Universidad Nacional de San Luis
Facultad de Ciencias Físico Matemáticas y Naturales
Departamento de Informática

Trabajo Final para alcanzar el grado de
Licenciado en Ciencias de la Computación

Algoritmos de Optimización basados en Colonias de Hormigas aplicados al Problema de Asignación Cuadrática y otros problemas relacionados

Franco Luis Alejandro Arito

Director: Dr. Guillermo Leguizamón

Co-Director: Dr. Marcelo Errecalde

San Luis - Argentina
Abril de 2010

Agradecimientos

En primer lugar quisiera agradecer profundamente a Guillermo Leguizamón por todo lo que me ha brindado durante los ultimos 2 años, por confiar en mí cuando ni siquiera yo creia en mí y por permitirme iniciarme en la docencia universitaria e incursionar en la investigación.

A mi familia por estar siempre conmigo en todo momento: a mi mamá Silvia, mi papá Jorge, mi hermana Guadalupe y mi hermano Federico.

A Pamela por compartir todas mis alegrias.

A la Universidad Nacional de San Luis por abrirme sus puertas, por haberme permitido cumplir el sueño de estudiar Ciencias de la Computación y a la cual siento realmente mi segunda casa.

A la Facultad de Ciencias Físico Matemáticas y Naturales por la Beca Estímulo concedida.

Por último agradecer a los integrantes del LIDIC por tratarme como un colega más cuando sólo era un estudiante.

Prólogo

La idea más genérica del término heurístico está relacionada con la tarea de resolver inteligentemente problemas reales usando el conocimiento disponible. El término heurística proviene de una palabra griega con un significado relacionado con el concepto de encontrar y se vincula a la supuesta exclamación *eureka* de Arquímedes al descubrir su famoso principio.

La concepción más común en Inteligencia Artificial es interpretar que heurístico es el calificativo apropiado para los procedimientos que, empleando conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad de recursos razonable. En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones factibles del problema, se busca la que es óptima según algún criterio de comparación entre ellas.

A diferencia de las heurísticas, las metaheurísticas pueden ser vistas como un marco general algorítmico, que puede ser aplicado a diferentes problemas de optimización con mínimos cambios para ser adaptado a un problema específico. Las metaheurísticas son ampliamente reconocidas como una de las mejores aproximaciones para atacar problemas de optimización combinatoria.

Entre las metaheurísticas se encuentra la Optimización basada en Colonias de Hormigas (ACO, del inglés, Ant Colony Optimization) que fue presentada por Marco Dorigo en su tesis doctoral en 1992 [18]. Básicamente, se trata de una técnica probabilística para resolver problemas computacionalmente complejos que pueden ser reducidos a la búsqueda de buenos caminos en grafos. ACO se inspira directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria. Se basan en una colonia de hormigas artificiales, esto es, agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales.

En un experimento realizado en 1990, Deneubourg y su grupo demostró que, cuando se les da la posibilidad de elegir entre dos caminos de diferente longitud que unan el nido a una fuente de alimento, una colonia de hormigas tiene una alta probabilidad de elegir colectivamente al camino más corto. Deneubourg ha demostrado que este comportamiento puede explicarse a través de un modelo probabilístico simple en el que cada

hormiga decide a dónde ir tomando decisiones aleatorias basadas en la intensidad de la feromona percibida en el suelo, la feromona es depositada por las hormigas mientras se desplazan desde el nido a la fuente de alimento y viceversa.

En este trabajo se describen las bases de la metaheurística ACO, y en particular se estudia la aplicación de un algoritmo perteneciente a la metaheurística ACO ($\mathcal{MAX} - \mathcal{MIN}$ Ant System [69]) a un problema de optimización combinatoria. Además, se propone una técnica alternativa en la construcción de soluciones en algoritmos ACO, inspirada principalmente en otra metaheurística llamada *Búsqueda Tabú*.

El problema a resolver se denomina Problema de Asignación Cuadrática (QAP, por sus siglas en inglés, Quadratic Assignment Problem); el cual es uno de los problemas de optimización combinatoria más difíciles de resolver en la práctica. Algunos de los motivos de la elección del mencionado problema se resumen en los siguientes puntos:

- El QAP es un problema de optimización $\mathcal{NP}$-duro. Es considerado uno de los problemas de optimización más dificiles ya que las instancias más grandes que pueden ser resueltas hoy en día con algoritmos exactos están limitadas a tamaños inferiores a 40. Si lo comparamos con el problema del viajante de comercio para el cual se resuelven instancias de tamaño 20000 o superiores.
- Muchos problemas prácticos como el cableado de tableros, disposición de campus y hospitales, diseño de teclados, planificación, procesamiento de imagenes (diseño de patrones de grises), Diseño Asistido por Computadora (CAD), entre otros pueden ser formulados como instancias de QAP.
- Distintos problemas de optimización combinatoria $\mathcal{NP}$-duros, tales como el Problema del Viajante de Comercio (TSP, del inglés, Traveling Salesman Problem), el problema Bin-Packing y el problema del Máximo Clique de un grafo, pueden ser modelados como problemas de asignación cuadrática.

Se debe destacar, que la parte central de este trabajo son los algoritmos ACO, y no así el problema a resolver. Se buscó un problema suficientemente difícil para que tuviera sentido la aplicación de los algoritmos ACO, así como también que permitiera ser comparado con otros métodos existentes que lo resolvieran. También ha sido de gran ayuda el hecho de que ya existieran implementaciones previas que aplicaban algoritmos ACO al mencionado problema, lo que permite incluso tener puntos de referencia en cuanto a su aplicación.

Organización del presente trabajo

- *Capítulo 1* **Introducción**: Presenta conceptos introductorios referentes a la temática abordada que son necesarios para una mejor comprensión del trabajo.

- *Capítulo 2* **El Problema de Asignación Cuadrática**: Se presenta el problema utilizado para evaluar la performance de los algoritmos propuestos.
- *Capítulo 3* **La metaheurística de Optimización basada en Colonias de Hormigas** : Introduce los conceptos principales de la metaheurística, también se presenta una breve reseña de como surgieron estos algoritmos.
- *Capítulo 4* **Algoritmos ACO aplicados al QAP**: Explica las distintas versiones de algoritmos ACO propuestos para QAP, incluyendo versiones que abordan uso de memoria externa, dentro de la cual se enmarcan los algoritmos propuestos en el presente trabajo.
- *Capítulo 5* **Experimentos y Análisis de Resultados**: Se detalla la configuración de los experimentos a realizar y se muestra un análisis estadístico de los resultados obtenidos. Se comparan los distintos algoritmos propuestos y se determina cuales obtienen mejor rendimiento.
- *Capítulo 6* **Conclusiones**: Se presentan las conclusiones obtenidas en base a los estudios realizados y los resultados obtenidos. Posteriormente, se presenta una posible línea de investigación a seguir en el futuro, como continuación de este trabajo.

Índice general

1. Introducción **1**
1.1. Optimización combinatoria 1
1.2. Metaheurísticas 3
1.2.1. Optimización basada en Colonias de Hormigas 4
1.2.2. Algoritmos Evolutivos 4
1.2.3. Búsqueda Tabú 5
1.2.4. Simulated Annealing 6
1.2.5. Búsqueda Local Iterada 6
1.3. Resumen 7

2. El Problema de Asignación Cuadrática **8**
2.1. Introducción 8
2.1.1. Ejemplos de problemas formulados como QAP 9
2.2. Complejidad del problema 10
2.3. Estado del arte: Métodos de resolución de QAP 12
2.3.1. Metaheurísticas aplicadas al QAP 12
2.3.2. Algoritmos exactos aplicados al QAP 13
2.4. Descripción de instancias de QAP a utilizar 14
2.4.1. Instancias de QAPLIB 14
2.4.2. Instancias de Stützle y Fernandes 15
2.5. Resumen 16

3. La metaheurística de Optimización basada en Colonias de Hormigas **18**
3.1. De las colonias de hormigas a las hormigas artificiales 18
3.1.1. Las colonias de hormigas 18
3.1.2. Hormigas artificiales 20
3.2. La metaheurística de Optimización basada en Colonias de Hormigas . . 23
3.2.1. Ejemplo: El Problema del Viajante de Comercio 26
3.3. Algoritmos basados en la metaheurística ACO 26
3.3.1. Ant System 27
3.3.2. $\mathcal{MAX} - \mathcal{MIN}$ Ant System 28
3.3.3. Ant Colony System 30
3.4. Resumen 32

4. Algoritmos ACO aplicados al QAP **33**
4.1. Respecto de la aplicación de ACO al QAP 33
4.2. Enfoques previos de algoritmos ACO para QAP 34
4.2.1. Ant System 35
4.2.2. ANTS 36
4.2.3. $\mathcal{MAX} - \mathcal{MIN}$ Ant System 38
4.2.4. FANT 39
4.3. Algoritmos ACO con uso de memoria explícita 40
4.3.1. ACO con Memoria Externa 40
4.3.2. Iterated Ants $\mathcal{MAX} - \mathcal{MIN}$ Ant System 42
4.3.3. Cunning Ant System 44
4.4. Enfoque de memoria externa propuesto 46
4.4.1. Uso de la memoria 46
4.4.2. Frecuencia de utilización de la memoria 49
4.4.3. Incorporación de búsqueda local en el algoritmo propuesto . . . 50
4.4.4. Algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System basado en memoria . . 52
4.5. Resumen 55

5. Experimentos y Análisis de Resultados **56**
5.1. Instancias utilizadas 56
5.2. Descripción de los experimentos 57
5.2.1. Entorno computacional 57
5.3. Resultados 57
5.4. Análisis Estadístico 61
5.5. Resumen 66

6. Conclusiones **67**
6.1. Resumen de resultados 67
6.2. Conclusiones 67
6.3. Trabajo Futuro 68

Índice de tablas

5.1. Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-ff. Los mejores resultados para cada clase de instancia son indicados en negrita. 58
5.2. Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-fr. Los mejores resultados para cada clase de instancia son indicados en negrita. 58
5.3. Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-rf. Los mejores resultados para cada clase de instancia son indicados en negrita. 59
5.4. Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-rr. Los mejores resultados para cada clase de instancia son indicados en negrita. 59
5.5. Resultados obtenidos de cada una de las variantes propuestas. 60
5.6. Resultados obtenidos para el algoritmo $\mathcal{MM}$AS tradicional 62
5.7. Comparación de los mejores valores encontrados con los algoritmos $\mathcal{MM}$AS-ff, $\mathcal{MM}$AS-fr, $\mathcal{MM}$AS-rf, $\mathcal{MM}$AS-rr y $\mathcal{MM}$AS junto con los respectivos p-valores obtenidos de la prueba ANOVA de un factor. 62

Índice de figuras

3.1. Imagen del experimento real llevado a cabo por Deneubourg *et al.* . . . 19
3.2. Imagen del experimento real llevado a cabo por Goss *et al.* 20

5.1. Boxplots del test de ANOVA aplicado a las seis clases de instancias. El eje y representa el porcentaje de error promedio respecto del mejor valor conocido. Sobre el eje de las x se muestran las respectivas variantes de los algoritmos incluidos el algoritmo $\mathcal{MM}$AS-QAP tradicional. 64
5.2. Representación gráfica de los resultados obtenidos con el método de Tukey. El punto medio de cada linea representa la media de los valores. 65

Capítulo 1

Introducción

Este capítulo introduce conceptos y definiciones que son utilizados en el resto del trabajo final, los cuales son necesarios para una mejor comprensión de los temas a tratar.

1.1. Optimización combinatoria

Informalmente, optimizar significa algo más que mejorar; sin embargo, en el contexto científico la optimización es el proceso de tratar de encontrar la mejor solución posible para un determinado problema. En un problema de optimización existen diferentes soluciones, un criterio para diferenciar entre ellas y el objetivo es encontrar la mejor. Muchos problemas de optimización de importancia teórica y/o práctica consisten en la búsqueda de una "mejor" configuración de un conjunto de variables para lograr ciertos objetivos. Los problemas se dividen naturalmente en dos categorías: aquellos en los que las soluciones son codificadas con un valor real de las variables, y aquellos en los que las soluciones están codificadas con variables discretas. Entre estos últimos, encontramos una clase de problemas llamados Problemas de Optimización Combinatoria (POC). De acuerdo a [61], en los POC, se busca un objeto de un conjunto finito (o posiblemente de un conjunto infinitamente contable). Este objeto puede ser un número entero, un subconjunto de ellos, una permutación, o una estructura de grafo.

Esto nos lleva a formular la definición de un problema de optimización combinatoria.

Definición 1.1.1 *Un problema de optimización combinatoria* $P = (\mathcal{S}, \mathbf{\Omega}, f)$ *puede ser definido por:*

- *un conjunto de variables* $X = \{x_1, \ldots, x_n\}$*;*
- *dominios de variables* $D_1, \ldots, D_n$*;*
- *un conjunto de restricciones* $\mathbf{\Omega}$ *entre las variables;*
- *una función objetivo* $f : D_1 \times \ldots \times D_n \to \mathbb{R}_0^+$ *a ser minimizada (o maximizada).*

El conjunto de todas las posibles asignaciones es:

$$\mathcal{S} = \{\mathbf{s} = \{(x_1, v_1), \ldots, (x_n, v_n)\} | v_i \in D_i, \ y \ \mathbf{s} \ satisface \ \mathbf{\Omega}\}$$

Al conjunto $\mathcal{S}$ se lo llama *espacio de búsqueda*, ya que cada elemento del conjunto puede ser visto como una solución candidata. Para resolver un POC se tiene que encontrar una solución $s^* \in \mathcal{S}$ que tenga un valor de función objetivo mínimo, esto es, $f(s^*) \leq f(s) \ \forall s \in \mathcal{S}$. A una solución s^* se la llama óptimo global de P, y al conjunto $\mathcal{S}^* \subseteq \mathcal{S}$ se lo llama el conjunto de todas las soluciones que son óptimos globales.

Los problemas de optimización combinatoria están presentes en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. El estudio de esta dificultad inherente para resolverlos, tiene lugar en el campo de la teoría de las Ciencias de la Computación, ya que muchos de ellos pertenecen a la clase de problemas $\mathcal{NP}$-duros (problemas para los cuales no se conoce o tal vez no exista un algoritmo que los resuelva en tiempo polinomial [31]). En la actualidad, siguen apareciendo nuevos problemas de este tipo, lo que ha dado lugar a muchas propuestas de algoritmos para tratar de resolverlos. Las técnicas existentes se pueden clasificar básicamente en dos, algoritmos exactos y algoritmos aproximados. Los algoritmos exactos intentan encontrar una solución óptima y demostrar que la solución obtenida es de hecho la óptima global; estos algoritmos incluyen técnicas como, por ejemplo: procesos de vuelta atrás (backtracking), Ramificación y Acotación (branch and bound), programación dinámica, etc. [61, 8]. Debido a que los algoritmos exactos muestran un rendimiento pobre para muchos problemas se han desarrollado múltiples tipos de algoritmos aproximados que proporcionan soluciones de alta calidad (soluciones con valores de función objetivo relativamente bajos, aunque no necesariamente sean óptimas) para estos problemas combinatorios en un tiempo computacional razonable.

Los algoritmos aproximados, se pueden clasificar a su vez en dos tipos principales: los algoritmos constructivos y los algoritmos de búsqueda local. Los primeros se basan en generar soluciones desde cero añadiendo componentes a cada solución paso a paso. Un ejemplo bien conocido son las heurísticas de construcción voraz o heurísticas voraces [8]. Su gran ventaja es la velocidad: normalmente son muy rápidas y, además, a menudo devuelven soluciones razonablemente buenas. Sin embargo, no puede garantizarse que dichas soluciones sean óptimas con respecto a pequeños cambios a nivel local. En consecuencia, una mejora típica es refinar la solución obtenida por la heurística voraz utilizando búsqueda local. Los algoritmos de búsqueda local intentan repetidamente mejorar la solución actual con movimientos a soluciones vecinas[1] (con la esperanza de que sean mejores). El caso más simple son los algoritmos de mejora iterativa: si en el vecindario de la solución actual s se encuentra una solución mejor s', ésta, reemplaza a la solución actual y se continua la búsqueda a partir de s'; si no se encuentra una

[1]Estas soluciones pueden ser vistas como soluciones que están "cercanas" a la solución actual, en base a algún criterio.

solución mejor en el vecindario, el algoritmo termina en un óptimo local. La vecindad de una solución se define a continuación.

Definición 1.1.2 *Una estructura de vecindad es una función* $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ *que asigna a cada solución* $s \in \mathcal{S}$ *un conjunto de vecinos* $\mathcal{N}(s) \subseteq \mathcal{S}$*. Se dice que* $\mathcal{N}(s)$ *es la vecindad de la solución* s*.*

La introducción de la estructura de vecindad permite definir el concepto de soluciones que son óptimos (mínimos) locales. Las soluciones que son óptimos locales, pueden ser vistas como las "mejores" soluciones dentro de una región acotada del espacio de búsqueda[2].

