**
iteraciones	10000	100	10000
hormigas	10	1000	1000
tasa evaporacion	0.25	0.2	0.15
alfa	0.5	0.25	0.15
beta	1	1.25	1.75
tau_max	1.75	1.75	1.75
tau_min	0.75	0.75	0.75

Tabla 2.5. Parámetros utilizados para la resolución del Problema del Agente Viajero

Para el Problema del Agente Viajero se emplearon una cantidad mucho mayor de iteraciones y hormigas. Esto con el fin de buscar mejores soluciones. Adicionalmente, para el Problema del Agente Viajero se emplearon los conjuntos de parámetros de la ruta más corta y del recorrido del caballo, Tablas 2.1 y 2.2 respectivamente.

2.7.3.d Grafos utilizados

Las siguiente tabla muestra los grafos utilizados para éste experimento.

Grafos Completos	Grafos Incompletos
cg10x10_0.graph	g10x10_0.3[0].graph
cg10x10_1.graph	g10x10_0.3[1].graph

cg15x15_1.graph	g10x10_0.3[2].graph
cg18x18_1.graph	g15x15_0.3[0].graph
cg20x20_1.graph	g15x15_0.3[1].graph
cg5x5_0.graph	g15x15_0.3[2].graph
cg5x5_1.graph	g15x15_0.7[0].graph
	g15x15_0.7[1].graph
	g20x20_0.3[0].graph
	g20x20_0.3[1].graph
	g20x20_0.3[2].graph
	g20x20_0.7[0].graph
	g20x20_0.7[1].graph
	g20x20_0.7[2].graph

Tabla 2.6. Grafos utilizados para la resolución del Problema del Agente Viajero

2.7.3.e Resultados

A continuación se presentan los resultados obtenidos al emplear algoritmos de Colonias de Hormigas para la resolución del problema del agente viajero:

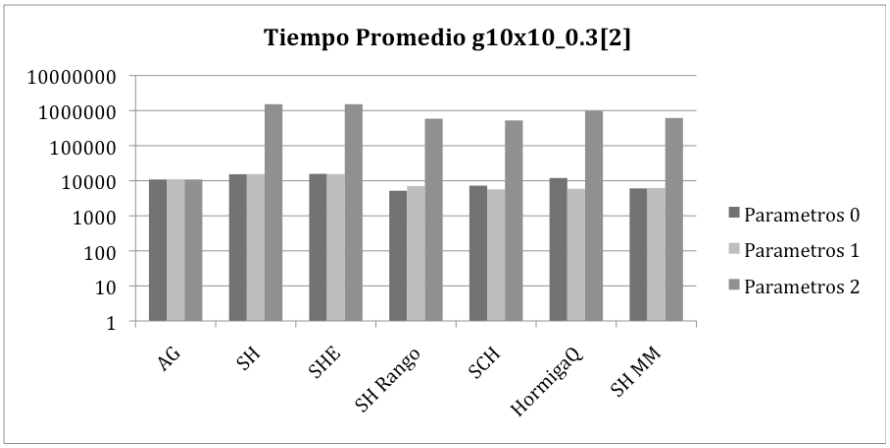

Figura 2.23. Tiempo Promedio Agente Viajero, g10x10_0.3[2], escala logarítmica

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
BT	0,1	0,1	0,1
AG	10.863,2	10.957,1	10.854,7
SH	15.289,8	15.442,2	1.523.157
SHE	15.716,6	15.439,7	1.523.496,33
SH Rango	5.184,8	7.052,2	587.621
SCH	7.237,9	5.668,1	524.504,33
HormigaQ	11.953,7	5.878,4	1.002.168,7
SHMM	6.045,4	6.219,6	616.180,7

Tabla 2.11. Tiempo Promedio, Agente Viajero, g10x10_0.3[2]

Como se mencionó en la sección 2.6.3.2, para este experimento se comparó el rendimiento de los algoritmos de Colonias de Hormigas contra un algoritmo genético y contra un *backtracking*. En el caso presentado en las figuras anteriores (grafo g10x10_0.3[2]), el *backtracking* tardó menos de 0.1 milisegundos en ejecutarse y encontrar la solución óptima. El error promedio obtenido para este mismo grafo se presenta en la siguiente figura.

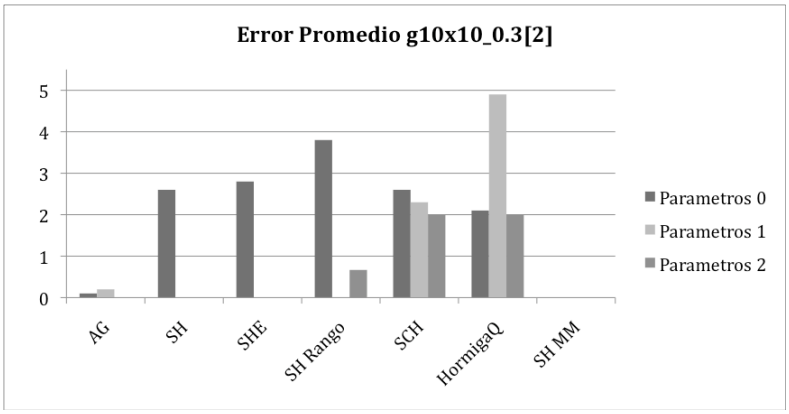

Figura 2.24. Error Promedio, Agente Viajero, g10x10_0.3[2]

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
AG	0,1	0,2	0
SH	2,6	0	0
SHE	2,8	0	0
SH Rango	3,8	0	0,6667
SCH	2,6	2,3	2
HormigaQ	2,1	4,9	2
SHMM	0	0	0

Tabla 2.12. Error Promedio, Agente Viajero, g10x10_0.3[2]

Se puede observar que para el grafo en cuestión los algoritmos consiguen una buena solución, considerando que el error es menor o igual a 5 y la longitud de un camino que sea válido como solución para este grafo es de 10 nodos. Por otra parte, un error menor, pero en un mayor tiempo de ejecución, se obtiene para el grafo g15x15_0.7[1].

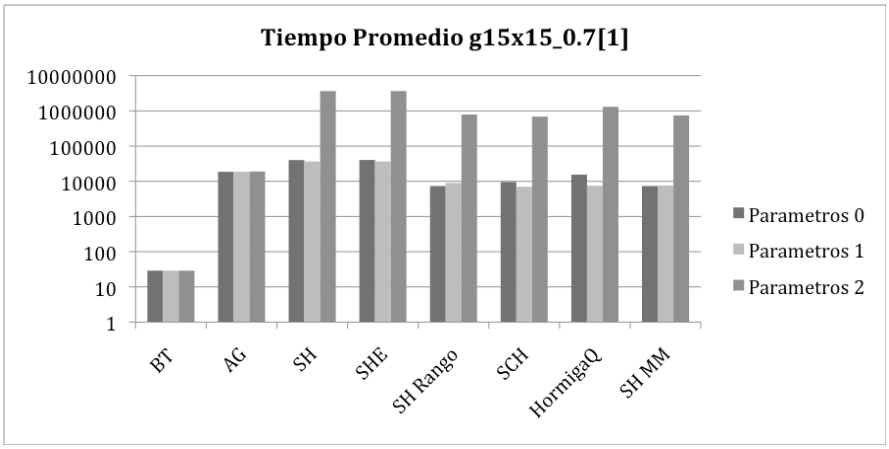

Figura 2.25. Tiempo Promedio, Agente Viajero, g15x15_0.7[1], escala logarítmica

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
BT	29	29	29
AG	18.580	18.687,8	18.928,6667
SH	40.148	36.584,6	3.643.350
SHE	40.432,3	36.640,7	3.654.242,3333
SH Rango	7.349,2	8.893,9	790.084,6667
SCH	9.581	7.012,7	689.612,6667
HormigaQ	15.473,7	7.489	1308.547,3333
SHMM	7.359,7	7.525,7	742.483,3333

Tabla 2.13. Tiempo Promedio, Agente Viajero, g15x15_0.7[1]

Para el grafo g15x15_0.7[1], el algoritmo de *backtracking* consigue la solución óptima en 29 milisegundos, mientras que todos los algoritmos de Colonias de Hormigas también consiguen esta solución pero en un tiempo superior. Por ejemplo, el mejor de los algoritmos para este caso es el SCH, parámetros 1, el cual consigue una solución óptima en 7.012,7 milisegundos. El algoritmo genético genera un error de 50, por lo que son más eficientes los algoritmos de Colonias de Hormigas.