Definición 1.1.3 *Una solución* s^+ *es un óptimo local con respecto a una estructura de vecindad* $\mathcal{N}$ *si* $\forall s \in \mathcal{N}(s^+)$ *se cumple* $f(s^+) \leq f(s)$*.*

Uno de los inconvenientes de los algoritmos de mejora iterativa es que pueden estancarse en soluciones de baja calidad (óptimos locales muy lejanos al óptimo global). Para permitir una mejora adicional en la calidad de las soluciones, la investigación en este campo en las últimas dos décadas ha centrado su atención en el diseño de técnicas de propósito general para guiar a la construcción de soluciones o a la búsqueda local en las distintas heurísticas. Estas técnicas son conocidas como metaheurísticas, y se describen en la próxima sección.

1.2. Metaheurísticas

El término *metaheurística* fue introducido por Glover [33], y deriva de la composición de dos palabras griegas. El sufijo *meta* significa "más allá de, en un nivel superior", y heurística deriva del verbo *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) que significa "encontrar, descubrir". Existen muchas definiciones de metaheurística, pero en este trabajo se adopta la formulada en [60]:

> *"Una metaheurística se define formalmente como un proceso de generación iterativo el cual guía a una heurística subordinada combinando inteligentemente diferentes conceptos para explorar y explotar el espacio de búsqueda, son utilizadas estrategias de aprendizaje para estructurar la información con el objetivo de encontrar eficientemente soluciones casi óptimas."* [60].

En resumen, se puede decir que las metaheurísticas son estrategias de alto nivel para explorar espacios de búsqueda usando diferentes métodos. Además es de gran importancia que exista un balance dinámico entre *diversificación* e *intensificación*. El término diversificación generalmente se refiere a la exploración del espacio de búsqueda (promover al proceso de búsqueda a examinar regiones no visitadas del espacio de búsqueda,

[2]La noción de "región" del espacio de búsqueda y de "localidad" sólo se puede expresar en forma difusa; y dependerá de las características del espacio de búsqueda, así como también de la definición de métricas en el espacio de búsqueda (las distancias entre soluciones).

para generar soluciones que difieran de manera significativa de las soluciones ya vistas), mientras que el término intensificación se refiere a la explotación de la experiencia de búsqueda acumulada (enfocar la búsqueda en examinar soluciones que pertenezcan a la vecindad de las mejores soluciones encontradas).

Las metaheurísticas incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otras. Algunos ejemplos de metaheurísticas son: Algoritmos Evolutivos [40, 5, 6], Simulated Annealing [43], Búsqueda Tabú [34], Búsqueda Local Iterativa [46], entre otras.

1.2.1. Optimización basada en Colonias de Hormigas

En el capítulo 3 del presente trabajo final se describe detalladamente a la metaheurística de Optimización basada en Colonias de Hormigas.

1.2.2. Algoritmos Evolutivos

Los Algoritmos Genéticos son técnicas que simulan la selección natural y la adaptación encontrada en la naturaleza. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor ó puntuación (llamado en la literatura *fitness*), relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea *seleccionado* para reproducirse, *combinando* su material genético con otro individuo seleccionado de igual forma. Esta combinación producirá nuevos individuos (descendientes de los anteriores) los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera, a través de los denominados *operadores genéticos* (selección, recombinación y mutación[3]) se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así, a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, se exploran las áreas más prometedoras del espacio de búsqueda. Este proceso continúa hasta que se cumple algún criterio de parada establecido.

[3]El operador de mutación, permite modificar en forma aleatoria parte de la solución representada por un individuo.

1.2.3. Búsqueda Tabú

La Búsqueda Tabú (BT) es una metaheurística cuya característica distintiva es el uso de memoria adaptativa y de estrategias especiales de resolución de problemas. Su filosofía se basa en la explotación de diversas estrategias inteligentes para la resolución de problemas, basadas en procedimientos de aprendizaje. El marco de memoria adaptativa de BT explota la historia del proceso de resolución del problema haciendo referencia a cuatro dimensiones principales, consistentes en la propiedad de ser reciente, en frecuencia, en calidad, y en influencia.

El algoritmo de BT simple aplica búsqueda local con el criterio de *"best-improvement"* como componente básico y usa una memoria de corto plazo, para poder escapar de óptimos locales y evitar ciclos en la búsqueda. La memoria de corto plazo está implementada como una *lista tabú* que mantiene registro de las soluciones visitadas más recientemente y prohibe movimientos hacia ellas. La implementación de la memoria a corto plazo como una lista que contiene soluciones completas puede no ser efectiva, por lo que en lugar de almacenar soluciones, se almacenan *atributos de soluciones* como por ejemplo componentes de soluciones, movimientos o diferencias entre dos soluciones. De esta forma, se restringe la vecindad de la solución actual a soluciones que no pertenecen a la lista tabú[4], previniendo de ciclos infinitos y forzando a la búsqueda a aceptar movimientos que incluso pueden generar soluciones peores. Ya que se puede considerar más de un atributo, se introduce una lista tabú para cada uno de ellos. El conjunto de atributos y las listas tabú correspondientes definen la lista de *condiciones tabú* que se utilizan para filtrar la vecindad de una solución y generar el conjunto permitido. Almacenar atributos en lugar de soluciones completas es mucho más eficiente, pero introduce pérdida de información, ya que prohibir un atributo significa asignar la condición de tabú a más de una solución probablemente. Por lo tanto, es posible que soluciones no visitadas de buena calidad sean excluidas del conjunto permitido. Para superar este problema, se definen *criterios de aspiración*, que permiten incluir una solución en el conjunto permitido aunque esté prohibida por las condiciones tabú. Los criterios de aspiración definen las condiciones de aspiración que se utilizan para construir el conjunto permitido.

La lista tabú, identificada usualmente con el uso de memoria a corto plazo, es sólo una de las maneras posibles de aprovechar la historia de la búsqueda. La información recolectada durante todo el proceso de búsqueda también puede ser muy útil, especialmente para una guía estratégica del algoritmo. Este tipo de memoria a largo plazo generalmente se agrega a la BT de acuerdo a cuatro principios: lo reciente, frecuencia, calidad e influencia. La memoria basada en lo reciente registra para cada solución (o atributo) la iteración más reciente en la que estuvo involucrada. En contraposición, la memoria basada en frecuencia mantiene registro de cuántas veces cada solución (o atributo) ha sido visitada (usado). Esta información identifica las regiones del espacio de soluciones donde la búsqueda estuvo concentrada o donde permaneció por mayor número de iteraciones. Es decir, es información que representa la memoria del proceso

[4] A este conjunto de soluciones se lo denomina el *conjunto permitido*.

de búsqueda y que puede ser explotado para diversificar la búsqueda. Por su parte, el atributo calidad hace referencia a la habilidad para diferenciar la bondad de las soluciones visitadas a lo largo del proceso de búsqueda. De esta forma, la memoria puede ser utilizada para la identificación de elementos comunes a soluciones buenas o a ciertos caminos que conducen a ellas. La calidad constituye un fundamento para el aprendizaje basado en refuerzos, donde se premian las acciones que conducen a buenas soluciones y se penalizan aquellas que, por el contrario, conducen a soluciones pobres. Por último, la cuarta dimensión de memoria, referida a la influencia, considera el impacto de las decisiones tomadas durante la búsqueda, no sólo en lo referente a la calidad de las soluciones, sino también en lo referente a la estructura de las mismas.

1.2.4. Simulated Annealing

Simulated Annealing es un algoritmo de búsqueda local que explota la analogía entre los algoritmos de optimización combinatoria y la mecánica estadística[5] [43]. Esta analogía se hace asociando las soluciones factibles de los problemas de optimización combinatoria a los estados de un sistema físico, teniendo costos asociados a estos estados de energía. Sean E_i y E_{i+1} dos estados de energía consecutivos, correspondientes a dos soluciones vecinas, y sea $\Delta E = E_{i+1} - E_i$. Pueden darse las siguientes situaciones: si $\Delta E < 0$, hay una reducción de energía y el proceso continúa. En otras palabras, hay una reducción en la función de costo del problema y la nueva asignación puede ser aceptada; si $\Delta E = 0$, hay una situación de estabilidad y no hay cambio en el estado de energía. Esto significa que no ha cambiado la función de costo del problema; y si $\Delta E > 0$, hay un aumento de energía y es útil para el proceso físico permitir el movimiento de partículas, es decir, la función de costo del problema ha aumentado. En lugar de eliminar este cambio de las partículas, su uso está sujeto a los valores de una función de probabilidad[6], para evitar convergencia en mínimos locales de baja calidad.

1.2.5. Búsqueda Local Iterada

Búsqueda Local Iterada (BLI) es una metaheurística simple y de aplicación general que aplica búsqueda local iterativamente a modificaciones del punto de búsqueda actual, lo que permite una exploración aleatorizada en el espacio de óptimos locales [46]. Al comienzo del algoritmo, se aplica búsqueda local a alguna solución inicial. Luego, un loop principal se repite hasta que se satisface algún criterio de parada. Consiste de un paso de modificación (kick-move) que devuelve una solución intermedia s' que corresponde a una mutación de una solución óptima local encontrada previamente. A continuación, se aplica búsqueda local a s' obteniendo una solución s''. Luego un criterio de aceptación decide a partir de que solución se continúa la búsqueda aplicando el próximo paso de

[5]El nombre e inspiración viene del proceso de recocido del acero, una técnica que consiste en calentar y luego enfriar controladamente un material para aumentar el tamaño de sus cristales y reducir sus defectos. El calor causa que los átomos se salgan de sus posiciones iniciales (un mínimo local de energía) y se muevan aleatoriamente; el enfriamiento lento les da mayores probabilidades de encontrar configuraciones con menor energía que la inicial.

[6]La probabilidad se computa generalmente siguiendo la distribución de Boltzman.

modificación. Tanto el paso de modificación como la prueba de aceptación pueden estar influenciados por el historial de búsqueda.

1.3. Resumen

En este capítulo se presento una introducción a los temas a tratar en el resto del presente trabajo final. Los mismos incluyen conceptos y definiciones que permiten un mejor entendimiento de la temática abordada.

Capítulo 2

El Problema de Asignación Cuadrática

En este capítulo se presenta el Problema de Asignación Cuadrática, el cual será usado como problema de prueba para los algoritmos desarrollados. Se realiza una introducción al problema, así como también se muestra la dificultad inherente para resolverlo, lo cual motivó su elección para el estudio. Se presenta un breve estado del arte respecto de los métodos de resolución utilizados desde su formulación. Al final del capítulo se presentan aplicaciones de metaheurísticas al problema.

2.1. Introducción

El Problemas de Asignación Cuadrática (QAP, por sus siglas en inglés, Quadratic Assignment Problem) fue introducido por Koopmans y Beckmann en 1957 como un modelo matemático para la ubicación de un conjunto de actividades económicas indivisibles [44]. El QAP puede ser descrito mejor como el problema de asignar un conjunto de elementos a un conjunto de ubicaciones, donde hay distancias entre las ubicaciones y flujos entre los elementos. El objetivo entonces es ubicar los elementos en las ubicaciones de forma tal que la suma del producto entre flujos y distancias sea mínima.

Más formalmente, dados n elementos y n ubicaciones, dos matrices $A = [a_{ir}]$ y $B = [b_{js}]$ de $n \times n$, donde a_{ir} es la distancia entre las ubicaciones i y r y b_{js} es el flujo entre los elementos j y s, el QAP puede ser formulado como:

$$\min_{\phi \in \Phi} \sum_{i=1}^{n} \sum_{r=1}^{n} a_{\phi_i \phi_r} b_{ir} \tag{2.1}$$

donde $\Phi(n)$ es el conjunto de todas las permutaciones (correspondientes a las asignaciones) en el conjunto de enteros $\{1, \ldots, n\}$, y ϕ_j da la ubicación del elemento j en la solución actual $\phi \in \Phi(n)$.

El término *cuadrático* deriva de la formulación del QAP como un problema de optimización entero con una función objetivo cuadrática. Sea x_{ij} una variable binaria la cual toma el valor 1 si el elemento i es asignado a la ubicación j, y 0 en otro caso. Entonces, el problema puede ser formulado como:

$$\text{mín} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ij} b_{kl} x_{ik} x_{jl} \tag{2.2}$$

Sujeto a las restricciones:

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1 \qquad 1 \leq j \leq n \\ \sum_{j=1}^{n} x_{ij} &= 1 \qquad 1 \leq i \leq n \\ x_{ij} &\in \{0,1\} \qquad 1 \leq i,j \leq n \end{aligned}$$

Estas restricciones formalizan la idea de que cada elemento debe ser asignado sólo a una ubicación y que para cada ubicación exactamente un elemento debe ser asignado. El término $a_{ij}b_{kl}x_{ik}x_{jl}$ contribuye a la función objetivo con un valor igual a $a_{ij}b_{kl}$ si y sólo si $x_{ik} = x_{jl} = 1$, esto es, si y sólo si el elemento i es asignado a la ubicación k y el elemento j es asignado a la ubicación l.

2.1.1. Ejemplos de problemas formulados como QAP

A continuación se describen brevemente algunas de las aplicaciones del QAP. Además de los problemas de disposición de instalaciones [17], el QAP aparece en aplicaciones como el cableado de tableros [65], fabricación de computadoras, planificación, comunicación de procesos, balanceo de turbinas, entre otros [11].

Cableado de Tableros

Una de las aplicaciones más antiguas del QAP se describe en [65], e involucra el cableado de tableros. Diferentes dispositivos tales como controles y displays tienen que ser ubicados en un panel, donde tienen que ser conectados unos con otros por cables. El problema es encontrar un posicionamiento de los dispositivos de manera tal que minimice la longitud total de cable. Sea n el número de dispositivos a ser ubicados y donde d_{kl} denota la longitud del cable desde la posición k a la posición l. La matriz de flujo $F = (f_{ij})$ está dada por:

$$f_{ij} = \begin{cases} 1 & \text{si el dispositivo } i \text{ está conectado al dispositivo } j \\ 0 & \text{en otro caso} \end{cases}$$

Luego la solución al correspondiente QAP minimizará la longitud total de cable.

Otra aplicación en el contexto de teoría de ubicación es un problema de planificación de campus propuesto en [17]. El problema consiste en planificar los lugares de n edificios en un campus, donde d_{kl} es la distancia desde el lugar k hasta el lugar l, y f_{ij} es la intensidad del tráfico entre el edificio i y el edificio j. El objetivo es minimizar la distancia total a caminar entre los edificios.

Diseño de teclados

En [11] se muestra que el QAP pueden ser aplicado, en el campo de la ergonomía, al diseño de teclados. El problema es organizar las teclas en un teclado de manera tal de minimizar el tiempo necesario para escribir algún texto. Sea el conjunto de enteros $N = 1, 2, \ldots, n$ que denota el conjunto de símbolos a ser colocados. Luego f_{ij} denota la frecuencia de aparición del par de símbolos i y j. Las entradas de la matriz de distancia $D = d_{kl}$ denota el tiempo que se necesita para presionar la tecla en la posición l después de presionar la tecla en la posición k, para todas las teclas a ser asignadas. Entonces una permutación $\varphi \in \mathcal{S}_n$ describe una asignación de símbolos a las teclas. Una solución óptima φ^* para el QAP minimiza el tiempo promedio para escribir un texto.

2.2. Complejidad del problema

En esta sección se tratan los aspectos de complejidad computacional del QAP. Los cuales evidencian el hecho de que el QAP es un problema "muy difícil" desde el punto de vista teórico. Primero, se muestra que el QAP pertenece a la clase de problemas $\mathcal{NP}$-duros[1]. Luego se considera la complejidad de encontrar una solución que sea un óptimo local.

Teorema 2.2.1 *El Problema de Asignación Cuadrática es fuertemente $\mathcal{NP}$-duro.*

Demostración: Consiste en mostrar que la existencia de un algoritmo para resolver una instancia del problema de asignación cuadrática en tiempo polinomial con valores de las matrices A y B pertenecientes a $\{0, 1, 2\}$, implica la existencia de un algoritmo para resolver un problema de decisión $\mathcal{NP}$-completo en tiempo polinomial. Esto implicaría entonces, por definición que el QAP es fuertemente $\mathcal{NP}$-duro. El problema de decisión $\mathcal{NP}$-completo mencionado anteriormente es el problema del ciclo Hamiltoniano [31].

Dado un grafo $G = (V, E)$ con conjunto de vértices V y conjunto de arcos E. ¿El grafo G contiene un ciclo Hamiltoniano?.

Asumamos que existe un algoritmo para resolver el QAP en tiempo polinomial con valores en $\{0, 1, 2\}$. Consideremos una instancia arbitraria del problema del ciclo Hamiltoniano, es decir, buscamos un ciclo Hamiltoniano en un grafo arbitrario $G = (V, E)$.

[1] Problemas para los cuales no se conoce o tal vez no exista un algoritmo que los resuelva en tiempo polinomial [31].

Sea $|V| = n$, $V = \{v_1, v_2, \ldots, v_n\}$. Definimos dos matrices de $n \times n$ $A = (a_{ij})$ y $B = (b_{ij})$ de la siguiente forma:

$$a_{ij} = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 2 & \text{en otro caso} \end{cases}$$

$$b_{ij} = \begin{cases} 1 & \text{si } j = i+1,\ 1 \leq i \leq n-1 \text{ ó } i = n,\ j = 1 \\ 0 & \text{en otro caso} \end{cases}$$

y consideramos a las matrices A y B como las matrices de una instancia del problema de asignación cuadrática. Se puede ver que el valor ópimo para la instancia del QAP es igual a n si y sólo si G contiene un ciclo Hamiltoniano. De esta manera, transformamos la instancia dada del problema del ciclo Hamiltoniano en una instancia de QAP y aplicamos el algoritmo de tiempo polinomial que se asume que existe. Luego, chequeamos si la solución provista por este algoritmo es igual a n. Todo esto puede ser hecho en tiempo polinomial con respecto al tamaño de la instancia del problema del ciclo Hamiltoniano. Por lo tanto, tendríamos un algoritmo para el problema del ciclo Hamiltoniano de tiempo polinomial. □

En [42] se definen los llamados *problemas de búsqueda local de tiempo polinomial* (problemas PLS, del inglés, Polynomial-time Local Search). Un par $(P,\mathcal{N})$, donde P es un problema de optimización (combinatorio) y $\mathcal{N}$ es una estructura de vecindad asociada, define un *problema de búsqueda local* que consiste en encontrar una solución que sea óptima local para el problema P con respecto a la estructura de vecindad $\mathcal{N}$. Sin entrar en detalles técnicos, un problema PLS es un problema de búsqueda local para el cual la optimalidad local puede ser chequeada en tiempo polinomial (la clase de problemas PLS se denota $\mathcal{PLS}$). En analogía con los problemas de decisión, aquí también se puede mostrar la existencia de problemas completos dentro de la clase $\mathcal{PLS}$. De aquí tenemos que, los problemas PLS-completos, son los problemas más difíciles dentro de la clase $\mathcal{PLS}$.

Teorema 2.2.2 *El problema de búsqueda local* $(QAP,\mathcal{N}_2)$*, donde* $\mathcal{N}_2$ *es la vecindad de intercambio de pares, es PLS-completo.*

□

El Teorema 2.2.2 implica que la complejidad temporal de un algoritmo de búsqueda local general para el QAP con la vecindad de intercambio de pares, es exponencial en el peor caso[2].

[2]Esta estructura de vecindad es la que utiliza el algoritmo de búsqueda local utilizado por los algoritmos ACO propuestos.

2.3. Estado del arte: Métodos de resolución de QAP

2.3.1. Metaheurísticas aplicadas al QAP

En esta sección se presenta un resumen de algunas metaheurísticas disponibles para resolver el QAP. Debido a que es inmensa la cantidad de metaheurísticas aplicadas a QAP, sólo se describe la aplicación de las metaheurísticas más representativas de la literatura [35].

Algoritmos Evolutivos

Diferentes algoritmos evolutivos, la mayoría basados en algoritmos genéticos, han sido implementados para el QAP. Uno de los más prominentes fue el algoritmo genético híbrido propuesto en [29], en el cual adaptaron un algoritmo genético para resolver el QAP usando un operador de recombinación especial y donde la mutación es reemplazada por ejecuciones cortas de un algoritmo Búsqueda Tabú.

Existen otros enfoques para resolver el QAP que están basados en algoritmos evolutivos. Estos incluyen algoritmos genéticos tradicionales [74], algoritmos genéticos paralelos [9, 57], estrategias evolutivas [58], y algoritmos genéticos de búsqueda local [50]. Los algoritmos genéticos híbridos [3, 50, 25, 53, 26, 55] están entre los algoritmos metaheurísticos de mayor redimiento para resolver el QAP, y han probado ser más prometedores que los tradicionales.

Búsqueda Tabú

Probablemente el algoritmo de Búsqueda Tabú más conocido es el algoritmo de Búsqueda Tabú Robusta [70]. Existen una gran variedad de otras implementaciones de Búsqueda Tabú para el QAP. La primera de las implementaciones se desarrolló en 1990 [63]. Algunas extensiones de estos primeros algoritmos fueron propuestos por el mismo autor en [64]; estas también incluyen una implementación de Búsqueda Tabú masivamente paralela [14]. Una variante particular de Búsqueda Tabú, llamada Búsqueda Tabú Reactiva (BTR) fue propuesta en [7], la principal idea de la BTR es el uso de la historia de búsqueda para adaptar dinámicamente la longitud de la lista tabú durante la ejecución del algoritmo. Otras adaptaciones de Búsqueda Tabú más recientes para resolver QAP incluyen [52, 54, 27].

Simulated Annealing

Simulated Annealing fue una de las primeras metaheurísticas disponibles, por lo tanto no es sorprendente que también haya sido la primera que se aplicó al QAP en 1984 [12]. Después de esto, en [77] se presentó un nuevo equilibrio de componentes. En [15] se introdujo un concepto de *temperatura óptima* que ha dado buenos resultados. Otros enfoques basados en Simulated Annealing más recientes fueron propuestos en [51, 56].

Búsqueda Local Iterada

La primer aplicación de Búsqueda Local Iterada al QAP es reportada en [67].

2.3.2. Algoritmos exactos aplicados al QAP

Mientras que se ha hecho gran progreso en generar "buenas soluciones" a instancias de QAP grandes y difíciles, éste no ha sido el caso para encontrar soluciones exactas. La historia de resolver instancias de QAP exactamente se centra en los conocidos problemas de Nugent. En 1968, Nugent, Vollmann y Ruml [59] propusieron un conjunto de instancias del problema de tamaño 5, 6, 7, 8, 12, 15, 20 y 30, de acuerdo a su dificultad. En el momento de publicación de su artículo, era un logro encontrar la solución óptima a la instancia de tamaño 8. Enumerar todas las 8! (ocho factorial) posibles asignaciones tomaba 3374 segundos en una computadora GE 265 (poco más de ocho horas). No fue sino hasta 1975, cuando Burkard resolvió la instancia Nugent 8 en 0.426 segundos en una máquina CDC Cyber76, que se logró un progreso notable en los métodos de resolución exactos. Para esto, Burkard usó el límite inferior de Gilmore-Lawler (GL) [32] en un algoritmo de *ramificación y acotación*.

En los años 70 y 80, sólo se podía esperar resolver instancias difíciles para $n < 16$. En 1978, Burkard y Stratmann resolvieron en forma óptima la instancias Nugent 12 en 29.32 segundos con una máquina CDC Cyber76. Ellos usaron un algoritmo de ramificación y acotación con perturbación; de nuevo utilizando el límite inferior GL. En 1980, Burkard y Derigs reportaron que eran los primeros en resolver la instancia Nugent 15 nuevamente con un algoritmo ramificación y acotación, esto lo hicieron en 2947.32 segundos en una CDC Cyber76. Luego de 29 años, desde su formulación, Clausen y Perregaard pudieron resolver exactamente la dificil instancia de Nugent 20. Para esto, usaron un algoritmo de ramificación y acotación paralelo en una máquina MEIKO con 16 procesadores Intel i860. El tiempo de reloj para resolverla fue de 57963 segundos y requirió la evaluación de 360148026 nodos. En un solo procesador, el tiempo habría sido de 811440 segundos (poco más de 9 dias). De nuevo, contaron con el límite de GL que había sido desarrollado décadas atrás. Se ha hecho mucho progreso desde entonces, aunque todavía, la resolución exacta de la instancia más difícil de Nugent (Nugent 30) requiere recursos computacionales que normalmente no se encuentran en las universidades de hoy.

En los últimos años se han desarrollado nuevos límites inferiores los cuales han permitido avances considerables en la resolución exacta de instancias [28]. En la siguiente sección se realiza una breve introducción a la temática de los límites inferiores.

Límites inferiores

El estudio de los límites inferiores es muy importante para el desarrollo de algoritmos para resolver problemas de programación matemática y problemas de optimización combinatoria. Generalmente, los métodos exactos emplean enumeración implícita, en un intento de garantizar la solución óptima y, al mismo tiempo, evitar la enumeración total

de las soluciones factibles. El rendimiento de estos métodos depende de la calidad y la eficiencia computacional de los límites inferiores.

Los límites inferiores son herramientas fundamentales para las técnicas de *ramificación y acotación* y para la evaluación de la calidad de las soluciones obtenidas a partir de algunos algoritmos heurísticos. Se puede medir la calidad de un límite inferior por la diferencia entre su valor y la solución óptima. Los límites inferiores buenos deberían estar cerca del óptimo. Límites inferiores son útiles en los métodos exactos, sólo cuando se puede calcular rápidamente. Cuando se utiliza en los métodos heurísticos, es más importante su calidad que la velocidad de calcularlo.

2.4. Descripción de instancias de QAP a utilizar

En esta sección se describen dos conjuntos de instancias ampliamente utilizadas en la literatura.

2.4.1. Instancias de QAPLIB

QAPLIB[3] es una librería para investigación sobre el QAP [10], que contiene un gran número de instancias de benchmark. Actualmente tiene más de 130 instancias que han sido utilizadas en investigaciones previas y una parte de estas deriva de aplicaciones reales como disposición de hospitales, diseño de teclados, etc. Además, QAPLIB contiene una serie de otros recursos como enlaces a la literatura, algunos códigos fuente y enlaces para investigación relacionada a QAP, incluyendo una lista de personas con intereses de investigación en el QAP.

Es conocido que el tipo particular de instancia de QAP tiene una influencia considerable en la performance de los métodos heurísticos. De acuerdo a [71] las instancias de QAPLIB pueden ser clasificadas en cuatro clases.

- Instancias no estructuradas, generadas aleatoriamente. Instancias con entradas de la matriz de distancia y flujo generadas aleatoriamente de acuerdo a una distribución uniforme. Estas instancias están entre las más difíciles de resolver exactamente. A pesar de esto, muchos métodos de búsqueda iterativa encuentran rápidamente soluciones dentro de 1–2 % respecto de la mejor solución conocida.
- Matriz de distancia basada en grilla. En esta clase de instancias la matriz de distancia viene de una grilla de $n_1 \times n_2$ y las distancias son definidas como la distancia Manhattan entre puntos de la grilla. Estas instancias tienen múltiples óptimos globales (al menos 4 en el caso que $n_1 \neq n_2$ y al menos 8 en el caso que $n_1 = n_2$) debido a la definición de las matrices de distancia.

[3]Disponible on-line en: `http://www.seas.upenn.edu/qaplib/`

- Instancias de la vida real. Instancias de esta clase son instancias de la "vida real" de aplicaciones prácticas del QAP. Los problemas de la vida real tienen en común que las matrices de flujos tienen muchas entradas en cero y las entradas restantes son claramente distribuidas no uniformemente. Las entradas de la matriz exhiben una estructura clara y es principalmente en este aspecto que las instancias de la vida real difieren de las instancias generadas aleatoriamente descritas en el primer punto.
- Ya que las instancias en QAPLIB son principalmente de tamaño pequeño, un nuevo tipo de problema generados aleatoriamente ha sido propuesto. Estas instancias son generadas de tal manera que las entradas de la matriz representan una distribución encontrada en problemas de la vida real.

Se debe remarcar algunas observaciones adicionales acerca de las instancias de QAPLIB. Algunas de las instancias se han generado aleatoriamente (con soluciones óptimas conocidas) con un generador propuesto en [45]. Esto permite evaluar la calidad de las soluciones de las metaheurísticas con respecto a óptimos conocidos también en instancias grandes, que de otro modo no sería posible debido al pobre comportamiento de los algoritmos exactos. Sin embargo, estas instancias suelen ser más fáciles que otras instancias de QAPLIB.

Para muchas instancias grandes se conocen soluciones "pseudo-óptimas" [71]. Las soluciones "pseudo-óptimas" son aquellas para las cuales muchos algoritmos han encontrado las mismas mejores soluciones conocidas y se puede asumir que estas son en realidad las óptimas. En segundo lugar, se tiene que mencionar que para muchos tipos de instancias de QAP, la diferencia [4] entre la peor solución y la solución óptima se vuelve arbitrariamente pequeña con una probabilidad tendiente a uno cuando el tamaño del problema tiende a infinito [13].

2.4.2. Instancias de Stützle y Fernandes

Este conjunto de instancias fue propuesto en [68] y ha sido ampliamente utilizado por *Metaheuristics Network*[5]. El conjunto de instancias fue generado de manera que:

- las características de las instancias varien de forma sistemática.
- sean lo suficientemente grande como para permitir estudios sistemáticos sobre la dependencia del rendimiento de las metaheurísticas de acuerdo a las características de las instancias.

Generación de instancias

La generación de instancias se hace de manera tal que puedan ser generadas instancias con características sintácticas muy diferentes relativas a:

[4] La diferencia generalmente está expresada en término de la cantidad de componentes de soluciones que no tienen en común dos soluciones

[5] `http://www.metaheuristics.net/`

- los valores de dominancia de flujo de las entradas de la matriz (el valor de dominancia es el coeficiente de variación de las entradas de la matriz);
- y a la dispersión de las entradas de la matriz (que mide el porcentaje de entradas de la matriz que son cero).

Estas dos medidas influyen fuertemente en las características de las instancias de QAP y se supone que también tienen una fuerte influencia en la performance de las metaheurísticas. Como segundo parámetro importante se consideran dos formas distintas de generar la matriz de distancias, lo que permite generar matrices de distancia con estructuras diferentes. En particular, para la matriz de distancia se aplican dos formas diferentes de generarla:

- Distancias Euclidianas: En un primer enfoque la matriz corresponde a la distancia Euclidiana entre n puntos en el plano. Estos puntos son generados de tal manera que se ubiquen en varios grupos de diferente tamaño.
- Distancias Manhattan: En un segundo enfoque, la matriz de distancia corresponde a la distancia de a pares entre todos los nodos en una grilla de $A \times B$, donde la distancia se calcula como la distancia Manhattan entre los puntos de una grilla.

Las razones de la elección de las diferentes matrices de distancia son las siguientes. Instancias basadas en las distancias euclidianas tendrán muy probablemente una única solución óptima y por los distintos ajustes de parámetros para la generación de matrices de distancia, puede ser imitada la agrupación de ubicaciones encontrada a menudo en las instancias de QAP de la vida real. Instancias con una matriz de distancias basada en la distancia Manhattan de puntos en una grilla tendrán, debido a simetrías en la matriz, al menos cuatro soluciones óptimas que se encuentran a una máxima distancia posible unas de otras. Por otra parte, las matrices de distancia muestran una agrupación mucho más baja. Adicionalmente, algunas instancias de la vida real tienen distancias que derivan de puntos en una grilla.

También se consideran diferentes formas de generar la *matriz de flujo*. Para todas las instancias se tiene una matriz de flujo asimétrica y con una diagonal nula. Las instancias generadas tienen dominancia de flujo y dispersión bastante variada. Para la generación de la matriz de flujo se consideran dos casos distintos:

- las entradas de flujo son generados aleatoriamente.
- las entradas de flujo son estructuradas en el sentido de que existan grupos de objetos que tengan una interacción significativa.

2.5. Resumen

En este capítulo se presentaron definiciones asociadas al problema a resolver (QAP), incluyendo dos formulaciones, un breve análisis de la complejidad desde el punto de vista

computacional, asi como también algunos ejemplos de problemas formulados como QAP. También se muestran aplicaciones de distintas metaheurísticas aplicadas al problema. Se finaliza el capítulo describiendo las principales librerías de instancias del problema en cuestión.

Capítulo 3

La metaheurística de Optimización basada en Colonias de Hormigas

En este capítulo se presenta la metaheurística de Optimización basada en Colonias de Hormigas (ACO, por sus siglas en inglés, Ant Colony Optimization). ACO fue propuesta por Dorigo y colegas [22, 18, 23] como un método para resolver problemas de optimización combinatorios (POCs) duros. Los algoritmos de optimización basados en colonias de hormigas son parte de la rama *inteligencia colectiva*, este es, el campo de investigación que estudia algoritmos inspirados en la observación del comportamiento de *enjambres* (swarms). Los algoritmos de inteligencia colectiva están compuestos de individuos simples que cooperan a través de la auto-organización, es decir sin ninguna forma de control central sobre los miembros del enjambre.

3.1. De las colonias de hormigas a las hormigas artificiales

La metaheurística ACO está inspirada en la observación del comportamiento de hormigas reales. En esta sección, se presentan observaciones hechas en experimentos con hormigas, y luego se muestra como estos experimentos inspiraron el diseño de la metaheurística.

3.1.1. Las colonias de hormigas

Las hormigas son insectos sociales, los cuales viven en colonias y cuyo comportamiento está dirigido más hacia la supervivencia de la colonia como un todo que al de una simple componente individual de la colonia. Los insectos sociales han capturado la atención de muchos científicos por el gran nivel de estructuración que alcanzan sus colonias, especialmente cuando se lo compara con la simplicidad relativa de los componentes de la colonia.