Hasta el momento se han analizado las instancias de los algoritmos de Colonias de Hormigas cuando consiguen soluciones óptimas o muy buenas, pero en un tiempo significativamente mayor que el algoritmo de *backtracking*. A continuación se analizarán casos en los cuales estos algoritmos consiguen excelentes soluciones en un tiempo mucho menor que un *backtraking*.

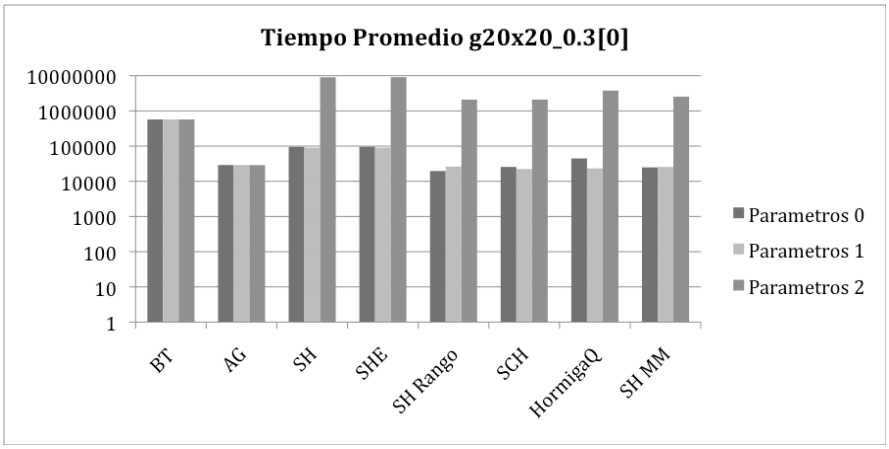

Figura 2.26. Tiempo Promedio, Agente Viajero, g20x20_0.3[0], escala logarítmica

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
BT	571.376	571.376	571.376
AG	28.959,8	29.093,9	28.846,6667
SH	95.402,9	90.631,2	9.051.668
SHE	96.102,3	90.775	9.126.133,6667
SH Rango	19.602,2	26.186,4	2.090.758,3333
SCH	25.589,1	22.402,8	2.092.556,3333
HormigaQ	44.646,8	23.261,6	3.769.517
SHMM	24.860,9	25.605,1	2.542.749,3333

Tabla 2.14. Tiempo Promedio, Agente Viajero, g20x20_0.3[0]

Para el grafo g20x20_0.3[0] el algoritmo de *backtracking* tarda 571.376 milisegundos (571 segundos aproximadamente, casi 10 minutos, 9 min. 31 seg.), en conseguir la solución óptima. Los tiempos de ejecución de los algoritmos de colonias de hormigas se muestra en las figura 2.26. Se puede observar que para los conjuntos de parámetros 1 y 2, los algoritmos tardan un tiempo considerablemente menor, mientras que para el conjunto de parámetros 3 el tiempo supera al del *backtracking*.

En cuanto al error de los algoritmos, la Figura 2.27 nos muestra como éste se acerca al óptimo a excepción de algunos casos particulares.

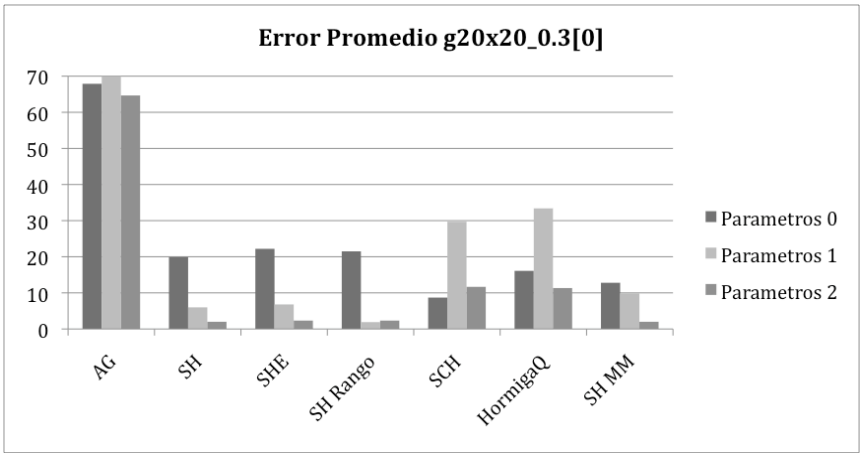

Figura 2.27. Error Promedio, Agente Viajero, g20x20_0.3[0]

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
AG	67,9	70,2	64,6667
SH	19,9	6	2
SHE	22,2	6,8	2,3333
SH Rango	21,5	1,9	2,3333
SCH	8,7	29,7	11,6667
HormigaQ	16,1	33,4	11,3333
SHMM	12,8	9,9	2

Tabla 2.15. Error Promedio, Agente Viajero, g20x20_0.3[0]

Para los algoritmos de Sistema de Hormigas (Élite, Rango y Máximo Mínimo) se cumple que el error disminuye proporcionalmente al índice del conjunto de parámetros utilizado. Los mejores resultados se obtienen con el conjunto de parámetros 2, en donde el error oscila está entre 0 y 5. Tomando en cuenta que una solución válida para este grafo se compone de 20 nodos, se puede concluir que estos algoritmos consiguen una solución cercana a la óptima en un tiempo mucho menor que el *backtracking*. Por ejemplo, para el algoritmo de Sistema de Hormigas Rango, parámetros 1, se obtiene un error igual a 1 en un tiempo de 26.186,4 milisegundos (26 segundos, menos de medio minuto), mientras que el *backtracking* consigue solución en casi 10 minutos. Esto supone un incremento de hasta 20 veces en la velocidad de ejecución para obtener una solución, y una penalidad de apenas 1 en el error.

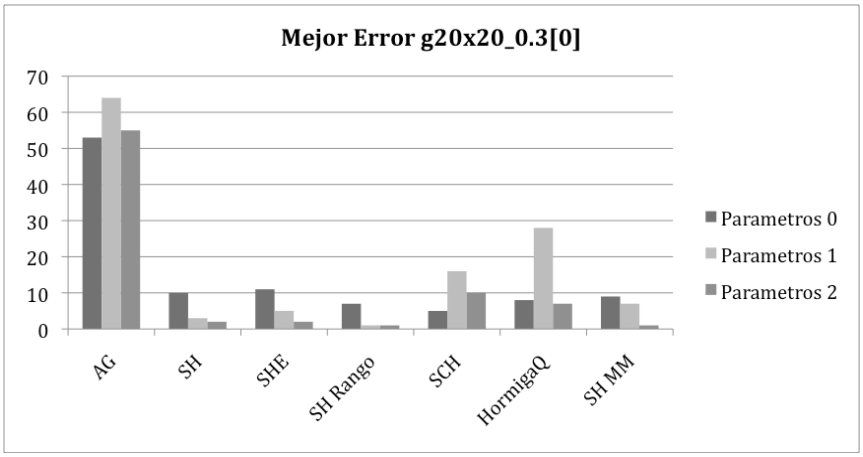

Figura 2.28. Mejor Error, Agente Viajero, g20x20_0.3[0]

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
AG	53	64	55
SH	10	3	2
SHE	11	5	2
SH Rango	7	1	1
SCH	5	16	10
HormigaQ	8	28	7
SHMM	9	7	1

Tabla 2.15. Mejor Error, Agente Viajero, g20x20_0.3[0]

Por otra parte, los algoritmos de Sistema de Colonias de Hormiga y HormigaQ, obtiene mejores resultados utilizando el conjunto de parámetros 0 (1000 iteraciones y 100 hormigas), con un error que oscila entre 5 y 8. El tiempo de ejecución de estos algoritmos va desde 20 segundos para SCH y 40 segundos para HormigaQ. Podemos concluir que para este problema estos algoritmos funcionarán mejor con un número mayor de iteraciones y menor de agentes artificiales. En general, cualquier algoritmo de Colonias de Hormigas conseguirá una solución relativamente cercana a la óptima en un tiempo mucho menor que el *backtracking*.

Un resultado similar al anteriormente descrito ocurre en el caso del grafo cg20x20_1. Recordemos que este es un grafo completo, de 20 nodos y pesos aleatorios.

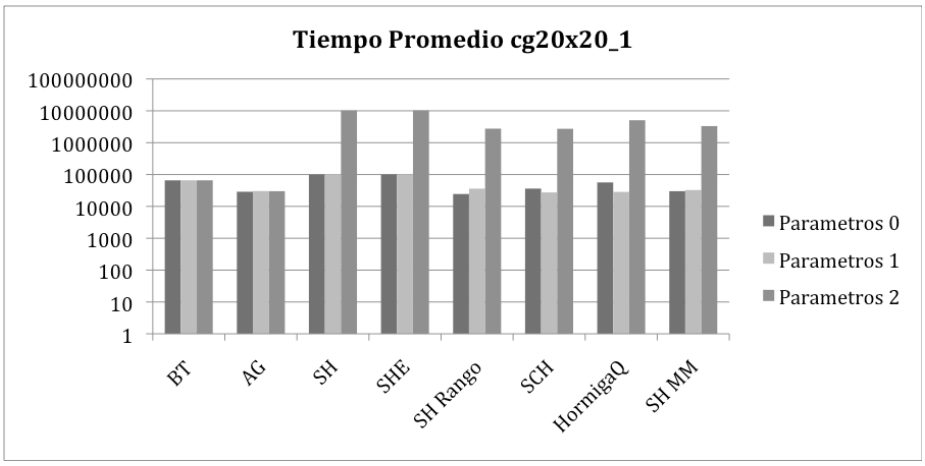

Figura 2.29. Tiempo Promedio, Agente Viajero, cg20x20_1, escala logarítmica

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
BT	65.875	65.875	65.875
AG	28.914,9	30.224,2	29.968,5
SH	99.573,6	102.295,5	1.02E+07
SHE	100.413,1	101.544,3	1.03E+07
SH Rango	24.485,8	35.971	2.764.370
SCH	36.282,8	27.534,2	2.732.590
HormigaQ	56.717,2	28.542,4	5.072.161,5
SHMM	30.004,2	32.596	3.317.834

Tabla 2.15. Mejor Error, Agente Viajero, g20x20_0.3[0]

El *backtracking* para el grafo cg20x20_1 tarda 65.875 milisegundos (65 segundos, un poco más de 1 minuto) en conseguir la solución óptima. Como se puede observar en la figura 2.29, los algoritmos de SH y SHE se ejecutan en aproximadamente 100 segundos, mientras que los demás algoritmos tardan un tiempo inferior al del *backtracking*. A continuación se muestra el error obtenido para este caso.

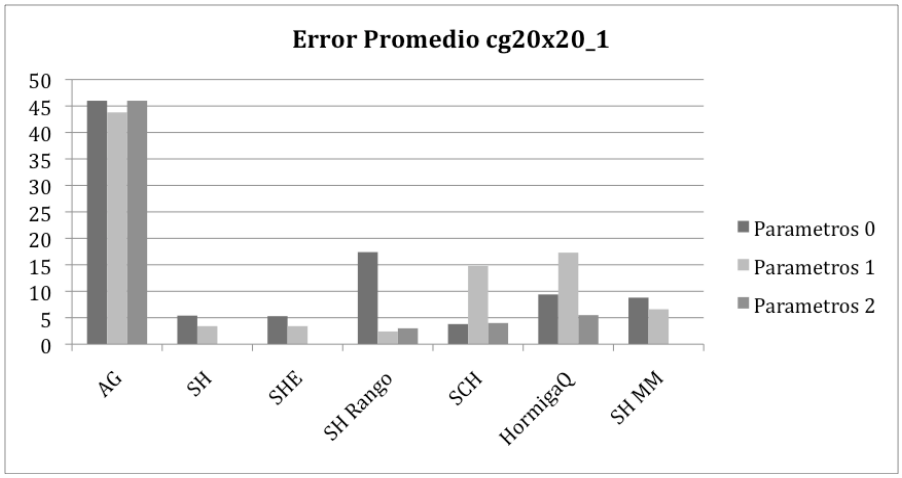

Figura 2.30. Error Promedio, Agente Viajero, cg20x20_1

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
AG	46	43,8	46
SH	5,4	3,4	0
SHE	5,3	3,4	0
SH Rango	17,4	2,4	3
SCH	3,8	14,8	4
HormigaQ	9,4	17,3	5,5
SHMM	8,8	6,6	0

Tabla 2.15. Mejor Error, Agente Viajero, g20x20_0.3[0]

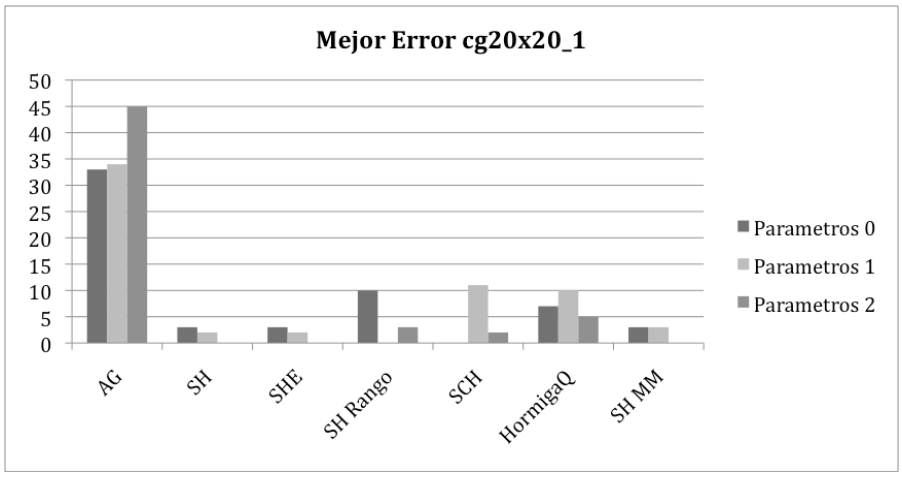

Figura 2.31. Mejor Error, Agente Viajero, cg20x20_1

Algoritmo	Parámetros 0	Parámetros 1	Parámetros 2
AG	33	34	45
SH	3	2	0
SHE	3	2	0
SH Rango	10	0	3
SCH	0	11	2
HormigaQ	7	10	5
SHMM	3	3	0

Tabla 2.15. Mejor Error, Agente Viajero, g20x20_0.3[0]

De la gráfica anterior se puede concluir que SH y SHE consiguen un buen error mas sin embargo, como explicamos antes, tardan más que el *backtracking* en conseguir la solución óptima, por lo que no serían la mejor elección para este tipo de grafos. Sin embargo, los demás algoritmos presentan resultados interesantes. Por ejemplo, SH Rango, para el conjunto de parámetro 1, consigue la solución óptima (error = 0) y tarda 33 segundos en ejecutarse, la mitad de lo que tarda el *backtracking* en conseguir el mismo resultado. Se puede deducir que el algoritmo consigue la solución óptima porque utiliza muchos agentes artificiales (1000 hormigas), los cuales, como se explicó en el capítulo anterior, emplean una estrategia elitista para actualizar el rastro de feromonas, lo que redunda en una preservación y refuerzo de los mejores posibles caminos posibles.