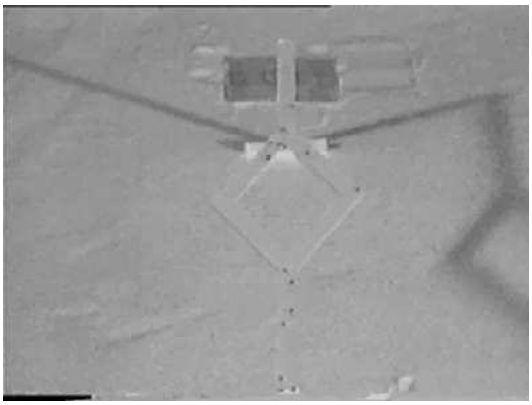


Figura 3.1: Imagen del experimento real llevado a cabo por Deneubourg *et al.*

Uno de los primeros científicos en investigar el comportamiento social de insectos, fue el entomólogo francés Pierre-Paul Grassé. Entre los años 40 y 50, se dedicó a observar el comportamiento de termitas – en particular, las especies *Bellicositermes natalensis* y *Cubitermes*. Descubrió (ver [37]) que estos insectos son capaces de reaccionar a lo que llamó "estímulos significantes", los cuales son señales que activan una reacción codificada genéticamente. Observó (ver [38]) que los efectos de estas reacciones pueden actuar como nuevos estímulos significantes, tanto para el insecto que las produjo como para los otros insectos en la colonia. Grassé usó el término *estigmergía* [1] (ver [38]) para describir este particular tipo de comunicación indirecta en la cual "los miembros de la colonia son estimulados por el desempeño que han logrado".

Ejemplos de estigmergía, también pueden ser observados en colonias de hormigas. Muchas especies de hormigas, mientras caminan desde el nido hacia una fuente de comida y viceversa, depositan en el suelo una sustancia llamada *feromona*. Otras hormigas pueden oler esta sustancia, y su presencia influencia la elección de su camino [2], esto es, tienden a seguir concentraciones de feromona fuertes. La feromona depositada en el suelo forma un *rastro de feromona*, el cual permite a las hormigas encontrar buenas fuentes de comida que han sido previamente identificadas por otras hormigas.

Algunos científicos investigaron experimentalmente este comportamiento de depositar feromona y seguir rastros de feromona para entenderlo mejor y poder cuantificarlo. Deneubourg *et al.* [16] diseñaron un experimento llamado "experimento del puente binario". Ellos usaron hormigas *Linepithema humile* (hormigas originarias de Argentina). El nido de las hormigas fue conectado a una fuente de comida por dos puentes de igual longitud, ver Figura 3.1. Las hormigas podían elegir libremente que brazo del puente usar cuando buscaban comida y la traían de vuelta al nido. Su comportamiento fue observado por un período de tiempo.

En este experimento, inicialmente no hay feromona en ninguno de los dos puentes. Las hormigas empiezan explorando los alrededores del nido y eventualmente cruzan uno de los puentes, alcanzando la fuente de comida. Al caminar hacia la fuente de comida y de vuelta, las hormigas depositan feromona en el puente que usaron. Inicialmente, cada

[1] La palabra estigmergía deriva de las palabras griegas *stigma* (marca, senñal) y *ergon* (trabajo, acción).

[2] Cabe recordar que muchas especies de hormigas son ciegas.

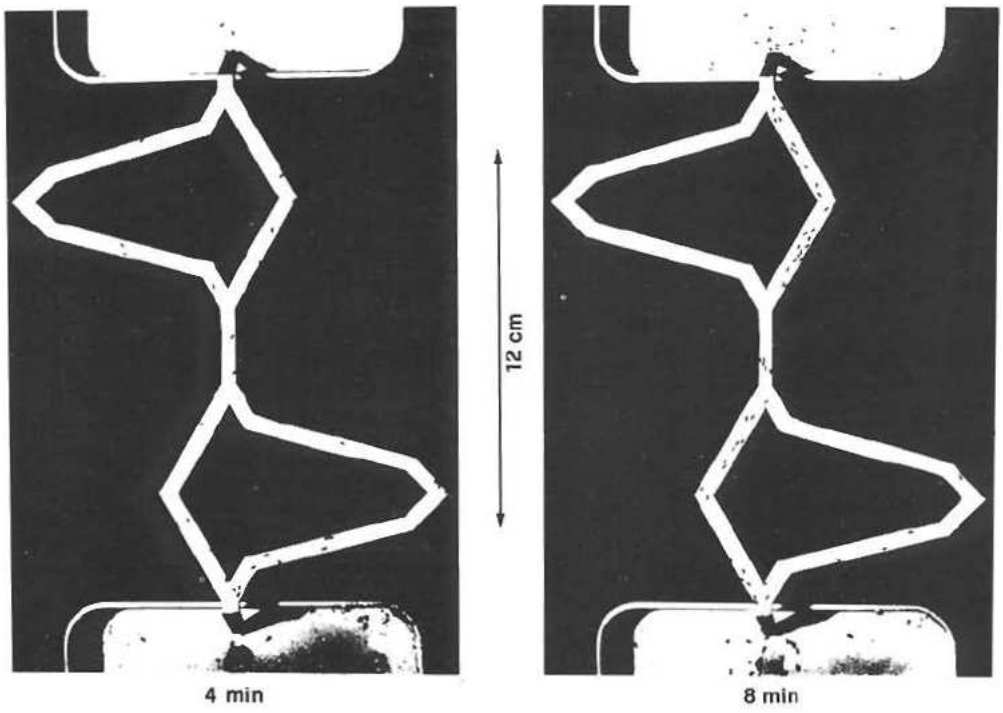


Figura 3.2: Imagen del experimento real llevado a cabo por Goss *et al.*

hormiga elige aleatoriamente uno de los puentes. Sin embargo, debido a cambios aleatorios, después de algún tiempo habrá más feromona depositada en uno de los puentes que en el otro. Debido a que las hormigas tienden a preferir (en probabilidad) seguir un rastro de feromona más fuerte, el puente que tiene más feromona atraerá más hormigas. Esto a su vez hace que el rastro de feromona crezca más, hasta que finalmente la colonia de hormigas converge hacia el uso del mismo puente[3].

Este comportamiento a nivel de la colonia, basada en autocatálisis (retroalimentación positiva), puede ser explotado por las hormigas para encontrar el camino más corto entre una fuente de comida y su nido. Esto se demostró en otro experimento dirigido por Goss *et al.* [36] en el que los dos puentes no eran de la misma longitud: uno era significantemente más largo que el otro, ver Figura 3.2. En este caso, las fluctuaciones aleatorias sobre la opción inicial de un puente eran más reducidas: debido a que aquellas hormigas que elegían por casualidad el puente más corto también eran las primeras en alcanzar el nido, y al volver al nido, escogían el puente más corto con mayor probabilidad cuando tenía un rastro de feromona mas fuerte. Por consiguiente, las hormigas – gracias al mecanismo de seguir y depositar feromona – convergían rápidamente al uso del puente más corto.

3.1.2. Hormigas artificiales

Estimulado por los interesantes resultados, descritos en la sección anterior, Goss *et al.* [36] desarrollaron un modelo para explicar el comportamiento observado en el

[3]Deneubourg *et al.* [16] realizaron varios experimentos, y los resultados mostraron que cada uno de los dos puentes era usado en aproximadamente el 50 % de los casos.

experimento del puente de dos brazos. Asumiendo que después de t unidades de tiempo desde el comienzo del experimento, m_1 hormigas han usado el primer brazo del puente y m_2 el segundo, la probabilidad p_1 para la $(m+1)$-ésima hormiga de elegir el primer brazo del puente puede estar dada por

$$p_{1(m+1)} = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \tag{3.1}$$

donde los parámetros k y h son necesarios para ajustar el modelo a los datos experimentales. La probabilidad que la misma $(m+1)$-ésima hormiga elija el segundo brazo es $p_{2(m+1)} = 1 - p_{1(m+1)}$. Simulaciones de Monte Carlo, ejecutadas para probar si el modelo corresponde a los datos reales (ver [62]), mostraron muy buen ajuste para valores de $k \approx 20$ y $h \approx 2$.

Este modelo básico, el cual explica el comportamiento de las hormigas, puede ser usado como una inspiración para diseñar hormigas artificiales que resuelvan problemas de optimización definidos de una manera similar. En el ejemplo descrito anteriormente del *comportamiento forrajero de las hormigas*, la comunicación por estigmergía ocurre a través de la feromona que las hormigas depositan en el suelo. Análogamente, las hormigas artificiales pueden simular la acción de depositar feromona modificando apropiadamente variables de feromona asociadas con estados del problema que visitan mientras construyen soluciones al problema de optimización. También, de acuerdo al modelo de comunicación por estigmergía, las hormigas artificiales sólo tendrían acceso en forma local a estas variables de feromona.

Por lo tanto, las características principales de estigmergía mencionadas en la sección anterior pueden ser extendidas a agentes artificiales:

- asociando variables de estado con diferentes estados de problema; y
- dándole a los agentes sólo acceso local a estas variables.

Otro aspecto importante de la conducta forrajera de las hormigas reales que puede ser explotado por hormigas artificiales es la relación entre el mecanismo autocatalítico[4] y la *evaluación implícita* de soluciones. Por evaluación implícita de soluciones, se entiende al hecho de que caminos más cortos (los cuales corresponden a soluciones de menor costo en el caso de las hormigas artificiales) son completados antes que los más largos, y por lo tanto reciben refuerzos de feromona más rápido. La evaluación implícita de soluciones junto con el autocatálisis puede ser muy efectivo: mientras más corto es el camino, más pronto se deposita la feromona, y más hormigas usan el camino más corto. Si se usa apropiadamente, puede ser un mecanismo poderoso en algoritmos de optimización basados en población (por ejemplo, en los algoritmos evolutivos [40, 30] el autocatálisis es implementado a través del mecanismo de selección/reproduccíon).

[4]El concepto de autocatálisis define al tipo de procesos propias de sistemas que son capaces de producir algunos de los elementos que son necesarios para su recurrencia continuada en el tiempo.

La estigmergía, junto con la evaluación implícita de soluciones y el comportamiento autocatalítico, ha dado lugar a los algoritomos ACO. La idea básica de los algoritmos ACO sigue muy estrechamente la inspiración biológica. Por lo tanto, hay muchas similitudes, entre las hormigas reales y las hormigas artificiales. Ambas colonias de hormigas (reales y artificiales) están compuestas de una población de individuos que trabajan juntos para lograr un cierto objetivo. Una colonia es una población de agentes simples, independientes y asíncronos que cooperan para encontrar una buena *solución* al problema a resolver. En el caso de las hormigas reales, el problema es encontrar la comida; mientras que en el caso de las hormigas artificiales, es el de encontrar una buena solución a un problema de optimización dado. Una sola hormiga (real o artificial) es capaz de encontrar una solución a su problema, pero sólo la cooperación entre muchos individuos a través de estigmergía les permite encontrar *buenas* soluciones.

En el caso de las hormigas reales, ellas depositan y reaccionan a una sustancia química llamada *feromona*. Las hormigas reales simplemente la depositan en la tierra mientras caminan. Las hormigas artificiales viven en un mundo *virtual*, de aquí que ellas sólo modifican valores numéricos (llamados por analogía, *feromona artificial*) asociados con diferentes estados de problema. Una secuencia de valores de feromona asociados con estados del problema se llama *rastro de feromona artificial*. En los algoritmos ACO, los rastros de feromona artificial son los únicos medios de comunicación entre las hormigas. Un mecanismo análogo a la evaporación física de feromona en colonias de hormigas reales permite a las hormigas artificiales *olvidar* la historia y enfocarse en nuevas direcciones de búsqueda prometedoras.

Así como las hormigas reales, las hormigas artificiales crean sus soluciones secuencialmente moviéndose de un estado del problema a otro. Las hormigas reales simplemente caminan, escogiendo una dirección basada en las concentraciones de feromona local y en una política de decisión aleatoria. Las hormigas artificiales también crean soluciones paso a paso, moviéndose a través de estados del problema disponibles y tomando decisiones aleatorias a cada paso.

Sin embargo hay algunos diferencias importantes entre las hormigas reales y artificiales:

- Las hormigas artificiales viven en un mundo discreto–se mueven secuencialmente a través de un conjunto finito de estados del problema.
- La actualización de feromona (es decir, el depósito y evaporación de feromona) no se realiza exactamente de la misma manera por las hormigas artificiales como por las reales. A veces la actualización de feromona es hecha sólo por algunas de las hormigas artificiales, y a menudo *sólo después* de que una solución se ha construido.
- Algunas implementaciones de hormigas artificiales usan mecanismos adicionales que no existen en el caso de las hormigas reales. Los ejemplos incluyen mirar-hacia-adelante, búsqueda local y vuelta atrás, entre otros.

3.2. La metaheurística de Optimización basada en Colonias de Hormigas

Los algoritmos de Optimización basados en Colonias de Hormigas han sido formalizados en una metaheurística de optimización de combinatoria por Dorigo *et al.* [19, 20] y desde entonces han sido usados para resolver muchos Problemas de Optimización Combinatoria (POC). Dado un POC, el primer paso para la aplicación de un algoritmo ACO, para resolverlo consiste en definir un modelo adecuado. Luego este modelo es usado para definir el componente central de los algoritmos ACO: el modelo de feromona.

El modelo de un POC es usado para derivar el modelo de feromona utilizado por los algoritmos ACO. Primero, una variable de decisión instanciada $X_i = v_i^j$ (es decir, una variable X_i con un valor v_i^j asignado de su dominio $\mathbf{D}_i$) se llama *componente de una solución* y se denota por c_{ij}. El conjunto de todas las posibles componentes de soluciones es denotado por $\mathbf{C}$. Un parámetro de rastro de feromona T_{ij} es asociado con cada componente c_{ij}. El conjunto de todos los parámetros de rastros de feromona es denotado por $\mathbf{T}$. El valor de un parámetro de rastro de feromona T_{ij} es denotado por τ_{ij} (y es llamado valor de feromona). Este valor de feromona es usado y actualizado por el algoritmo ACO durante la búsqueda, y permite modelar la distribución de probabilidad de diferentes componentes de una solución.

En los algoritmos ACO, las hormigas artificiales construyen una solución para un POC atravesando un grafo llamado *grafo de construcción*, $G_C(\mathbf{V}, \mathbf{E})$. El grafo de construcción (totalmente conectado) consiste de un conjunto de vértices $\mathbf{V}$ y un conjunto de arcos $\mathbf{E}$. El conjunto de componentes $\mathbf{C}$ puede ser asociado con el conjunto de vértices $\mathbf{V}$, o con el conjunto de arcos $\mathbf{E}$. Las hormigas se mueven de un vértice a otro vértice a lo largo de los arcos del grafo, construyendo incrementalmente una *solución parcial*. Además, las hormigas depositan una cierta cantidad de feromona sobre las componentes, es decir, en los vértices o en los arcos que atraviesan. La cantidad de feromona $\Delta\tau$ depositada puede depender de la calidad de la solución encontrada. Las hormigas siguientes utilizan la información de la feromona como una guía hacia regiones más prometedoras del espacio de búsqueda.

La metaheurística ACO se muestra en el Algoritmo 1. El mismo consiste en una fase de inicialización y una iteración sobre tres componentes. Esta iteración consiste de la construcción de soluciones por todas las hormigas, la mejora de la solución (fase optativa) con el uso de un algoritmo de búsqueda local, y la actualización de feromona. A continuación, se explican estos tres componentes en detalle.

ConstruirSolucionesporHormigas Un conjunto de m hormigas artificiales construyen soluciones a partir de elementos de un conjunto finito de componentes de soluciones disponibles $\mathbf{C} = \{c_{ij}\}$, $i = 1, \ldots, n$, $j = 1, \ldots, |\mathbf{D}_i|$. La construcción de una solución empieza con una solución parcial vacía $s^p = \emptyset$. Luego, en cada paso de la

Algoritmo 1 Metaheurística ACO

```
Establecer parámetros, inicializar rastros de feromona
while (no se cumpla condición de terminación) do
  ConstruirSolucionesporHormigas
  AplicarBúsquedaLocal { Opcional }
  ActualizarFeromona
end while
```

construcción, la solución parcial actual s^p es extendida agregando una componente de solución factible del conjunto de vecinos factibles $\mathbf{N}(s^p) \subseteq \mathbf{C}$. El proceso de construcción de soluciones puede ser considerado como un camino en el grafo de construcción $G_C(\mathbf{V}, \mathbf{E})$. Los caminos permitidos en G_C están definidos implícitamente por el mecanismo de construcción de soluciones que define el conjunto $\mathbf{N}(s^p)$ con respecto a una solución parcial s^p.

La elección de una componente de solución de $\mathbf{N}(s^p)$ se hace probabilísticamente en cada paso de la construcción. Las reglas exactas para la elección probabilística de componentes de soluciones varían entre diferentes variantes de algoritmos ACO. La regla más conocida es de Ant System [23]:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{il} \in \mathbf{N}(s^p)} \tau_{il}^{\alpha} \cdot \eta(c_{il})^{\beta}}, \qquad \forall c_{ij} \in \mathbf{N}(s^p), \tag{3.2}$$

donde τ_{ij} es el valor de feromona asociado con la componente c_{ij}, y $\eta(\cdot)$ es una función que asigna a cada paso de la construcción un valor heurístico para cada componente de solución factible $c_{ij} \in \mathbf{N}(s^p)$. Los valores que son retornados por esta función son llamados normalmente *información heurística*. Además, α y β son parámetros positivos cuyos valores determinan la importancia relativa de la feromona y de la información heurística respectivamente. La ecuación 3.2 es una generalización de la ecuación 3.1. Como se puede apreciar, la formalización de los algoritmos ACO sigue estrechamente la inspiración biológica.