Otros resultados que se desprenden del análisis del grafo cg20x20_1 son los de los algoritmos SCH y SHMM. Para el SCH se consigue la solución óptima con el conjunto de parámetros 1 en un tiempo de 27 segundos, lo que representa una mejora sobre el SH Rango y el *backtracking*. Por otra parte, el SHMM arroja un error de 3 en un tiempo de ejecución de aproximadamente 30 segundos. Para este algoritmo, no hubo variación del error entre el conjunto de parámetros 0 y el 1. Por último, el algoritmo HormigaQ presenta un rendimiento inferior en comparación con otros algoritmos, independiente de los parámetros utilizados. Sin embargo, se pudiera argumentar que el error de éste algoritmo, oscilante entre 5 y 10, pudiera ser útil para algún tipo de aplicaciones, considerando que se obtiene entre 20 y 40 segundos.

Por último, cabe destacar que de la experimentación con grafos a partir de 25 nodos se puede concluir que utilizar un algoritmo de *backtracking* no es una opción viable. Se pudo comprobar que para un grafo de más de 20 nodos un *backtracking* tarda más de 4 días en ejecutarse, sin obtener la solución óptima final, mientras que por su parte, un algoritmo de Colonias tardará un tiempo mucho menor en conseguir una solución. No obstante, nada se puede decir del error de una solución obtenida de esta forma ya que no se cuenta con una solución óptima contra la que hacer comparaciones. Sin embargo, en la mayoría de las aplicaciones prácticas, dichas soluciones posiblemente serán mucho más atractivas que aquellas soluciones que tarden varios días en conseguirse.

2.7.4 Consideraciones Finales

De la experimentación con las diversas variantes de los algoritmos basados en Colonias de Hormigas se puede concluir lo siguiente:

Los algoritmos basados en Colonias de Hormigas se presentan como una alternativa viable frente a otros paradigmas en optimización. Para el caso de las rutas más cortas, estos algoritmos son capaces de construir caminos en un tiempo razonable. Para el problema del recorrido de caballo, estos algoritmos también pueden emplearse pero no obtendrán buenas soluciones sino se aplica una heurística específica al problema en cuestión. Estos dos experimentos se realizaron como una prueba de control y para probar la velocidad de los algoritmos respectivamente.

Por otra parte, para el caso del problema del agente viajero, los algoritmos de colonias de hormigas no sólo representan una alternativa viable sino que, bajo ciertas condiciones, superan el rendimiento de un *backtracking* y de un algoritmo genético. Se dice que superan el rendimiento a pesar de que no consigan una solución óptima porque éstos tardan sólo una fracción del tiempo en conseguir una solución muy cercana a la óptima. Lo que implica que para aplicaciones sensibles al tiempo de ejecución, los algoritmos de Colonias de Hormigas serán una excelente alternativa. Y estas soluciones subóptimas podrían usarse como punto de partida para un algoritmo

backtracking a modo de cota superior/inferior y reducir el espacio de búsqueda considerablemente.

También se encontró que para diferentes tipos de los algoritmos de colonias de hormigas, habrán parámetros que favorecerán la calidad de las soluciones obtenidas. Por ejemplo, para el algoritmo SH Rango se obtienen mejores soluciones cuando se emplea una cantidad superior de agentes artificiales y se disminuye la cantidad de iteraciones. Por el contrario, el algoritmo HormigaQ actúa mucho mejor con menos "hormigas" y más iteraciones. Otro ejemplo es el SHMM, el cual parece necesitar tanto una mayor cantidad de agentes y de iteraciones para obtener mejores resultados. El algoritmo de SCH, al igual que HormigaQ responde mejor a mayor número de iteraciones. De los algoritmos SH y SHE se puede concluir que en general fueron los que más tiempo consumieron, sin importar el tamaño del espacio de búsqueda o los parámetros empleados.

Por último, es importante mencionar que los resultados analizados en el presente documento fueron los más representativos a efectos de los objetivos planteados. Sin embargo, dichos resultados no fueron los únicos obtenidos durante el proceso de investigación. En el anexo se podrán encontrar los resultados de los algoritmos para todos los grafos nombrados en la sección 2.6, organizados por experimento, error y tiempo de ejecución.

CONCLUSIONES Y RECOMENDACIONES

Para finalizar este trabajo se presentan algunas conclusiones y conjeturas que se desprendieron del proceso de investigación del mismo.

Como objetivo principal del presente trabajo de investigación, se planteó la creación de una herramienta en software libre que integre los algoritmos de optimización por Colonias de Hormigas en una sola plataforma. Adicionalmente, se verificó el comportamiento de los algoritmos en distintos problemas de optimización. Para alcanzar estos objetivos se analizaron en profundidad los modelos computacionales derivados del comportamiento de Colonias de Hormigas naturales en búsqueda de alimento.

Se estableció que es posible realizar una abstracción de las variantes de los algoritmos basados en Colonias de Hormigas, con el fin de organizar una jerarquía de clases y objetos según el paradigma de programación orientada a objeto. Luego, dichas clases y objetos se conjugaron para formar un Framework en el cual se abstraen las funcionalidades básicas de cualquier algoritmo, y se le delega al programador la responsabilidad de implementar ciertas funciones para adaptar el funcionamiento del Framework a un problema de optimización específico.

En cuanto al desempeño de los algoritmos de Colonias de Hormigas, se comprobó que, bajo ciertas condiciones, éstos son una opción viable en contraste con otras opciones como algoritmos NP-completos y algoritmos genéticos. En comparación con los algoritmos NP-completos sin heurísticas, como en el caso del *backtracking* empleado para solucionar el problema del agente viajero, los algoritmos de Colonias de Hormigas en general consiguen una solución muy cercana a la óptima en un tiempo mucho menor. Para los algoritmos NP-completos con heurísticas, como el caso del recorrido de caballo, se observó que éste supera en rendimiento a los algoritmos de Colonias de Hormigas. Como conclusión se obtiene que la calidad de la solución (en tiempo y precisión) esta directamente relacionada a la calidad de la

heurística utilizada, lo cual es siempre un factor a considerar cuando se resuelve un problema de optimización.

En el caso específico del problema del agente viajero, se obtuvieron soluciones muy cercanas a la óptima en un tiempo menor y se pudo comprobar que un aumento en los parámetros de ejecución, específicamente en el número de iteraciones y hormigas empleadas, redundan en una disminución del error en las soluciones obtenidas. Se puede concluir entonces que el error será inversamente proporcional al tiempo que tarde en ejecutarse un algoritmo, lo que no implica que un tiempo de ejecución muy prolongado garantice la obtención de una solución óptima.

Por lo anterior se debe tener en cuenta los objetivos del problema a resolver para determinar si un algoritmo de Colonias de Hormigas es una buena opción. Si en una aplicación no es necesario obtener una solución óptima y se desea obtener una solución buena en un lapso de tiempo razonable, entonces un algoritmo de Colonias de Hormigas es una opción viable. En caso contrario se deben explorar otras alternativas como lo pueden ser algoritmos NP-completos con heurísticas apropiadas. Adicionalmente, si se emplea un algoritmo de Colonias de Hormigas con heurísticas apropiadas y un tiempo de ejecución razonable en función del espacio de búsqueda, se puede garantizar que en general se conseguirán buenas soluciones las cuales pueden servir como entrada de un algoritmo *backtracking,* el cual luego puede calcular la solución óptima en un tiempo mucho menor.

Para finalizar se proponen algunas recomendaciones para trabajos futuros sobre este tema:

- Ampliar el Framework elaborado en el presente trabajo para incluir nuevos algoritmos y variantes de Colonias de Hormigas actualmente en investigación.
- Extender el funcionamiento de los algoritmos en el Framework para que funcionen en aplicaciones distribuidas y en paralelo.

- Verificar la aplicabilidad de los algoritmos para otro tipo de tareas en el área de minería de datos como agrupamiento, minería de textos o descubrimiento de conocimiento.