AplicarBúsquedaLocal Una vez que las soluciones han sido construidas, y antes de actualizar la feromona, a veces pueden ser necesarias algunas acciones opcionales. Estas acciones son llamadas *acciones auxiliares* [5], y pueden ser usadas para implementar acciones centralizadas y/o específicas del problema, las cuales no pueden ser realizadas por simples hormigas. La acción auxiliar más usada consiste en la aplicación de búsqueda local a la solución construida: las soluciones localmente óptimas son usadas para decidir como actualizar feromona.

ActualizarFeromona El objetivo de la actualización de feromona es incrementar los valores de feromona asociados con soluciones buenas o prometedoras, y decrementar

[5] El término original en inglés es *daemon actions*, concepto que caracteriza a procesos que se ejecutan cuando se cumplen determinadas condiciones.

aquéllos valores que están asociados con malas soluciones. Normalmente, esto se logra:

- decrementando todos los valores de feromona a través de la *evaporación de feromona*.
- aumentando los niveles de feromona asociados con un conjunto de buenas soluciones $\mathbf{S}_{upd}$ elegido

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\sum_{s\in\mathbf{S}_{upd}|c_{ij}\in s} F(s) \tag{3.3}$$

donde $\mathbf{S}_{upd}$ es el conjunto de soluciones que son usadas para la actualización, $\rho \in (0,1]$ es un parámetro llamada *factor de evaporación*, y $F : \mathbf{S} \rightarrow \mathbb{R}_0^+$ es una función tal que $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathbf{S}$. La función $F(\cdot)$ es llamada *función de fitness*.

La evaporación de feromona es necesaria para evitar una convergencia demasiado rápida del algoritmo. Implementa una forma útil de *olvidarse* y favorecer la exploración de nuevas áreas en el espacio de búsqueda. Diferentes algoritmos ACO, por ejemplo, $\mathcal{MAX}-\mathcal{MIN}$ Ant System [69] o Ant Colony System [21] difieren en la forma en que actualizan feromona.

Instanciaciones de la regla de actualización presentada en la ecuación 3.3 son obtenidas por diferentes especificaciones de $\mathbf{S}_{upd}$, la cual en muchos casos es un subconjunto de $\mathbf{S}_{iter} \cup \{s_{bs}\}$ donde $\mathbf{S}_{iter}$ es el conjunto de soluciones que fueron construídas en la iteración actual, y s_{bs} es la solución *best-so-far* (mejor hasta ahora), esto es, la mejor solución encontrada desde la primer iteración del algoritmo. Un ejemplo muy conocido es la regla de actualización de Ant System, donde

$$\mathbf{S}_{upd} \leftarrow \mathbf{S}_{iter} \tag{3.4}$$

Un ejemplo de regla de actualización de feromona que es muy utilizada en la práctica es la regla de actualización IB (donde IB significa *la mejor de la iteración*)

$$\mathbf{S}_{upd} \leftarrow \arg\operatorname*{máx}_{s\in\mathbf{S}_{iter}} F(s) \tag{3.5}$$

La regla de actualización IB introduce un sesgo mucho más fuerte hacia las soluciones buenas encontradas respecto de la regla de Ant System. Aunque esto aumenta la velocidad con qué se encuentran buenas soluciones, también aumenta la probabilidad de convergencia prematura. Un sesgo aun más fuerte es introducido por la regla de actualización BS, donde BS se refiere al uso de la solución mejor-hasta-ahora s_{bs}. En este caso, $\mathbf{S}_{upd}$ toma el valor de $\{s_{bs}\}$. En la práctica, los algoritmos ACO que usan variantes de las reglas de actualización IB ó BS y que adicionalmente incluyen mecanismos para evitar convergencia prematura obtienen mejores resultados que aquellos que usan la regla de actualización de Ant System.

3.2.1. Ejemplo: El Problema del Viajante de Comercio

El Problema del Viajante de Comercio (TSP, por sus siglas en inglés, Traveling Salesman Problem) juega un rol importante en los algoritmos de optimización basados en colonias de hormigas porque fue el primer problema en ser atacado con estos métodos. El TSP fue elegido por varias razones: (i) es un problema para el cual la metáfora de las colonias de hormigas se adapta fácilmente, (ii) es uno de los problemas $\mathcal{NP}$-duro más estudiados en el campo de optimización combinatoria, y (iii) es muy fácil de explicar.

Una definición general del TSP es la siguiente. Considere un conjunto de nodos N, representando ciudades, y un conjunto de arcos E conectando completamente los nodos de N. Sea d_{ij} la longitud del arco $(i, j) \in E$, esto es la distancia entre las ciudades i y j, con $i, j \in N$. El TSP es el problema de encontrar un circuito Hamiltoniano de longitud mínima sobre el grafo $G = (N, E)$, donde un circuito Hamiltoniano del grafo G es un tour cerrado que visita una y solo una vez todos los nodos $n = |N|$ de G, y su longitud está dada por la suma de las longitudes de los arcos del cual está compuesto el tour.

La aplicación de ACO al TSP es simple. Los movimientos entre ciudades son las componentes de soluciones, esto es, el movimiento desde la ciudad i a la ciudad j es el componente de solución $c_{ij} \equiv c_{ji}$. El grafo de construcción $G_C(\mathbf{V}, \mathbf{E})$ se define asociando el conjunto de ciudades con el conjunto de vértices $\mathbf{V}$ del grafo. Ya que, en principio, es posible moverse desde cualquier ciudad a cualquier ciudad, el grafo de construcción es completamente conectado y el número de vértices es igual al número de ciudades definidas por la instancia del problema. Asimismo, las longitudes de los arcos entre los vértices son proporcionales a las distancias entre las ciudades representadas por esos vértices. La feromona se asocia con el conjunto de arcos $\mathbf{E}$ del grafo.

Es importante recalcar que también es posible asociar el conjunto de componentes de solución del TSP (o de cualquier otro problema de optimización combinatoria, como se verá en el próximo capítulo para el QAP) con el conjunto de vértices $\mathbf{V}$ en lugar del conjunto de arcos $\mathbf{E}$ del grafo de construcción $G_C(\mathbf{V}, \mathbf{E})$. Para el TSP, significaría asociar los movimientos entre ciudades con el conjunto de vértices $\mathbf{V}$, y las ciudades con el conjunto de arcos $\mathbf{E}$. Cuando se usa este enfoque, el proceso de construcción de soluciones de las hormigas tiene que ser modificado apropiadamente: las hormigas tienen que moverse de vértice a vértice del grafo de construcción eligiendo de este modo las *conexiones entre las ciudades*.

3.3. Algoritmos basados en la metaheurística ACO

En la literatura se han propuesto diversos algoritmos que siguen la metaheurística ACO. Entre los principales algoritmos ACO disponibles para problemas de optimización combinatoria $\mathcal{NP}$-duro, se encuentran:

- Ant System

- $\mathcal{MAX} - \mathcal{MIN}$ Ant System
- Ant Colony System

En las secciones subsiguientes, se presenta una breve descripción de cada uno de estos algoritmos aplicados al Problema del Viajante de Comercio, anteriormente descrito.

3.3.1. Ant System

El Sistema de Hormigas (AS, por sus siglas en inglés, Ant System) [22, 23] fue el primer algoritmo ACO propuesto en la literatura. Inicialmente, se presentaron tres variantes distintas: *ant-density*, *ant-quantity* y *ant-cycle*, que se diferenciaban en la manera en que actualizaban los rastros de feromona. En los dos primeros, las hormigas depositaban feromona mientras que construían sus soluciones (esto es, aplicaban una actualización *on-line* paso a paso de feromona). Mientras, que en *ant-cycle*, la actualización de feromona se llevaba a cabo una vez que todas las hormigas habían construido una solución y la cantidad de feromona depositada por cada hormiga era establecida en función de la calidad de la solución. Esta última variante era la que obtenía mejores resultados y es por lo tanto la que se conoce como AS en la literatura (y en el resto de este trabajo).

Construcción de la solución

En AS, m hormigas concurrentemente construyen una solución (en este caso un tour para el problema TSP). Inicialmente, las hormigas son ubicadas en ciudades elegidas aleatoriamente). En cada paso de la construcción, la hormiga k aplica una regla de elección de acción probabilística (llamada regla *proporcional aleatoria*) para decidir que ciudad visitar a continuación. En particular, la probabilidad con la cual la hormiga k, estando en la ciudad i, elige una ciudad j para ir es:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} & \text{si } j \in \mathcal{N}_i^k \\ 0 & \text{en otro caso} \end{cases} \tag{3.6}$$

donde $\eta_{ij} = 1/d_{ij}$ es un valor heurístico disponible *a priori*, α, β son dos parámetros que establecen la importancia relativa de los rastros de feromona y de la información heurística respectivamente, y $\mathcal{N}_i^k$ es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo i (esto es, el conjunto de ciudades que la hormiga k aún no ha visitado). Cada hormiga k mantiene una memoria $\mathcal{M}^k$ la cual contiene las ciudades visitadas, y en el orden en que fueron visitadas. Esta memoria se usa para determinar la vecindad factible $\mathcal{N}_i^k$ en cada paso de la construcción.

Respecto a los parámetros α y β, su función es la siguiente: si $\alpha = 0$, aquellos nodos con una preferencia heurística mejor tienen una mayor probabilidad de ser escogidos,

haciendo el algoritmo muy similar a un algoritmo voraz probabilístico clásico (con múltiples puntos de partida en caso de que las hormigas estén situadas en nodos distintos al comienzo de cada iteración). Sin embargo, si $\beta = 0$, sólo se tienen en cuenta los rastros de feromona para guiar el proceso constructivo, lo que puede causar un rápido estancamiento, esto es, una situación en la que los rastros de feromona asociados a una solución son ligeramente superiores que el resto, provocando por tanto que las hormigas siempre construyan las mismas soluciones, normalmente óptimos locales. Por tanto es necesario establecer un balance adecuado entre la información heurística y los rastros de feromona.

Actualización de rastros de feromona

Como se ha dicho, el depósito de feromona se realiza una vez que todas las hormigas han acabado de construir sus soluciones. Primero, los rastros de feromona asociados a cada arco se evaporan reduciendo todos los rastros de feromona en un factor constante:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \tag{3.7}$$

donde $0 < \rho \leq 1$ es la tasa de evaporación. El parámetro ρ es usado para evitar una acumulación ilimitada de los rastros de feromona y permite al algoritmo "olvidar" malas decisiones tomadas previamente. De hecho, si un arco no es elegido por las hormigas su valor de feromona asociado decrece exponencialmente con el número de iteraciones. Después de la evaporación, todas las hormigas depositan feromona sobre los arcos que han elegido en su tour:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{3.8}$$

donde $\Delta\tau_{ij}^{k}$ es la cantidad de feromona que la hormiga k deposita en los arcos que ha visitado. Se define como:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{si la hormiga } k \text{ usó el arco } (i,j) \text{ en su tour} \\ 0 & \text{en otro caso} \end{cases} \tag{3.9}$$

donde Q es una constante y L_k es la longitud del tour construido por la k-ésima hormiga. Esto significa, que cuanto mejor sea el tour construido por la hormiga, mayor es la cantidad de feromona que reciben los arcos de ese tour.

3.3.2. $\mathcal{MAX} - \mathcal{MIN}$ Ant System

El Sistema de Hormigas $\mathcal{MAX} - \mathcal{MIN}$ ($\mathcal{MM}$AS, por sus siglas en inglés, $\mathcal{MAX} - \mathcal{MIN}$ Ant System) [69] es un algoritmo mejorado que toma como base las ideas de AS. $\mathcal{MM}$AS fue propuesto por Stützle y Hoos, ellos introdujeron un número de cambios de los cuales los más importantes son:

- solo la mejor hormiga puede actualizar los rastros de feromona (esta puede ser la hormiga que generó el mejor tour de la iteración actual o la hormiga que generó el mejor tour desde el inicio del algoritmo).
- limita el rango posible de los valores de rastros de feromona al intervalo $[\tau_{min}, \tau_{max}]$. Este mecanismo fue ideado para contrarrestar el efecto que produce permitir que solo la mejor hormiga actualice la feromona.
- los rastros de feromona son inicializados con el valor del límite superior de rastro de feromona, el cual, junto con un pequeño factor de evaporación de feromona, incrementa la exploración de los tour al comienzo de la búsqueda.
- los rastros de feromona son reinicializados cada vez que el sistema conduce a la situación de estancamiento o cuando no ha sido generado ningún tour mejor durante un cierto número de iteraciones.

Actualización de rastros de feromona

Después de que todas las hormigas han construido una solución, se actualizan los rastros de feromona aplicando evaporación de acuerdo a la ecuación 3.7 (la misma que es utilizada por el Sistema de Hormigas), seguida por el depósito de nueva feromona de acuerdo a:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \tag{3.10}$$

donde $\Delta\tau_{ij}^{best}$ es la cantidad de feromona que la *mejor* hormiga deposita en los arcos que ha visitado. Se define como:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & \text{si la mejor hormiga usó el arco } (i,j) \text{ en su tour} \\ 0 & \text{en otro caso} \end{cases} \tag{3.11}$$

donde L_{best} es la longitud del tour de la mejor hormiga. La hormiga a la que se le permite añadir feromona puede ser la que generó la mejor solución de la iteración actual, en cuyo caso $L_{best} = L_{ib}$ donde L_{ib} es la longitud del mejor tour encontrado en la iteración actual; o la que generó la mejor solución desde el inicio del algoritmo, en cuyo caso $L_{best} = L_{bs}$ donde L_{bs} es la longitud del mejor tour encontrado desde el inicio del algoritmo. Los resultados experimentales demuestran que el mejor rendimiento se obtiene incrementando gradualmente la frecuencia de elegir la mejor global para la actualización de feromona respecto de la mejor de la iteración.

Límites de rastros de feromona

En $\mathcal{MM}$AS, los límites inferior y superior (τ_{min} y τ_{max}) sobre los posibles valores de feromona en cualquier arco son impuestos para evitar estancamiento de la búsqueda, al darle a cada conexión existente una probabilidad, aunque bastante pequeña, de ser

escogida. En particular, los límites impuestos tienen el efecto de limitar indirectamente la probabilidad p_{ij} de seleccionar una ciudad j cuando una hormiga está en una ciudad i al intervalo $[p_{min}, p_{max}]$, con $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. Sólo si una hormiga tiene una sola opción posible para la próxima ciudad, entonces $p_{min} = p_{max} = 1$. Resultados experimentales sugieren que el límite de rastro inferior usado en el algoritmo es más importante que el límite superior, ya que la cantidad máxima de rastro posible sobre los arcos está limitada en largas ejecuciones debido a la evaporación de la feromona.

Inicialización y Reinicialización de rastros de feromona

Al comienzo del algoritmo, los rastros de feromona son inicializados con un valor que es una estimación del límite superior del rastro de feromona. En combinación con un valor pequeño para el factor de evaporación, esto produce que las diferencias relativas entre los rastros de feromona no sean muy diferentes, de manera que al comienzo el algoritmo sea muy explorativo.

$\mathcal{MM}$AS introduce un medio de exploración adicional, para los caminos que tienen una baja probabilidad de ser elegidos, a través de la reinicialización de los rastros de feromona. Este mecanismo se activa cuando el algoritmo se acerca a una situación de estancamiento (medido por estadísticas) o si no se han encontrado mejores tour por un cierto número de iteraciones.

3.3.3. Ant Colony System

El Sistema de Colonia de Hormigas (ACS, por sus siglas en inglés, Ant Colony System) [21] es uno de los primeros sucesores de AS que introduce tres modificaciones importantes con respecto a dicho algoritmo:

- usa una regla de elección de acción más agresiva que AS.
- la evaporación de feromona y el depósito de feromona se aplica sólo a los arcos del mejor tour global.
- cada vez que una hormiga usa un arco (i, j) para moverse de la ciudad i a la ciudad j, disminuye cierta cantidad de feromona de ese arco.

Construcción de la solución

El ACS usa una regla de transición distinta, denominada regla *proporcional pseudo-aleatoria*. Sea k una hormiga situada en el nodo i, el siguiente nodo j se elige aleatoriamente mediante la siguiente distribución de probabilidad:

$$j = \begin{cases} \arg\max_{l \in \mathcal{N}_i^k} \{\tau_{il}[\eta_{il}]^\beta\} & \text{si } q \leq q_0 \\ J & \text{en otro caso} \end{cases} \tag{3.12}$$

donde q es una variable aleatoria uniformemente distribuida en $[0, 1]$, $q_0 \in [0, 1]$ es un parámetro, y J es una variable aleatoria seleccionada de acuerdo a la distribución de probabilidad dada por la ecuación (3.6).

Como puede observarse, la regla tiene una doble intención: cuando $q \leq q_0$, explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona. Sin embargo, si $q > q_0$ se aplica una exploración controlada, tal como se hacía en AS. En resumen, la regla establece un compromiso entre la exploración de nuevas conexiones y la explotación de la información disponible en ese momento.

Actualización de rastros de feromona global

En ACS sólo se permite que una hormiga (la hormiga best-so-far) agregue feromona después de cada iteración. Por lo tanto, la actualización se implementa mediante la siguiente ecuación:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{bs} \tag{3.13}$$

donde $\Delta\tau_{ij}^{bs} = \frac{1}{C^{bs}}$. Es importante recalcar que, en ACS, la actualización de rastros de feromona (tanto la evaporación como el hecho de depositar nueva feromona) sólo se aplica a los arcos de la mejor solución, no a todos los arcos como en el AS.