- Constatar el rendimiento de los algoritmos contra otros paradigmas de la Inteligencia Artificial, como pueden ser: Redes Neuronales, Aprendizaje Q, etc. Adicionalmente, se puede comparar contra algoritmos de Inteligencia de Enjambre, rama a la cual pertenecen los algoritmos de Colonias de Hormigas.

- Explorar el rendimiento de los algoritmos de Colonias de Hormigas en la resolución de otros problemas de optimización distintos a los presentados en éste trabajo.

- Crear un sistema dirigido a usuarios finales que permita, a través de una interfaz usable, hacer uso de los algoritmos del Framework sin necesidad de saber de programación.

Referencias

- Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). *A new rank-based versiono f the Ant System: A computacional study*. Central European Journal for Operations Research and Economics.
- Deneubour, J.-L., Aron, S., Goss, S., & Pasteels, J.-M. (1990). *The self-organizing exploratory pattern of the Argentine ant. Journal of Insect Behavior*, 3, páginas 159-168.
- Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The Ant System: An Autocatalytic optimizing process*. Technical report 91-016 revised, Dipartimento di Elettronica, Politecnico di Milano, Milan.
- Dorigo, M. (1992) *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan.
- Dorigo, M., Maniezzo, V., / Colorni, A. (1996). *Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics.
- Dorigo, M. (2004). Ant Colony Optimization. (Primera Edición). Boston, EEUU: Massachusetts Institute of Technology.
- Engelbrecht, A. (2007). Computational Intelligence. (Segunda Edición). Pretoria, Sur África: University of Pretoria.
- Gambardella, L. M. & Dorigo, M. (1995) *Ant-Q: A reinforcement learning approach to the traveling salesman problem*. Consultado de Proceedings of the Twelfth International Conference on Machine Learning. Palo Alto, CA, Morgan Kaufmann.
- Gambardella, L. M. & Dorigo, M. (1996). *Solving symmetric and asymmetric TSPs by ant colonies*. Consultado de *Proceedings of the IEEE Congress on Evolutionary Computation,* páginas 622-627. Piscatawa, NJ, IEEE Press.
- Gambardella L.M., Rizzoli A.E., Oliverio F., Casagrande N., Donati A.V., Montemanni R. & Lucibello E. (2003) *Ant Colony Optimization for vehicle routing in advanced logistic systems*. Consultado de *Proceedings of MSH 2003*

– *International Workshop on Modelling and Applied Simulation*, páginas 3-9, 2-4, Bergeggi, Italia.

- Small, P. (2003). Consultado el día 23 de Febrero de 2009 de la World Wide Web:http://www.stigmergicsystems.com/stig_v1/stigrefs/article1.html
- Stützle, T. & Hoos, H. (1997). *MAX-MIN Ant System and Local Search for the Traveling Salesman Problem.* Consultado de *Proceedings of the IEEE Congress on Evolutionary Computation,* páginas 309-314. Piscatawa, NJ, IEEE Press.
- (2009, 14 de Marzo). Consultado el día 14 de Marzo de 2009 de la World Wide Web: http://www.scholarpedia.org/article/Ant_colony_optimization
- (2009, 6 de Abril). Consultado el día 6 de Abril de 2009 de la World Wide Web: http://en.wikipedia.org/wiki/Marco_Dorigo

ANEXOS

Anexo 1. Pseudo Código del algoritmo SHMM

Inicializar todos los parámetros: $\alpha, \beta, \rho, \hat{p}, \tau_{\min}, \tau_{\max}, f_\lambda, n_k$;

t = 0, $\tau_{max}(0) = \tau_{max}$, $\tau_{min}(0) = \tau_{min}$;

Ubicar a todas las hormigas artificiales $k = 1,\ldots, n_k$;

Para todos los arcos (i, j) **hacer**

$\tau_{ij}(t) = \tau_{max}(0)$

Fin para

$x^+(t) = \varnothing$, $f(x^+(t)) = 0$;

Repetir

Si punto de estancamiento **entonces**

Para cada arco (i, j) **hacer**

Calcular $\Delta\tau_{ij}(t)$ utilizando la ecuación:

Ecuación (1.16)

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}(t);$$

Fin Para

Fin si

Para cada hormiga k = 1,...,n_k **hacer**

$X^k(t) = \varnothing$;

Repetir

Seleccionar el siguiente nodo j según

la ecuación (1.1).

$x^k(t) = x^k(t) \cup \{(i,j)\}$;

hasta construir un camino o tour completo

Calcular $f(x^k(t))$;

Fin Para

Si (t mod f_λ)= 0 **entonces**

Mejor Iteración = falso;

sino

Mejor Iteración = verdad;

Fin si

Si Mejor Iteración **entonces**

Encontrar el mejor de la iteración actual:

$x^+(t) = x^k(t)$: $f(x^k(t)) = \min_{k'=1,\ldots,nk}\{f(k^{k'}(t))\}$;

Calcular $f(x^+(t))$;

Sino

Encontrar el mejor global:

$x(t) = x^k(t)$: $f(x^k(t)) = \min_{k'=1,\ldots,nk}\{f(k^{k'}(t))\}$;

Calcular $f(x(t))$;

Si $f(x) < f(x^+(t))$ **entonces**

$x^+(t) = x$;

$f(x^+(t)) = f(x)$;

fin si

Fin si

Para cada arco $(i, j) \in x^+(t)$ **hacer**

Aplicar regla de actualización global:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t);$$

Fin para

Para todos los arcos (i, j) **hacer**

Normalizar $\tau_{ij}(t)$ para que se encuentre en $[\tau_{min}, \tau_{max}]$

Fin para

$x^+(t+1) = x^+(t)$;

$f(x^+(t+1)) = f(x^+(t))$;

$t = t + 1$;

Actualizar $\tau_{max}(t)$ según la siguiente ecuación:

$$\tau_{\max}(t) = (\frac{1}{1-\rho})\frac{1}{f(\hat{x}(t))}$$

Actualizar $\tau_{min}(t)$ según la siguiente ecuación:

$$\tau_{\min}(t) = \frac{\tau_{\max}(t)(1-\sqrt{\hat{p}n_G})}{(n_G/2-1)\sqrt{\hat{p}n_G}}$$

hasta alcanzar la condición de parada;

Retorna x+(t) como la solución;

Anexo 2. Pseudo Código del algoritmo SCH

t = 0;

Inicializar todos los parámetros: β, p_1, p_2, r_0, τ_0, n_k;

Ubicar a todas las hormigas artificiales $k = 1,\ldots, n_k$;

Para cada arco (i, j) del grafo *G* **hacer**

$\tau_{ij}(t)$ = U(0 , τ_0);

Fin Para

$\hat{x}(t) = \varnothing;$

$f(\hat{x}(t)) = 0;$

Repetir

Para cada hormiga $k = 1,\ldots,n_k$ **hacer**

x^k (t) = Ø; #siendo x^k el vector de caminos construidos para #la hormiga k

Repetir

Si $\exists j \in$ lista de candidatos **entonces**

Seleccionar $j \in N_i^k(t)$ de la lista de candidatos según las ecuaciones:

Ecuación (2.1)

Ecuación (2.2)

Sino

Seleccionar a un nodo *J* fuera de la lista de candidatos:

$$J \in N_i^k(t);$$

Fin si

x^k(t) = x^k(t) U {(i,j)};

Aplicar actualización local de feromonas según:

$$\tau_{ij}(t) = (1-\rho_2)\tau_{ij}(t) + \rho_2\tau_0;$$

hasta se haya construido un camino completo;

Calcular $f(x^k(t))$;

Fin para

$x = x^k(t) : f(x^k(t)) = \min_{k'=1,\ldots,nk} \{f(x^{k'}(t))\}$;

Calcular *f(x);*

Si $f(x) < f(\hat{x}(t))$ **entonces**

$\hat{x}(t) = x;$

$f(\hat{x}(t)) = f(x);$

Fin si

Para cada arco $(i,j) \in \hat{x}(t)$ **hacer**

Aplicar la regla de actualización global de feromonas:

$$\tau_{ij}(t+1) = (1-\rho_1)\tau_{ij}(t) + \rho_1 \Delta\tau_{ij}(t);$$

Fin para

Para cada arco *(i, j)* **hacer**

Aplicar la regla de actualización global:

$$\tau_{ij}(t+1) = \tau_{ij}(t);$$

Fin para

$\hat{x}(t+1) = \hat{x}(t);$

$f(\hat{x}(t+1)) = f(\hat{x}(t));$

$t = t + 1;$

hasta alcanzar la condición de parada;

Retorna $\hat{x}(t)$ como la solución;

Anexo 3. Grafos Utilizados

Grafo cg5x5_0

0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Grafo cg5x5_1

0	5	5	3	4
4	0	2	3	1
3	1	0	1	3
1	3	2	0	4
3	4	1	2	0

Grafo cg10x10_0

0	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0

Grafo cg10x10_1

0	3	5	8	6	1	2	5	5	9
2	0	2	9	5	10	7	7	7	5
3	8	0	4	2	10	2	6	7	1
7	3	2	0	5	6	10	10	1	7
7	10	2	7	0	2	7	6	10	8
7	4	8	6	9	0	9	2	6	8
2	8	6	8	3	8	0	7	7	3
3	9	3	8	3	10	3	0	9	5
10	3	5	9	9	1	6	5	0	4
7	4	2	6	4	5	2	4	5	0

Grafo cg15x15_1

0	7	4	6	14	7	15	10	4	7	5	12	8	6	3
5	0	1	12	9	8	3	7	15	11	8	11	15	11	13
5	9	0	14	9	14	4	14	6	15	5	4	12	12	13
5	6	2	0	12	15	12	6	5	6	14	6	7	15	1
5	9	12	5	0	6	8	10	6	12	7	11	5	7	15
10	15	13	7	12	0	3	6	12	12	3	2	13	8	5
8	12	11	14	11	2	0	10	13	8	5	6	12	8	7
11	6	1	3	4	14	12	0	7	6	10	1	5	3	7
7	6	9	6	4	11	14	12	0	13	7	7	4	14	8
14	9	8	13	12	13	7	9	10	0	1	13	7	7	7
8	4	8	15	12	3	13	14	15	5	0	15	14	4	3
6	4	4	3	7	4	8	3	13	7	7	0	4	7	5
7	15	3	3	14	2	15	11	15	12	10	10	0	14	6
4	3	7	8	4	10	14	13	15	14	3	15	6	0	14
11	14	10	3	5	11	1	15	8	1	1	2	5	7	0

Grafo cg18x18_0

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1

1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Grafo cg18x18_1

0	2	8	16	14	13	11	5	12	9	17	12	13	8	17	17	11	5
5	0	15	16	12	13	4	15	10	11	7	13	10	4	4	18	1	1
16	11	0	8	10	18	13	15	18	17	10	5	1	10	2	15	14	14
2	2	10	0	3	10	16	1	14	5	15	18	15	17	6	18	15	12
8	10	15	1	0	6	4	7	9	13	16	9	11	1	4	3	17	2
6	10	7	17	17	0	5	13	3	3	6	9	10	3	11	9	15	12
8	7	13	6	5	2	0	10	8	13	3	16	11	13	3	6	5	15
9	10	10	5	1	18	13	0	4	11	8	15	16	2	9	14	8	9
1	9	7	1	18	13	16	1	0	16	15	9	12	12	1	1	9	7
2	1	13	14	6	11	16	5	5	0	7	11	13	9	1	2	10	14
4	15	4	13	3	6	3	13	6	5	0	17	5	2	18	10	4	18
18	2	17	12	16	14	9	3	17	1	7	0	13	17	10	8	10	9
8	9	11	1	10	9	4	2	16	17	16	11	0	12	3	5	17	9
7	3	14	5	18	12	10	4	16	8	13	14	17	0	18	11	16	10
12	14	4	5	8	11	11	9	15	10	5	17	7	8	0	3	11	3
3	5	9	4	6	17	4	16	17	7	18	12	14	2	18	0	11	8
7	8	9	4	2	4	5	8	12	1	17	2	6	16	16	7	0	2
17	4	5	18	1	11	6	4	11	3	12	16	15	8	12	2	5	0

Grafo cg20x20_0

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Grafo cg20x20_1

0	4	6	2	15	4	1	12	12	15	15	16	6	20	17	18	8	9	12	14
4	0	8	2	6	12	5	20	3	20	10	9	6	2	2	11	3	12	11	4
2	1	0	11	20	1	11	3	6	14	4	11	17	1	6	18	18	16	17	18
3	7	1	0	3	14	6	14	9	8	18	17	9	9	11	4	9	2	9	9
20	19	19	19	0	8	19	9	19	16	1	14	11	6	1	16	1	18	20	8
7	9	12	7	6	0	20	8	12	5	5	1	17	8	12	7	16	6	1	7
13	5	6	1	17	5	0	7	5	3	4	13	6	9	1	19	12	7	7	5
1	15	9	20	4	8	12	0	12	17	8	1	16	12	10	17	15	16	2	15
4	20	20	14	9	2	11	4	0	2	16	2	11	4	13	9	3	13	4	15
13	9	19	13	2	11	19	7	3	0	20	12	11	14	14	6	6	3	14	3
15	17	10	16	15	4	5	2	19	3	0	12	12	1	14	14	5	18	12	6
20	9	11	1	13	13	13	19	6	1	18	0	11	14	13	12	15	11	2	12
15	18	11	10	1	5	11	13	2	9	3	8	0	19	10	7	13	18	9	5
13	9	9	2	5	10	1	18	12	2	13	1	9	0	3	12	11	15	13	15
8	16	4	1	3	8	20	14	6	13	15	13	13	9	0	4	18	17	2	1
10	16	1	18	12	13	9	13	12	17	11	15	17	13	9	0	13	3	8	12
4	12	10	3	3	14	11	5	2	11	5	11	1	20	1	19	0	3	14	16
9	9	18	15	9	19	17	14	10	17	13	5	7	12	4	10	7	0	15	7
17	10	9	6	19	18	5	14	19	3	6	4	18	1	6	6	7	6	0	9
4	14	4	11	18	2	13	10	5	2	4	19	4	17	3	8	19	18	16	0

Grafo g10x10_0.3[0]

0	7	0	0	1	4	3	10	6	3
8	0	4	0	10	0	0	0	2	0
7	5	0	0	0	0	0	5	10	1
7	3	8	0	7	7	1	0	10	10
8	8	8	4	0	0	7	5	0	0
0	7	4	0	0	0	8	6	4	4
0	0	5	10	0	5	0	1	9	8
10	2	6	7	7	0	10	0	7	7
0	5	0	0	1	8	0	6	0	0
3	4	0	9	7	0	1	7	0	0

Grafo g10x10_0.3[1]

0	4	6	0	5	0	9	5	0	0
5	0	4	4	0	5	0	0	0	9
9	0	0	5	1	0	3	0	8	0
0	5	0	0	1	0	6	1	6	0
0	0	0	0	0	5	10	6	10	10
0	6	0	5	9	0	0	5	2	0
1	5	0	1	7	10	0	9	2	0
1	8	1	0	2	1	4	0	2	7
0	10	10	3	3	0	10	5	0	10
5	8	8	6	0	5	0	8	6	0

Grafo g10x10_0.3[2]

0	5	7	0	0	5	4	0	9	0
0	0	1	1	8	9	3	5	6	0
2	8	0	8	10	0	5	4	0	9
5	8	2	0	0	0	9	8	8	1
10	5	3	8	0	9	0	9	0	6
2	6	2	0	0	0	4	7	0	0
0	0	6	0	6	1	0	9	4	0
0	0	0	8	2	1	7	0	6	0
4	0	4	3	0	1	0	10	0	9
7	10	3	0	3	7	6	10	0	0

Grafo g10x10_0.7[0]