Actualización de rastros de feromona local

Además de la regla de actualización de feromona que se realiza al final de cada iteración, en el ACS, las hormigas usan una regla de actualización de feromona local (también referida como *online*) que aplican cada vez que atraviesan un arco (i, j) durante la construcción de la solución:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \tag{3.14}$$

donde $\varphi \in (0, 1)$ es el coeficiente de disminución de feromona, y τ_0 es el valor inicial de los rastros de feromona.

Como puede verse, la regla de actualización *online* paso a paso incluye tanto la evaporación de feromona como el depósito de la misma. Ya que la cantidad de feromona depositada es muy pequeña, la aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan. Así, esto lleva a una técnica de exploración adicional del algoritmo ACS ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el resto de hormigas que las recorren en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

3.4. Resumen

En este capítulo se presentó una introducción a la metaheurística ACO, la cual está inspirada en el comportamiento de forrajero de las hormigas reales. El componente central de la metaheurística ACO es el modelo de feromona basado en el modelo subyacente del problema a ser resuelto. La idea básica de ACO deja muchas opciones y elecciones para el diseñador del algoritmo. Se han propuesto muchos algoritmos basados en la metaheurísitca ACO, siendo los de mayor éxito los algoritmos $\mathcal{MM}$AS y ACS. La Optimización basada en Colonia de Hormigas es un metaheurística relativamente joven, en comparación con otras, tales como computación evolutiva, búsqueda tabú, o simulated annealing. Sin embargo, ha demostrado ser muy eficiente y flexible. Los algoritmos de optimización basados en colonia de hormigas son actualmente estado-del-arte para la resolución de muchos problemas de optimización combinatoria. Para un panorama detallado de ACO, incluyendo la teoría y las aplicaciones, el lector interesado debería referirse al libro *Ant Colony Optimization* escrito por el autor de la metaheurística [24].

Capítulo 4

Algoritmos ACO aplicados al QAP

En este capítulo se presentan las principales variantes de algoritmos ACO que han sido aplicadas al QAP, desde que se inventó la mencionada metaheurística. También se presentan las principales aplicaciones de algoritmos ACO que utilizan el enfoque de memoria explícita. Por último se presenta detalladamente la variante de algoritmo ACO propuesta en este trabajo final.

4.1. Respecto de la aplicación de ACO al QAP

Como se mencionó en el capítulo anterior, para aplicar la metaheurística ACO a un problema de optimización combinatoria, es conveniente representar el problema mediante el grafo de construcción $G_C(\mathbf{V}, \mathbf{E})$, donde $\mathbf{V}$ es el conjunto de vértices y $\mathbf{E}$ es el conjunto de arcos. Para representar el QAP como este grafo, el conjunto de vértices $\mathbf{V}$ está compuesto por los elementos y las ubicaciones de la descripción del problema, y el conjunto de arcos $\mathbf{E}$ conecta a los vértices, es decir, hay arcos desde elementos a ubicaciones y arcos desde ubicaciones a elementos. La construcción de una asignación de elementos a ubicaciones (o de ubicaciones a elementos) puede ser interpretada como un movimiento de la hormiga sobre la representación del grafo de construcción del QAP, guiado por la información del rastro de feromona (asociado a los arcos) y posiblemente por información heurística disponible localmente. El movimiento de la hormiga adicionalmente tiene que obedecer ciertas restricciones para asegurar que se genere una solución factible. En el caso del QAP, tal solución factible asigna cada elemento a exactamente una ubicación y cada ubicación tiene asignada exactamente un elemento. Por lo tanto, una solución factible ϕ para el QAP consiste de n duplas (i, j) de elementos y ubicaciones.

Para la aplicación práctica de la metaheurística ACO al QAP es conveniente usar una dirección de asignación fija, por ejemplo, asignar elementos a ubicaciones (o ubicaciones a elementos). Entonces, los elementos son asignados por las hormigas en algún orden,

el cual se denomina *orden de asignación*, a las ubicaciones. Una hormiga iterativamente aplica los siguientes dos pasos hasta que ha construido una solución completa. En un primer paso la hormiga elige un elemento, y en un segundo paso el elemento elegido es asignado a alguna ubicación. Los rastros de feromona y la información heurística pueden ser asociados con ambos pasos. Con respecto al primer paso, los rastros de feromona y la información heurística pueden ser usados para aprender un orden de asignación apropiado (obviamente, el orden de asignación puede influenciar el rendimiento del algoritmo). En cuanto al segundo paso, el rastro de feromona τ_{ij} y la información heurística η_{ij} asociadas con la dupla entre el elemento i y la ubicación j determina la deseabilidad de poner el elemento i en la ubicación j. Después de que todas las hormigas han generado una solución factible y posiblemente la hayan mejorado aplicando búsqueda local, se les permite depositar feromona, teniendo en cuenta la calidad de la solución.

4.2. Enfoques previos de algoritmos ACO para QAP

Desde la primera aplicación de Ant System al QAP, varios algoritmos ACO mejorados para este problema han sido propuestos por distintos autores. A pesar de las diferencias entre estos algoritmos, comparten al menos dos aspectos comunes importantes.

Uno de los aspectos se refiere a la construcción de soluciones. Todas los algoritmos propuestos para el QAP, asocian rastros de feromona τ_{ij} únicamente a asignaciones de la forma $\phi_i = j$, por lo tanto, τ_{ij} se puede interpretar como la deseabilidad de asignar el elemento i a la ubicación j. En cuanto al orden de asignación, todos los algoritmos propuestos suponen que es fijo durante la ejecución del algoritmo [49, 47, 48] o se selecciona aleatoriamente en base a una distribución uniforme [66, 73, 69].

El segundo aspecto es que todos los algoritmos propuestos mejoran las soluciones construidas por las hormigas usando un algoritmo de búsqueda local. Por lo tanto, estos algoritmos son algoritmos híbridos que combinan la construcción de soluciones que realizan las hormigas (artificiales) con algoritmos de búsqueda local. Esta combinación puede ser muy interesante. Los algoritmos constructivos a menudo resultan en una baja calidad de soluciones en comparación con los algoritmos de búsqueda local. Por otro lado, se ha observado que repetir búsqueda local a partir de soluciones iniciales generadas aleatoriamente resulta (para la mayoría de los problemas) en una diferencia considerable respecto de la solución óptima [41] . Sin embargo, la experiencia ha demostrado que la combinación de una heurística constructiva (probabilística y adaptable) con búsqueda local puede obtener soluciones significativamente mejores [21, 69]. Los algoritmos ACO son estos algoritmos heurísticos de construcción adaptativa, en el sentido de que una colonia de hormigas modifica la representación de una solución asignando mayor rastro de feromona a las conexiones (en este caso, a asignaciones de elementos a ubicaciones), que están en las mejores soluciones. Durante la construcción de soluciones las hormigas seleccionan con mayor preferencia asignaciones que tienen mayor cantidad de feromona

y combinando estas asignaciones generan soluciones iniciales prometedoras para el algoritmo de búsqueda local. Una ventaja adicional de la utilización de algoritmos ACO es que, mediante la generación de buenas soluciones iniciales, el algoritmo de búsqueda local necesita un número mucho menor de iteraciones para alcanzar un óptimo local. De esta manera, para un límite de tiempo determinado se pueden ejecutar muchos más "búsquedas locales" que si se comenzará a partir de soluciones generadas aleatoriamente.

En las siguientes secciones se presentan algunos algoritmos ACO para el QAP en mayor detalle.

4.2.1. Ant System

Ant System es el primer algoritmo ACO que ha sido aplicado al QAP (AS-QAP) [49, 23]. En AS-QAP el orden de asignación es determinado por un pre-ordenamiento de los elementos, como se explica más abajo. Luego, en cada paso un elemento i es asignado probabilísticamente a alguna ubicación j, prefiriendo ubicaciones con rastro de feromona τ_{ij} alto e información herística η_{ij} prometedora.

Información heurística La información heurística sobre la potencial bondad de una asignación es determinado en AS-QAP como sigue. Dos vectores d y f son calculados en los cuales la i-esima componente representa respectivamente la suma de las distancias desde la ubicación i a todas las otras ubicaciones y la suma de los flujos desde el elemento i a todos los otros elementos. Mientras más bajo sea el valor de d_i, la distancia potencial de la ubicación i, más central es la ubicación considerada; y mientras más alto sea el valor de f_i, el flujo potencial del elemento i, más importante es el elemento. Luego se calcula una matriz de acoplamiento $\mathbf{E} = f \cdot d^T$, donde el $e_{ij} = f_i \cdot d_j$. Entonces, la deseabilidad heurística de asignar el elemento i a la ubicación j esta dada por $\eta_{ij} = 1/e_{ij}$. La motivación por usar este tipo de información heurística es que, intuitivamente, las soluciones buenas pondrán elementos con alto flujo potencial en las ubicaciones con baja distancia potencial.

Construcción de soluciones Una solución se construye como sigue. En AS-QAP los elementos son ordenados en orden decreciente de flujo potencial. En cada paso de la construcción una hormiga k asigna el siguiente elemento aún no asignado i a una ubicación libre j con una probabilidad dada por:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \qquad \text{si } j \in \mathcal{N}_i^k \tag{4.1}$$

donde el $\tau_{ij}(t)$ es el rastro de feromona en la iteración t, α y β son parámetros que determinan la influencia relativa de la cantidad de feromona y la información heurística, y $\mathcal{N}_i^k$ es la vecindad factible del nodo i, es decir, sólo aquellas ubicaciones que estan todavía libres (note que $\sum_{j \in \mathcal{N}_i^k} p_{ij}(t) = 1$). Estos pasos se repiten hasta que se construye una asignación completa.

Actualización de feromona La actualización del rastro de feromona aplicado a todas las duplas es realizado de acuerdo a la siguiente ecuación:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{4.2}$$

donde ρ, con $0 < \rho < 1$, es la persistencia del rastro de feromona, de modo que $(1-\rho)$ representa la evaporación. El parámetro ρ es usado para evitar acumulación ilimitada de rastros de feromona y permitir al algoritmo olvidar malas decisiones tomadas previamente. $\Delta\tau_{ij}^{k}$ es la cantidad de feromona que la hormiga k pone en la dupla (i, j); y esta dado por

$$\Delta\tau_{ij} = \begin{cases} Q/J_{\psi}^{k} & \text{si el elemento } i \text{ es asignado a la ubicación } j \text{ en la solución de la hormiga } k \\ 0 & \text{en otro caso} \end{cases} \tag{4.3}$$

donde ψ^k es la solución de la hormiga k, J_{ψ}^{k} es el valor de la función objetivo, y Q es la cantidad de feromona depositada por una hormiga.

Una primer mejora de AS-QAP es presentada en [26] y nos referiremos a ella como AS2-QAP. AS2-QAP difiere principalmente en la forma en que la información heurística es calculada y en una manera diferente de calcular la probabilidad de asignar elementos a ubicaciones. Aún, desde una mejora adicional sobre AS2-QAP, llamada ANTS-QAP y presentada en la próxima sección, obtiene mejores resultados, no presentamos los detalles de AS2-QAP.

4.2.2. ANTS

En [47] se presenta un algoritmo ACO, llamado ANTS[1], de aquí en más referido como ANTS-QAP. El algoritmo ANTS introduce varias modificaciones con respecto al AS. La modificación más significativa es el uso de límites inferiores sobre el costo de una solución, en la finalización de una solución parcial para computar dinámicamente valores heurísticos η_{ij} variables. Otras modificaciones adicionales son el uso de una regla de elección de acción diferente y una regla de actualización de rastros de feromona modificada.

Uso de límites inferiores En ANTS-QAP se usan límites inferiores (ver sección 2.3.2) respecto del costo de una solución parcial para dar información heurística sobre que tan atractiva es agregar la asignación (i, j) específica. Esto se logra agregando tentativamente la asignación a la solución parcial actual y estimando el costo de una solución

[1]De acuerdo a [47] el término ANTS deriva del hecho de que el algoritmo propuesto puede ser interpretado como un Approximate Nondeterministic Tree Search (Arbol de Búsqueda No determinístico Aproximado) ya que comparte varios elementos con un procedimiento aproximado de Ramificación y Acotación.

completa que contenga esa asignación por medio del límite inferior. Esta estimación luego es usada para influenciar las decisiones probabilisticas tomadas por la hormiga durante la construcción de la solución: mientras más baja es la estimación del límite, más atractiva es agregar la asignación específica. El uso de límites inferiores para la información heurística presenta varias ventajas, como la eliminación de posibles movimientos si la estimación del costo es mayor que la mejor solución encontrada hasta ahora. Como el límite inferior es calculado en cada paso de la construcción de una hormiga, estos deberían ser computados eficientemente.

En ANTS-QAP se computa el límite inferior de Gilmore-Lawler (GLB) [32] al comienzo del algoritmo. Junto con el computo del límite inferior uno obtiene los valores de las variables duales correspondientes a las restricciones cuando se formula el QAP como un problema de programación entera. Una desventaja del límite GLB es una complejidad computacional relativamente alta, la cual es $\mathcal{O}(n^3)$. Para disminuir el esfuerzo computacional asociado con las computaciones del límite inferior, en ANTS-QAP un límite inferior más débil que GLB es propuesto, llamado LBD. Este límite inferior tiene la ventaja de tener una complejidad computacional que es $\mathcal{O}(n)$. Los resultados computacionales presentados en [25] muestran que usando este límite inferior débil es suficiente para guiar la construcción de soluciones de las hormigas.

Construcción de soluciones En ANTS-QAP es dado un pre-oredenamiento de las ubicaciones por los valores de las variables duales las cuales son obtenidas cuando se computa el límite GLB para una instancia. Dada una ubicación j, la hormiga k decide asignar el elemento i a esta ubicación con la siguiente probabilidad [2]:

$$p_{ij}^k(t) = \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha) \cdot \eta_{ij}}{\sum_{l \in \mathcal{N}_j^k} \alpha \cdot \tau_{lj}(t) + (1-\alpha) \cdot \eta_{lj}} \qquad \text{si } i \in \mathcal{N}_j^k \tag{4.4}$$

Una ventaja de la ecuación 4.4 comparada con la ecuación 4.1, es que se elimina un parámetro. Adicionalmente, son aplicadas operaciones más simples las cuales son más rápidas de computar, como multiplicaciones en lugar de exponenciación. $\mathcal{N}_j^k$ es la vecindad factible; es definida por los elementos que no han sido asginados a ninguna ubicación. Aquí $\sum_{i \in \mathcal{N}_j^k} p_{ij}(t) = 1$. De hecho, en esta formulación el pre-ordenamiento es realizado respecto de las ubicaciones en lugar de los elementos. Obviamente, también es posible usar un pre-ordenamiento de ubicaciones como en AS-QAP.

Actualización de feromona La actualización de feromona en ANTS-QAP no usa evaporación de feromona, esto es, tenemos $\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k$. Aquí, $\Delta\tau_{ij}^k$ es definido como:

[2] La misma regla de elección de acción es usada por el algoritmo AS2-QAP.

$$\Delta\tau_{ij} = \begin{cases} \tau_0 \cdot (1 - \frac{J_\psi^k - LB}{J_{avg} - LB}) & \text{si el elemento } i \text{ es asignado a la ubicación } j \text{ en la solución de la hormiga } k \\ 0 & \text{en otro caso} \end{cases} \tag{4.5}$$

donde J_{avg} es el movimiento promedio de las últimas l soluciones provistas por las hormigas y LB es el valor del límite GLB el cual es calculado al comienzo del algoritmo. Si una solución de una hormiga es peor que el promedio de movimiento actual, la cantidad de rastro de feromona de las duplas construidas por las hormigas es decrementada. El efecto adicional de usar la ecuación 4.5 es una escala dinámica de las diferencias de la función objetivo lo cual puede ser muy ventajoso si en etapas posteriores de la búsqueda la diferencia absoluta entre las calidades de la solución de la hormiga se vuelven menores.

4.2.3. $\mathcal{MAX} - \mathcal{MIN}$ Ant System

$\mathcal{MAX} - \mathcal{MIN}$ Ant System fue propuesto primero para el TSP y luego para el QAP [69], por lo cual es necesario adaptar algunos componentes del algoritmo para su correcta aplicación.

Construcción de soluciones En $\mathcal{MM}$AS-QAP, en cada paso de la construcción, la hormiga k primero elige aleatoriamente un elemento i entre aquellos aún no asignados, y luego lo ubica en una ubicación libre j con una probabilidad dada por:

$$p_{ij}^k(t) = \frac{\tau_{ij}(t)}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}(t)} \qquad \text{si } j \in \mathcal{N}_i^k \tag{4.6}$$

Es notable que $\mathcal{MM}$AS-QAP no hace uso de información heurística. Como se mostró en la aplicación de $\mathcal{MM}$AS al problema TSP [69], la información heurística no es necesaria para alcanzar soluciones de alta calidad cuando las soluciones son mejoradas con un algoritmo de búsqueda local. No usar información heurística además elimina un parámetro y, por lo tanto, reduce el esfuerzo para configurarlos.