0	0	0	0	0	10	0	10	0	0
0	0	0	0	0	0	0	0	5	7
0	5	0	0	0	0	8	0	4	7
0	0	4	0	0	7	0	6	0	8
0	5	0	0	0	0	0	6	0	4
0	4	0	0	0	0	0	4	9	5
0	0	7	0	7	8	0	4	10	4
0	10	0	0	0	9	0	0	0	0
0	3	0	0	9	0	9	3	0	0
0	4	0	0	0	0	0	8	1	0

Grafo g10x10_0.7[1]

0	0	0	0	0	0	0	8	5	0
0	0	8	8	6	0	8	6	0	0
0	8	0	0	0	0	0	0	0	0
0	0	0	0	3	9	0	2	0	0
9	0	0	9	0	0	0	0	0	0
0	0	8	9	0	0	0	0	7	6
0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	2	0
0	0	3	0	10	0	0	1	0	0
0	0	4	1	0	0	1	0	0	0

Grafo g10x10_0.7[2]

0	0	0	0	7	0	0	0	0	0
0	0	0	0	0	0	0	3	9	0
0	7	0	0	0	0	3	0	0	0
0	0	1	0	0	0	0	0	9	5
0	0	8	0	0	0	10	0	0	0
3	0	0	6	0	0	0	0	0	5
0	0	6	0	0	0	0	7	0	7
6	3	0	0	0	8	9	0	9	6
0	8	0	0	0	0	3	0	0	4
0	8	0	0	0	0	0	0	8	0

Grafo g15x15_0.3[0]

0	14	15	8	0	11	14	12	12	11	4	0	8	7	14
0	0	2	0	0	8	0	5	3	3	10	0	0	0	0
13	0	0	9	0	9	7	10	6	15	1	0	2	6	15
0	10	0	0	0	11	0	0	15	0	11	0	12	1	11
0	0	0	0	0	8	1	6	5	2	3	5	9	1	10
2	0	0	2	0	0	0	15	6	8	0	0	3	2	0
12	15	4	0	0	11	0	2	13	0	4	3	2	3	6
0	12	0	13	0	12	12	0	14	6	9	5	5	5	0
8	10	15	0	15	13	12	0	0	3	0	5	12	3	2
5	12	0	4	3	7	7	0	10	0	5	0	0	5	0
0	11	8	3	4	0	3	0	0	6	0	3	0	0	4
11	14	0	0	11	1	12	15	9	0	0	0	11	11	14
13	13	3	0	0	10	0	3	7	10	5	13	0	8	13
0	8	0	3	0	14	13	0	5	12	0	10	12	0	10
9	0	13	6	12	0	13	6	12	2	0	6	0	11	0

Grafo g15x15_0.3[1]

0	0	0	11	3	6	8	0	5	3	11	8	1	4	11
9	0	15	9	13	10	6	0	2	9	10	0	10	14	6
11	14	0	0	0	14	8	0	0	10	6	0	0	0	15
8	11	0	0	2	12	0	1	0	0	0	0	8	4	15
0	0	0	10	0	9	0	1	14	2	11	0	0	0	0
0	14	14	2	6	0	0	14	13	15	9	10	13	0	6
0	3	2	10	13	0	0	12	10	6	12	0	1	0	12
0	2	14	6	0	5	0	0	11	0	0	0	3	15	14
0	11	15	9	14	5	2	7	0	5	0	7	15	11	5
6	7	0	0	0	0	0	7	1	0	7	7	14	1	0
15	13	0	5	8	1	4	0	11	5	0	9	13	10	5
1	9	0	9	7	15	0	4	0	2	0	0	11	0	0
13	11	3	2	15	14	5	15	0	12	0	3	0	0	9
15	8	0	0	0	1	10	9	0	5	0	0	0	0	0
0	2	10	7	0	2	2	15	5	5	8	8	0	14	0

Grafo g15x15_0.3[2]

0	11	12	2	4	7	14	4	0	2	4	0	10	10	13
0	0	0	3	7	0	0	14	13	0	11	8	1	6	7
0	0	0	15	14	14	10	15	11	14	7	10	14	3	0
4	1	3	0	12	6	2	9	0	0	5	6	0	12	6
11	14	6	6	0	13	7	0	0	14	0	0	0	0	6
0	8	0	13	14	0	12	0	1	0	3	12	3	13	8
13	14	4	0	0	12	0	0	9	13	10	15	8	3	6
15	2	0	0	7	0	7	0	4	0	12	2	3	4	0
15	1	10	13	5	11	1	2	0	0	8	0	9	4	15
8	4	0	0	14	6	0	0	3	0	0	1	14	8	0
0	14	0	0	4	12	13	1	7	2	0	0	0	7	0
5	7	5	8	0	0	6	7	11	0	9	0	10	10	0
2	2	4	12	15	0	15	9	0	10	1	4	0	10	5
10	7	0	15	9	0	6	4	0	0	0	12	13	0	0
0	9	10	6	2	9	2	2	0	0	8	0	5	0	0

Grafo g15x15_0.7[0]

0	0	0	1	3	0	0	8	6	0	0	0	0	0	0
0	0	14	0	0	13	0	6	0	0	0	0	0	0	0
0	0	0	0	9	7	0	10	0	0	0	0	11	0	0
0	7	0	0	2	15	0	0	4	0	0	11	1	13	0
0	0	0	0	0	8	14	0	0	0	0	0	0	0	0
9	0	0	3	2	0	3	0	0	0	5	0	11	14	7
5	0	0	15	10	0	0	0	5	10	0	0	0	0	0
4	0	0	13	8	0	7	0	0	0	0	0	0	6	1
0	0	6	6	0	0	0	0	0	12	0	0	0	8	0
0	0	0	0	0	0	0	0	0	0	0	0	1	7	4
0	0	12	15	0	0	0	0	0	0	0	0	11	0	0
0	12	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	4	0	0	5	0	0	9	14	0	0	0	8	0
0	10	4	0	1	0	0	0	0	6	0	0	0	0	0
0	0	5	0	0	0	13	0	13	0	0	0	0	0	0

Grafo g15x15_0.7[1]

0	9	0	0	13	0	0	0	0	5	0	15	5	0	4
13	0	0	0	0	5	0	0	11	0	0	0	0	0	6
4	0	0	0	0	0	0	0	0	0	5	0	5	0	6
0	0	0	0	0	0	1	0	0	1	0	0	0	7	6
0	8	0	13	0	0	0	0	0	8	4	0	0	10	10
0	11	0	0	0	0	8	0	0	12	0	2	0	0	0
0	3	0	13	9	11	0	0	14	5	0	0	13	3	5
0	11	0	15	0	0	0	0	0	0	15	0	8	0	2
0	0	0	0	0	6	0	0	0	0	2	0	0	12	0
0	0	1	0	0	0	0	0	0	0	0	14	0	0	0
0	0	0	0	15	0	11	0	12	0	0	0	0	13	0
3	0	0	8	15	2	0	0	4	0	0	0	9	0	0
0	0	0	3	0	0	0	6	0	11	5	0	0	0	0
0	1	0	2	9	0	7	0	8	0	0	0	1	0	15
0	0	0	1	0	0	0	0	10	0	0	0	14	0	0

Grafo g15x15_0.7[2]

0	9	5	14	0	1	0	0	0	0	11	13	0	9	0
3	0	8	0	0	0	9	5	0	5	0	0	4	0	1
11	0	0	0	0	13	0	0	0	2	2	0	0	7	13
0	11	3	0	0	7	5	0	0	0	4	0	0	4	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	7	6	0	5	3	0	0	0	0	3	8	0
0	11	0	0	9	0	0	3	2	0	0	0	0	0	0
0	0	0	0	0	1	0	0	11	0	0	8	0	0	1
12	0	0	2	0	0	0	13	0	0	13	0	0	0	0
0	0	0	5	0	0	0	0	0	0	3	14	11	0	7
0	0	0	2	4	9	0	13	15	14	0	15	0	0	2
2	0	0	0	1	3	2	0	0	0	0	0	0	0	1
9	0	0	0	15	10	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	6	0	8	0	0	8
10	0	0	0	0	0	0	2	0	0	0	6	10	0	0