Actualización de rastros de feromona Después de que todas las hormigas han construido una solución, los rastros de feromonas son actualizados de acuerdo a:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{best} \tag{4.7}$$

Aquí, $\Delta\tau_{ij}^{best}$ es definido como:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/J_\psi^{best} & \text{si el elemento } j \text{ es asignado a la ubicación } i \text{ en la solución } \psi^{best} \\ 0 & \text{en otro caso} \end{cases} \tag{4.8}$$

donde J_{ψ}^{best} es el valor de la función objetivo de ψ^{best}. ψ^{best} puede ser la solución *iteration-best* ψ^{ib}, o la mejor solución durante la ejecución del algoritmo, la solución *global-best* ψ^{gb}. Por lo tanto, si en las mejores soluciones los elementos son puestos frecuentemente en ubicaciones específicas, estas asignaciones tendrán una cantidad mayor de feromona. Una elección de buen sentido en el uso de ψ^{ib} o ψ^{gb} para la actualización del rastro de feromona puede ayudar fácilmente a mejorar el rendimiento del algoritmo. En general, se obtienen mejores resultados si la frecuencia de elegir ψ^{gb} incrementa durante la ejecución del algoritmo. Adicionalmente uno tiene que asegurar que la cantidad de rastro de feromona respete los límites. Si después de la actualización de feromona tenemos $\tau_{ij} > \tau_{max}$, hacemos $\tau_{ij} = \tau_{max}$; análogamente, si $\tau_{ij} < \tau_{min}$, asignamos $\tau_{ij} = \tau_{min}$.

Características de diversificación adicional Para la aplicación de $\mathcal{MM}$AS al QAP se usa una técnica adicional para incrementar la diversificación de la búsqueda. En particular, sólo si el progreso de la búsqueda es muy pequeño, los rastros de feromona son reinicializados con el valor τ_{max}.

4.2.4. FANT

En [72] se propone otro algoritmo ACO, llamado Fast Ant System (Sistema de Hormigas Rápido). En el algoritmo FANT las soluciones son construidas de la misma forma que en $\mathcal{MM}$AS-QAP usando la regla de elección de acción dada en la ecuación 4.6, por lo que tampoco utiliza información heurística. Además se diferencia de otros algoritmos ACO, en dos aspectos: el número de hormigas que usa y el manejo de los rastros de feromona.

Número de hormigas FANT sólo utiliza una hormiga, es decir, no usa una población. El uso de una sola hormiga permite al algoritmo encontrar buenas soluciones rápido. Esto se considera una configuración de parámetros específica más que una característica importante del algoritmo.

Actualización de feromona Al igual que el algoritmo ANTS-QAP, FANT no utiliza evaporación de rastros de feromona. Inicialmente, todos los rastros de feromona son inicializados en uno y después de cada iteración se agrega feromona de acuerdo a la ecuación:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + r \cdot \Delta\tau_{ij} + r^* \cdot \Delta\tau_{ij}^{gb} \tag{4.9}$$

donde $\Delta\tau_{ij} = 1/J_{\psi}$ para cada par (i, j) que es parte de la solución generada en la iteración actual, y $\Delta\tau_{ij}^{gb} = 1/J_{\psi}^{gb}$ para cada par de la mejor solución global ψ^{gb}. r y r^* son dos parámetros. El valor de r puede ser modificado deurante la ejecución del algoritmo (inicialmente es tiene el valor uno), mientras que el valor de r^* se mantiene

fijo. Los dos parámetros determinan el refuerzo relativo provisto por la solución actual y por la mejor solución global.

En dos ocaciones los rastros de feromona se actualizan de una manera diferente a la descrita en la regla anterior.

- Si la mejor solución global ha sido mejorada, el parámetro r se inicializa en uno y los rastros de feromona son reinicializados con un valor igual a uno (con esto se logra intensificación de la búsqueda en torno a la mejor solución global).
- Si la hormiga construye una solución que sea igual a ψ^{gb}, r es incrementado en uno y de nuevo los rastros de feromona son reinicializados con el valor de r (diversificación de la búsqueda decrementando el peso relativo de ψ^{gb}).

4.3. Algoritmos ACO con uso de memoria explícita

En esta sección se presentan los principales algoritmos ACO, que siguen la temática abordada en este trabajo final: el uso de memoria explícita. Estos algoritmos tiene en común que en todos los casos, las hormigas completan la construcción de soluciones que toman como base de la memoria explícita.

4.3.1. ACO con Memoria Externa

En [1, 2] se introduce un enfoque basado en memoria externa a algoritmos ACO; donde la población incluye segmentos de solución de longitud variable [1] ó secuencias de permutaciones parciales de longitud variable [2], formados a partir de las mejores soluciones de iteraciones previas. Inicialmente, la memoria esta vacía y un algoritmo ACO clásico se ejecuta por un pequeño número de iteraciones para inicializar la librería de soluciones parciales. Dependiendo de si se utiliza segmentos o secuencias de soluciones, tenemos:

- Cada segmento de solución almacenado es asociado con el valor de la función objetivo de la solución de la cual se extrajo el segmento y que será usado como medida para la selección de segmentos, y en la actualización de la memoria. No hay una distribución particular de hormigas en el espacio del problema y el punto de inicio para una hormiga es el punto inicial del segmento con el que comienza.
- Cada secuencia almacenada esta asociada con un "tiempo de vida"[3] y con el valor de la función objetivo de la solución de la cual se extrajo la secuencia que son usados como medida para recuperar y actualizar soluciones parciales dentro de la memoria.

Para construir una solución, una hormiga particular recupera un segmento de la memoria basado en su valor objetivo ó una secuencia de permutación parcial y luego completa el resto de la solución siguiendo el proceso usual. Las soluciones construidas también son usadas para actualizar la memoria.

[3]Indicado almacenando la iteración en la cual se creó esta secuencia.

Descripción del algoritmo

En el algoritmo, hay m hormigas que construyen las soluciones. Se mantiene una memoria externa (formada por segmentos de soluciones de longitud variable o por secuencias de soluciones parciales de longitud variable) la cual inicialmente esta vacía, y se realizan un número de iteraciones determinado de un algoritmo ACO clásico para completar la memoria. Después de cada iteracion, las mejores k soluciones son consideradas y dependiendo de como esta formada la memoria externa:

- segmentos de soluciones de longitud variable. Se "cortan" segmentos de soluciones (aleatoriamente posicionados) de las mejores k soluciones, un segmento por solución, y se almacenan en la memoria.
- secuencias de soluciones de longitud variable. Se selecciona un número de componentes de soluciones (en posiciones aleatorias) de las mejores k soluciones, una secuencia por solución y se almacenan en la memoria.

En ambos casos el valor de fitness asociado al segmento o a la secuencia de soluciones, está dado por el valor de la función objetivo de la solución de la cual se generó el segmento o la secuencia. El tamaño de la memoria M es fijo, por lo que no varía durante la ejecución del algoritmo, y se repiten iteraciones de un algoritmo ACO tradicional hasta que se completa la memoria.

Después de la fase inicial, el algoritmo ACO trabaja en conjunto con la memoria externa implementada: no hay una asignación de hormigas sobre el espacio del problema y, para construir una solución, cada hormiga selecciona un segmento o secuencia de solución de la memoria usando una estrategia de selección por torneo. Luego cada hormiga construye el resto de la solución. En la selección de segmentos o secuencias de solución de la memoria, cada hormiga realiza un torneo entre Q segmentos o secuencias de solución seleccionados aleatoriamente y el/la mejor es seleccionado/a como la semilla para comenzar la construcción. El procedimiento de construcción de solución es el mismo que se utiliza en el algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Después de que todas las hormigas han completado su solución, se realiza la actualización de feromona de la misma forma que es hecha en algoritmos ACO tradicionales.

En el procedimiento de actualización de la memoria, las soluciones construidas por las hormigas son ordenadas en orden ascendente y las mejores k son consideras como soluciones candidatas de las cuales nuevos segmentos o secuencias de permutaciones parciales serán insertadas en la memoria.

- Un segmento posicionado aleatoriamente y de longitud aleatoria se forma a partir de cada una de las mejores soluciones, y hay dos estrategias para actualizar los elementos de la memoria. Primero, un segmento nuevo reemplaza el peor de todos los segmentos que tenga un costo asociado mayor que el del segmento nuevo. Si no existe este segmento (porque todos los segmentos tienen costo igual o menor que el segmento nuevo), luego, el segmento nuevo se concatena con el elemento de la memoria que tenga el mayor costo asociado, y los valores repetidos se

quitan del segmento resultante. El propósito de esta operación de concatenación es lograr diversificación a través de la combinación de dos segmentos de solución prometedores.

- Una secuencia de permutación parcial de longitud aleatoria y posicionada aleatoriamente se forma a partir de cada una de las mejores soluciones, y esta reemplaza el peor de los elementos de la memoria que tengan un valor de fitness peor que el de la secuencia extraída o el peor de los elementos de la memoria que tenga un "tiempo de vida" mayor (al más antiguo).

4.3.2. Iterated Ants $\mathcal{MAX} - \mathcal{MIN}$ Ant System

Este algoritmo examina la posibilidad de cambiar la forma en la que se construyen las soluciones respecto de los algoritmos ACO tradicionales [76]. Se plantea comenzar la construcción de soluciones a partir de soluciones parciales que son obtenidas quitando algunas componentes de soluciones de una solución de una hormiga. Esta modificación esta inspirada en gran parte en los métodos Voraces Iterados[4].

Descripción del algoritmo

En los algoritmos ACO, las soluciones candidatas son generadas comenzando cada construcción de una solución desde cero. Sin embargo, otros métodos de búsqueda local estocástica constructivos son conocidos por usar construcción de soluciones repetidas pero comenzando de soluciones parciales – esta es la idea central de los algoritmos *voraces iterados* (VI). Los algoritmos VI comienzan con alguna solución candidata completa y luego iteran en un bucle principal que consiste de tres procedimientos; primero, un procedimiento *Destruir* elimina de una solución completa s un número de componentes de solución, resultando en una solución candidata parcial s_p. Comenzando desde s_p, se reconstruye una solución completa s' con un procedimiento *Construir* y finalmente, un procedimiento *CriterioAceptación* decide si la próxima iteración se continúa desde s o desde s'. Además, es simple incluir un procedimiento de búsqueda local que mejore una solución candidata una vez que ha sido completada por el procedimiento *Construir*.

Las *Hormigas Iteradas* (Iterated Ants) aplican la idea central subyacente de algoritmos VI a los algoritmos ACO. Esto puede ser hecho de una manera más bien simple considerando a cada hormiga como que implementa un algoritmo VI. Esto es, una hormiga individual sigue los pasos de un algoritmo VI y, por lo tanto, la construcción de la solución en al algoritmo híbrido VI-ACO comienza de una solución parcial que es obtenida de eliminar componentes de soluciones de una solución candidata completa de una

[4]Los algoritmos Voraces Iterados iteran sobre algoritmos de construcción de la siguiente manera. Dada alguna solución inicial s, primero se quitan algunas componentes de solución, resultando en una solución candidata parcial s_p. A partir de s_p se reconstruye una solución candidata completa s' mediante una heurística de construcción voraz. Luego un criterio de aceptación decide a partir de cual de las dos soluciones (s o s') continúa la próxima iteración.

hormiga. La construcción de soluciones por las hormigas sigue los mismos pasos que los algoritmos ACO tradicionales, esto es, las componentes de soluciones son agregadas tomando en cuenta rastros de feromona e información heurística posiblemente.

Iterated Ants $\mathcal{MAX} - \mathcal{MIN}$ Ant System (ia$\mathcal{MM}$AS) es el algoritmo resultante de integrar la idea de los algoritmos VI en el algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System, y se ha probado para resolver el QAP. El algoritmo ia$\mathcal{MM}$AS considera distintas opciones para el procedimiento *Destruir*, donde varía la elección de como las componentes de soluciones son eliminadas de soluciones completas, cuantas componentes de solución son eliminadas, y el tipo de actualización de feromona elegido en el algoritmo (para el caso del QAP, eliminar una componente de una solución corresponde a deshacer una asignación de un elemento a una ubicación). Para la eliminación de componentes de soluciones, se tiene tres variantes.

- `rand`: las componentes de soluciones a ser removidas son elegidas aleatoriamente de acuerdo a una distribución uniforme.
- `prob`: la probabilidad de quitar una componente de solución es proporcional al rastro de feromona τ_{ij} correspondiente, esto es, mientras mayor sea el rastro de feromona mayor es la probabilidad de quitar una componente de solución.
- `iprob`: la probabilidad de quitar una componente de solución es inversamente proporcional al rastro de feromona τ_{ij} asociado, esto es, mientras menor sea el rastro de feromona mayor es la probabilidad de quitar una componente de solución.

Respecto del número de componentes de soluciones a ser eliminadas considera dos posibilidades.

- `fixed(k)`: Se remueven exactamente k componentes de solución de cada hormiga, donde k es un parámetro.
- `variable`: En este caso, el número de componentes de soluciones a ser quitadas no esta fijado de antemano, pero se mantiene variable similar a como se modifica la intensidad de perturbación en la metaheurística *Variable Neighborhood Search* [39]: si para un valor actual de k la hormiga no obtiene una solución mejorada, se establece $k = k + 1$; en otro caso, se establece con algún valor mínimo.

Finalmente, también considera dos posibilidades para la regla de actualización de feromona.

- `gb`$^+$: La regla de actualización de feromona es muy parecida a la que usa el algoritmo $\mathcal{MM}$AS-QAP original.
- `gb`$^-$: La solución candidata best-so-far y la mejor solución candidata desde la reinicialización de feromona no son tenidas en cuenta cuando se deposita feromona, esto es, sólo la hormiga mejor de la iteración deposita feromona.

4.3.3. Cunning Ant System

El algoritmo Sistema de Hormigas astutas (cAS, del inglés, cunning Ant System) [75] introduce dos esquemas importantes. Uno es un esquema para utilizar soluciones parciales llamado *astuto*. En la construcción de una nueva solución, cAS usa soluciones parciales preexistentes. Con este esquema, se previene un estancamiento prematuro de la búsqueda reduciendo fuertes recompensas a la densidad del rastro. El otro esquema es el uso de un modelo de colonia, que divide las colonias en unidades, el cual tiene una característica de explotación fuerte, mientras mantiene un cierto grado de diversidad entre las unidades.

Descripción del algoritmo

En cAS, se introduce un agente llamado hormiga astuta (*c-ant*, del inglés, cunning ant). La hormiga *c-ant* se diferencia de una hormiga tradicional en su manera de construir una solución. Ella construye una solución "pidiendo prestado" una parte de una solución existente. El resto de la solución lo construye basándose en los rastros de feromona como es habitual. Ya que esta hormiga, en parte, se apropia del trabajo de otras hormigas, se la llama *c-ant* debido a su comportamiento astuto (una solución construida por una hormiga astuta también se representa con la notación *c-ant*). Además, un agente que "ha prestado" una solución parcial a una hormiga astuta, se lo llama hormiga donante (*d-ant*) y la solución parcial donada a la hormiga *c-ant* también es representada con la notación *d-ant*.

Como se mencionó anteriormente el otro esquema que introduce el cAS, es un modelo de colonia que consiste de m unidades. Cada unidad consiste de una sola $ant^*_{k,t}$ ($k = 1, 2, \ldots, m$). En la iteración t en la unidad k, una nueva $c - ant_{k,t+1}$ crea una solución con la hormiga existente en la unidad (es decir, $ant^*_{k,t}$) como la hormiga $d - ant_{k,t}$. Luego, son comparadas la nueva $c - ant_{k,t+1}$ generada y $d - ant_{k,t}$, y la mejor se convierte en la próxima $ant^*_{k,t+1}$ de la unidad. De esta forma, en este modelo de colonia, $ant^*_{k,t}$, el mejor individuo de la unidad k, es siempre preservado. El rastro de feromona es actualizado con $ant^*_{k,t}$ ($k = 1, 2, \ldots, m$) y $\tau_{ij}(t+1)$ se obtiene con la siguiente ecuación:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta^* \tau^k_{ij}(t) \tag{4.10}$$

donde el parámetro $\rho \in [0, 1)$ es la persistencia del rastro y $\Delta^* \tau^k_{ij}(t)$ es la cantidad de feromona que $ant^*_{k,t}$ agrega a los arcos que ha usado en su solución

$$\Delta^* \tau^k_{ij}(t) = \begin{cases} 1/C^*_{k,t} & \text{si } (i,j) \in ant^*_{k,t} \\ \\ 0 & \text{en otro caso} \end{cases} \tag{4.11}$$

y donde $C^*_{k,t}$ es el fitness de $ant^*_{k,t}$.

En el cAS, la actualización de feromona es realizada con m $ant^*_{k,t}$ $(k = 1, 2, \ldots, m)$ por la ecuación (4.10) dentro de los límites $[\tau_{min}, \tau_{max}]$ como en el $\mathcal{MM}$AS [69]. Aquí, τ_{min} y τ_{max} para cAS se define como:

$$\tau_{max}(t) = \frac{1}{1-\rho} \cdot \sum_{k=1}^{m} \frac{1}{C^*_{k,t}} \tag{4.12}$$

$$\tau_{min}(t) = \frac{\tau_{max} \cdot (1 - \sqrt[n]{p_{best}})}{(\frac{n}{2} - 1) \cdot \sqrt[n]{p_{best}}} \tag{4.13}$$

donde p_{best} es un parámetro de control introducido en $\mathcal{MM}$AS [69].