Grafo g20x20_0.3[0]

0	9	11	6	19	12	5	0	10	8	18	5	0	10	0	0	5	18	0	0
0	0	20	0	18	0	0	13	0	1	12	2	0	6	0	1	0	20	0	0
20	6	0	0	12	2	14	20	4	0	2	8	16	0	10	3	9	2	14	2
10	5	14	0	16	2	4	10	18	14	9	9	18	18	6	1	14	0	15	11
0	13	0	13	0	2	19	9	0	8	14	6	0	10	18	1	3	0	0	0
15	20	0	18	4	0	17	20	11	0	16	6	2	3	8	5	20	9	0	0
8	6	18	0	13	0	0	0	12	0	19	0	14	0	19	14	20	0	0	19
14	5	16	20	15	13	0	0	14	0	6	12	0	8	0	6	0	19	10	1
0	9	1	7	13	2	19	10	0	2	0	4	4	15	0	0	6	18	19	12
8	12	1	4	15	0	0	5	7	0	0	1	17	9	9	14	0	12	3	17
6	5	15	20	2	0	0	0	16	2	0	8	0	0	9	0	2	2	0	0
13	0	0	15	0	8	9	6	9	9	13	0	0	0	20	0	10	10	0	9
7	2	6	11	18	12	0	4	10	0	17	19	0	5	16	7	8	10	9	5
12	6	0	4	0	0	12	1	0	2	8	15	17	0	0	0	19	7	0	0
0	0	0	3	0	9	2	6	0	0	10	2	3	11	0	14	17	0	8	15
17	8	0	0	5	6	0	3	16	13	0	1	8	2	16	0	0	1	15	18
20	11	14	9	5	15	6	20	0	0	5	3	15	8	3	8	0	3	0	0
20	13	0	0	1	19	11	0	16	0	8	0	11	8	0	0	9	0	15	2
0	20	19	20	6	16	19	14	0	18	17	13	0	17	0	9	0	0	0	15
0	14	1	17	2	0	8	13	13	0	17	16	19	0	12	15	0	6	0	0

Grafo g20x20_0.3[1]

0	1	17	0	0	3	11	3	0	0	0	0	11	11	14	7	1	9	5	19
14	0	2	0	3	6	12	11	0	0	16	2	7	1	0	11	8	0	14	16
3	0	0	9	9	16	10	2	7	0	0	0	14	0	18	19	13	17	10	0
13	20	0	0	7	9	9	0	12	0	0	0	14	7	0	0	9	9	11	0
5	7	17	0	0	0	3	6	4	0	0	9	0	0	6	0	11	10	20	9
6	0	3	2	17	0	5	0	9	15	19	14	20	13	14	5	15	5	1	0
0	4	13	5	5	5	0	9	6	16	16	17	0	1	3	3	18	0	13	12
6	19	10	2	7	9	16	0	0	11	13	18	19	0	15	0	18	4	2	0
16	2	9	14	2	11	10	20	0	0	0	0	17	6	1	3	8	0	16	0
15	8	7	3	5	16	14	0	2	0	9	9	2	9	0	17	7	4	0	20
0	0	14	0	8	3	2	10	5	20	0	9	2	0	3	2	6	10	0	18
0	5	15	0	8	18	12	9	0	12	13	0	14	19	5	11	8	13	0	3
4	0	3	0	0	19	13	0	8	0	20	12	0	7	8	0	19	14	0	0
16	0	12	0	16	5	4	16	9	7	7	4	8	0	19	0	0	14	13	8
0	0	5	13	18	0	4	15	7	17	3	0	0	0	0	6	20	0	11	19
9	8	0	12	11	0	3	6	3	5	9	0	20	0	9	0	0	6	0	2
0	12	16	3	0	1	14	17	0	20	2	19	0	19	0	0	0	0	2	2
10	13	0	16	4	9	3	0	0	0	18	15	0	7	8	4	1	0	0	13
13	0	14	3	12	12	14	16	20	6	14	2	0	14	18	8	0	0	0	0
7	11	0	0	16	0	3	19	9	0	7	0	0	9	11	4	13	0	16	0

Grafo g20x20_0.3[2]

0	0	17	7	3	10	18	0	0	8	7	13	7	0	20	8	19	12	0	12
8	0	1	4	14	14	15	1	9	8	0	18	4	0	0	1	2	4	1	0
12	10	0	4	0	0	6	0	0	13	11	11	3	0	2	9	12	7	19	16
11	0	0	0	0	0	15	10	19	0	0	0	13	0	17	0	18	8	4	16
20	13	3	0	0	14	11	20	8	16	11	14	12	1	17	11	0	5	16	1
17	8	0	1	0	0	8	10	0	10	18	0	10	14	0	7	11	16	7	19
6	0	20	0	2	0	0	14	0	11	0	19	0	5	12	6	18	0	0	7
17	16	4	2	3	0	17	0	0	6	7	0	0	3	0	15	13	0	17	9
16	10	0	0	0	3	2	14	0	3	18	16	5	20	18	18	2	1	4	19
0	15	6	0	16	0	0	6	0	0	12	0	11	18	15	0	0	17	9	1
0	19	5	14	0	9	0	11	0	0	0	0	12	4	5	15	2	12	0	0
8	18	6	4	0	0	14	0	12	11	20	0	10	0	14	2	0	0	0	9
7	3	0	1	11	11	2	0	15	10	17	10	0	0	0	0	18	9	7	9
1	7	15	0	10	17	16	15	0	16	14	8	0	0	14	2	13	11	13	19
0	1	18	10	18	11	16	0	7	13	11	1	20	11	0	6	13	10	0	11
2	0	1	6	3	4	2	19	19	5	0	2	20	17	13	0	10	19	3	12
0	11	18	16	14	16	0	16	20	0	12	6	13	0	10	0	0	0	2	0
10	0	13	12	1	0	0	0	16	3	0	0	0	6	4	16	5	0	12	10
16	0	0	0	0	0	17	20	4	18	9	20	9	0	0	0	6	0	0	7
4	8	1	2	0	9	1	0	0	19	10	0	10	6	0	19	12	7	5	0

Grafo g20x20_0.7[0]

0	0	0	0	0	3	0	6	9	0	0	0	0	3	0	0	6	3	0	7
0	0	14	16	5	11	0	17	0	0	4	0	0	18	0	0	0	0	0	0
18	0	0	0	16	6	20	8	0	20	8	4	0	12	0	0	0	2	0	12
1	0	0	0	4	0	0	0	0	0	4	0	12	17	0	20	19	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	10	0	7	0	0	19	0
0	0	13	0	0	0	0	0	0	0	0	20	0	0	0	0	0	13	7	0
0	18	16	8	0	15	0	12	0	0	0	0	0	6	0	5	17	0	9	11
0	0	0	0	6	0	0	0	8	10	0	0	0	3	0	0	0	5	0	11
0	0	0	0	12	15	0	0	0	0	19	0	0	0	0	8	0	0	1	0
0	0	0	0	0	10	0	0	11	0	3	16	0	0	0	2	5	7	0	11
15	0	0	0	0	0	14	9	0	5	0	12	0	0	0	0	0	0	0	0
0	4	12	0	0	9	15	3	0	5	0	0	0	0	18	0	0	0	0	0
0	18	11	0	0	0	3	0	12	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	12	0	0	0	0	0	0	4	0	3	0	0	8	11	0
15	0	0	0	0	1	0	7	0	2	6	0	0	0	0	0	3	0	0	0
0	0	10	0	16	0	14	17	6	0	0	0	10	0	0	0	10	0	0	0
17	0	0	7	0	0	16	0	0	0	0	0	0	13	0	0	0	0	9	19
0	17	19	0	7	16	0	0	18	0	0	0	0	5	0	0	0	0	2	13
0	0	0	0	0	0	0	12	0	0	0	0	9	0	0	4	0	0	0	6
0	0	0	0	0	1	0	16	0	0	0	0	0	4	0	0	0	0	0	0