Métodos de muestreo

Una pregunta crucial cuando una *c-ant* crea una nueva solución es cómo determinar qúe parte de la solución la hormiga *c-ant* pedirá prestada a la hormiga *d-ant*. Para asegurar robustez a lo largo de una amplia variedad de problemas, debería ser ventajoso introducir variaciones tanto en la porción y en el número de nodos de la solución parcial que es pedida a la hormiga *d-ant*.

El número de nodos que son construidos en base a los rastros de feromona, se representa por l_s. Luego, l_c, el número de nodos de la solución parcial, los cuales *c-ant* pide a *d-ant*, es $l_c = n - l_s$. Además se introduce un parámetro de control γ el cual define $E(l_s) = n \cdot \gamma$ (donde $E(l_s)$ el promedio de l_s) y se usa la función de densidad de probabilidad $f_s(l)$ definida en [75] como:

$$f_s(l) = \begin{cases} \frac{1-\gamma}{n\gamma}(1 - \frac{l}{n})^{\frac{1-2\gamma}{\gamma}} & \text{para } 0 < \gamma \leq 0.5 \\ \\ \frac{\gamma}{n(1-\gamma)}(\frac{l}{n})^{\frac{2\gamma-1}{1-\gamma}} & \text{para } 0.5 < \gamma < 1 \end{cases} \tag{4.14}$$

En cAS para el TSP, los nodos en posiciones continuas de *d-ant* son copiadas a *c-ant*, porque las soluciones parciales de *d-ant* son representadas por nodos en posiciones continuas. Sin embargo, para el caso del QAP no existe esta restricción y no es necesario que las componentes estén continuas cuando *c-ant* pide a *d-ant* o utiliza los rastros de feromona. De esta forma, para el caso del QAP, cuando se crea una nueva *c-ant* las componentes en las mismas posiciones son copiadas y otras son especificadas con una secuencia aleatoria de posiciones de acuerdo a: El número de nodos l_s a ser especificados es generado por la ecuación (4.14) con un determinado valor para γ. Luego se copia el número de componentes, $l_c = n - l_s$, de la *d-ant* en posiciones aleatorias y se completa el resto de las componentes, l_s, de acuerdo a la ecuación (4.6).

4.4. Enfoque de memoria externa propuesto

Los algoritmos ACO generan soluciones candidatas para un problema de optimización con un mecanismo de construcción donde la elección de la componente de solución a ser agregada en cada paso de la construcción está influenciada probabilísticamente por rastros de feromona e información heurística [24]. En este trabajo, se examina la posibibilidad de alternar la forma en la que se eligen las componentes de solución, introduciendo una memoria externa como mecanismo auxiliar para tomar las decisiones en cada paso de la construcción de una solución. Esta modificación está inspirada en la metaheurística Búsqueda Tabú[5], la cual usa explícitamente la historia de la búsqueda, para escapar de óptimos locales e implementar una estrategia explorativa. Este tipo de enfoque pertenece a una de las tendencias actuales en algoritmos ACO, en los cuales se incorpora una memoria externa como alternativa al mecanismo de elección de componentes de soluciones [1, 2, 75, 76].

El uso de memoria en la búsqueda tabú es tanto explícita como implícita. En el primer caso, se almacenan en memoria soluciones completas, generalmente las mejores soluciones visitadas durante la búsqueda, mientras que en el segundo caso, se almacena información sobre determinados atributos de las soluciones que cambian al pasar de una solución a otra. Aunque, en algunos casos, la memoria explícita es usada para evitar visitar soluciones más de una vez, esta aplicación es limitada dado que es necesario la implementación de estructuras de memoria muy eficientes para evitar requerimientos de memoria excesivos. De cualquier manera, estos dos tipos de memoria son complementarios, puesto que la memoria explícita permite expandir los entornos de búsqueda usados durante un proceso de búsqueda local mediante la inclusión de las mejores soluciones, mientras que la memoria basada en atributos los reduce prohibiendo determinados movimientos. El propósito de clasificar ciertos movimientos como prohibidos es para evitar que se caiga en ciclos durante la búsqueda.

4.4.1. Uso de la memoria

El enfoque abordado agrega un mecanismo con características determinísticas en la construcción de soluciones, en contraposición con la filosofía presente en los algoritmos ACO, los cuales construyen soluciones en forma totalmente probabilística (basada en los rastros de feromona y en la información heurística)[6]. Las estructuras de memo-

[5]El significado de la palabra tabú designa a una conducta, actividad o costumbre prohibida por una sociedad, grupo humano o religión, es decir, es la prohibición de algo natural, de contenido religioso, económico, político, social o cultural por una razón de utilidad social. La búsqueda tabú no se refiere obviamente a ninguna de estas temáticas, sino al hecho de imponer restricciones para guiar un proceso de búsqueda con el objetivo de superar regiones del espacio de búsqueda. Estas restricciones operan de varias formas, como por ejemplo mediante la exclusión directa de alternativas de búsqueda clasificadas como "prohibidas".

[6]El algoritmo Ant Colony System [21] tambiíen introduce características determinísticas en la cosntrucción de soluciones.

ria utilizadas permiten que las hormigas en ciertas ocasiones[7] no tomen decisiones en forma aleatoria sino que elijan componentes de soluciones, de manera determinística, influenciadas por los valores registrados en dicha memoria. Como esta memoria almacena información específica de la historia de la búsqueda desde el comienzo del algoritmo, permite efectivamente enfocarse en regiones del espacio de búsqueda no visitadas (no registradas en la memoria y con valores que reflejen esto) o por el contrario concentrarse en regiones ya visitadas y que sean prometedoras (también registradas en la memoria y con valores que lo reflejen). Estos usos de la memoria reflejan los mecanismos de intensificación y diversificación que, aunque ya estén presentes en los algoritmos ACO, son de utilidad para evitar convergencia prematura y lograr un mayor rendimiento frente a problemas de optimización con características particulares.

Se debe recalcar que aunque el mecanismo en estudio está inspirado en Búsqueda Tabú, existen ciertas diferencias en cuanto al uso de la memoria y también tiene sus diferencias respecto de otros algoritmos ACO que utilizan el enfoque de memoria explícita.

- En la búsqueda tabú, las memorias basadas en lo reciente y en frecuencia se complementan la una a la otra para lograr el balance entre intensificación y diversificación, pero la memoria basada en lo reciente es utilizada como memoria a corto plazo y la memoria basada en frecuencia como memoria a largo plazo; mientras que en el enfoque propuesto no existe esta distinción.
- En los enfoques citados en secciones previas, éstos almacenan en la memoria partes de soluciones (en algunos casos de las mejores soluciones encontradas durante la búsqueda); mientras que en el enfoque propuesto la memoria almacena atributos recolectados durante el historial de la búsqueda, los cuales representan indirectamente patrones interesantes de las soluciones.

Memoria basada en lo reciente

Este tipo de memoria almacena componentes de las soluciones que han cambiado en el pasado reciente. La forma habitual de explotar este tipo de memoria es etiquetando las componentes seleccionadas de soluciones visitadas recientemente. Lo que se intenta lograr es "prohibir" o "incentivar" ciertas elecciones, que impidan o favorezcan a que las hormigas construyan soluciones ya evaluadas en el pasado reciente y con lo cual se logra abarcar una mayor región del espacio de búsqueda o concentrarse en una región particular. Es importante tener en cuenta que en algunas instancias, un buen proceso de búsqueda resultará en volver a visitar una solución encontrada anteriormente. El objetivo más general es continuar estimulando el descubrimiento de nuevas soluciones de alta calidad.

[7]Para esto se utilizan parámetros que indican el nivel de exploración que tiene el algoritmo en determinados instantes de tiempo, como por ejemplo la *entropía*.

Para el caso del QAP, la memoria basada en lo reciente almacena en que iteración del algoritmo, el elemento i ha sido asignado a la ubicación j, $\forall i, j\ 1 \leq i, j \leq n$. Luego esta memoria permite a las hormigas tomar decisiones teniendo en cuenta que elementos son los *más recientemente* asignados a una ubicación particular, por tener asociada un valor cercano a la iteración corriente del algoritmo; o por el contrario tomar decisiones teniendo en cuenta que elementos son los *menos recientemente* asignados a una ubicación particular, por tener asociada un valor lejano a la iteración corriente del algoritmo.

El algoritmo tiene asociado una matriz basada en lo reciente de $n \times n$, denominada `recency`. Donde la posición `recency[i,j]` almacena la iteración más reciente (la última) en la cual el objeto j ha sido asignado a la ubicación i durante la ejecución del algoritmo. Luego, esta matriz se utiliza en el proceso de construcción de soluciones de dos maneras posibles:

- Intensificación, escogiendo aquel objeto que más recientemente fue asignado a la ubicación actual (es decir si la ubicación actual es la i, se escoge aquel objeto j tal que `recency[i,j]` tenga el mayor valor de iteración).
- Diversificación, escogiendo aquel objeto que menos recientemente fue asignado a la ubicación actual (es decir si la ubicación actual es la i, se escoge aquel objeto j tal que `recency[i,j]` tenga el menor de iteración).

Memoria basada en frecuencia

Este tipo de memoria almacena componentes de las soluciones que forman parte de una solución con mayor frecuencia, i.e., la mayor cantidad de veces presentes en una solución y en una posición particular de la solución. La forma habitual de explotar este tipo de memoria es etiquetando las componentes seleccionadas de soluciones más frecuentemente escogidas. Esta memoria, permite "prohibir" que una hormiga escoja una componente de solución porque es muy frecuentemente escogida en las soluciones. Se busca a través de la "prohibición", generar soluciones que efectivamente se diferencien de las ya generadas, extendiendo de esta manera, la exploración del espacio de búsqueda. Inversamente, se puede usar esta información para "promover" su elección por considerarla atractiva ya que una gran mayoría de las hormigas la han elegido como parte de sus soluciones y por lo tanto, se considera deseable de que forme parte de una nueva solución.

Para el caso del QAP, la memoria basada en frecuencia almacena la cantidad de veces que el elemento i ha sido asignado a la ubicación j, $\forall i, j\ 1 \leq i, j \leq n$. Luego esta memoria permite a las hormigas tomar decisiones teniendo en cuenta que elementos son los *más frecuentemente* asignados a una ubicación particular, por tener asociada una alta frecuencia de asignación; o por el contrario tomar decisiones teniendo en cuenta que elementos son *menos frecuentemente* asignados a una ubicación particular, por tener asociada una baja frecuencia de asignación.

El algoritmo tiene asociado una matriz de frecuencias de $n \times n$, denominada `frecuency`. Donde la posición `frecuency[i,j]` almacena la cantidad de veces que el objeto j ha sido asignado a la ubicación i durante la ejecución del algoritmo. Luego, esta matriz se utiliza en el proceso de construcción de soluciones de dos maneras posibles:

- Intensificación, escogiendo aquel objeto que más veces fue asignado a la ubicación actual (es decir si la ubicación actual es la i, se escoge aquel objeto j tal que `frecuency[i,j]` sea mayor).
- Diversificación, escogiendo aquel objeto que menos veces fue asignado a la ubicación actual (es decir si la ubicación actual es la i, se escoge aquel objeto j tal que `frecuency[i,j]` sea menor).

4.4.2. Frecuencia de utilización de la memoria

Una pregunta que surge (ó debería surgir) es: ¿Con qué frecuencia las hormigas utilizan la memoria externa durante la construcción de soluciones?.

Durante la construcción de una solución, las hormigas van a aplicar una regla de elección de acción, similar a la que utiliza el algoritmo Ant Colony System[8] (ver Sección 3.3.3). Se introduce un parámetro $q_0 \in [0, 1]$, el cual va a permitir escoger una componente de solución favoreciendo la explotación de soluciones o la exploración de soluciones. Cuando una hormiga tiene que elegir una componente de una solución, primero genera un número aleatorio uniformemente distribuido en el intervalo $[0, 1]$, si este número es menor que el valor del parámetro q_0 se elige una componente para lograr explotación de soluciones; si el número generado es mayor que el parámetro q_0 se elige la componente favoreciendo la exploración de soluciones. Además, cada una de estas opciones a su vez incluye una decisión respecto de rastros de feromona o memoria explícita. En este trabajo, se proponen dos variantes para establecer el valor del parámetro q_0:

- Se establece un valor fijo. Generalmente, debería ser un valor cercano a 0.
- Se establece un valor que depende de una medida de convergencia del algoritmo (entropía y factor de ramificación-λ [24]).

Entropía de un conjunto de soluciones

La entropía de la población de soluciones es una medida que indica cuan lejos de converger está una población de soluciones. La misma puede ser vista como una medida de la diversidad en la población de soluciones. Para el QAP, la entropía de una ubicación se define como:

[8] También conocida como regla proporcional pseudo-aleatoria.

$$E_i = \frac{-\sum_{j=1}^{n} \left(\frac{n_{ij}}{m}\right) \log\left(\frac{n_{ij}}{m}\right)}{\log(n)}$$

donde n_{ij} representa el número de veces que el elemento i es asignado a la ubicación j en una población de tamaño m. La entropía de la población puede ser definida como:

$$E = \frac{\sum_{i=1}^{n} E_i}{n}$$

El valor de E varía entre 0 y 1. Un valor de $E = 0$ significa que solo una solución esta representada en la población, mientras que un valor cercano a 1 indica una amplia variedad en las asignaciones que están representadas en la población.

4.4.3. Incorporación de búsqueda local en el algoritmo propuesto

Para el algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System propuesto, se consideran dos algoritmos de búsqueda local: búsqueda local tradicional y *Búsqueda Tabú*. La elección de cual de los algoritmos de búsqueda local usar es un balance entre la velocidad de ejecución y la calidad de las soluciones que produce. A continuación se explica como se define la vecindad en los algoritmos de búsqueda local para el QAP y luego se detallan los algoritmos de búsqueda local para el algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System.

Definición de la vecindad

Para el QAP, la vecindad de una permutación ϕ esta definida por el conjunto de permutaciones que pueden ser obtenidas intercambiando dos elementos. La diferencia de la función objetivo $\Delta(\phi, r, s)$ obtenida intercambiando los elementos ϕ_s y ϕ_r puede ser computada en $O(n)$, usando la siguiente ecuación [71]:

$$\begin{aligned}
\Delta(\phi, r, s) \quad = \quad & b_{rr} \cdot (a_{\phi_s\phi_s} - a_{\phi_r\phi_r}) + b_{rs} \cdot (a_{\phi_s\phi_r} - a_{\phi_r\phi_s}) + \\
& b_{sr} \cdot (a_{\phi_r\phi_s} - a_{\phi_s\phi_r}) + b_{ss} \cdot (a_{\phi_r\phi_r} - a_{\phi_s\phi_s}) + \\
& \sum_{k=1, k\neq r, k\neq s}^{n} (b_{kr} \cdot (a_{\phi_k\phi_s} - a_{\phi_k\phi_r}) + b_{ks} \cdot (a_{\phi_k\phi_r} - a_{\phi_k\phi_s}) + \\
& b_{rk} \cdot (a_{\phi_s\phi_k} - a_{\phi_r\phi_k}) + b_{sk} \cdot (a_{\phi_r\phi_k} - a_{\phi_s\phi_k})) \qquad (4.15)
\end{aligned}$$

El efecto de un cambio particular puede ser evaluado más rápido usando información de iteraciones previas. Sea ϕ' la solución que es obtenida intercambiando los elementos r

y s en la solución ϕ, entonces para cambiar los elementos u y v, con $(\{u,v\} \cap \{r,s\} = \emptyset)$ el movimiento puede evaluarse en tiempo constante:

$$\begin{aligned}\Delta(\phi', u, v) \quad = \quad & \Delta(\phi, r, s) + (b_{ru} - b_{rv} + b_{sv} - b_{su}) \cdot (a_{\phi_s\phi_u} - a_{\phi_s\phi_v} + a_{\phi_r\phi_v} - a_{\phi_r\phi_u}) + \\ & (b_{ur} - b_{vr} + b_{vs} - b_{us}) \cdot (a_{\phi_u\phi_s} - a_{\phi_v\phi_s} + a_{\phi_v\phi_r} - a_{\phi_u\phi_r}) \end{aligned} \quad (4.16)$$

Búsqueda Local para $\mathcal{MAX} - \mathcal{MIN}$ Ant System

El algoritmo de búsqueda local más simple basado en la vecindad descrita anteriormente es el de mejora iterativa, también llamado `2-opt`. La mejora iterativa puede ser implementada usando una regla de transición de *first-improvement* ó de *best-improvement*. Mientras que en el primer caso un movimiento de mejora se realiza inmediatamente, en el segundo caso se examina la vecindad completa y se elige el movimiento que resulte en la mejor mejora. La regla *best-improvement* `2-opt` para el QAP se beneficia del hecho de que el efecto de intercambiar dos elementos puede calcularse más rápido usando la información de iteraciones anteriores (ver Ecuación 4.16); la primer iteración es de complejidad $O(n^3)$, mientras que las iteraciones subsecuentes pueden hacerse $O(n^2)$. Con la regla *first-improvement* `2-opt` normalmente se tienen que realizar más movimientos para alcanzar un mínimo local y cada chequeo de la vecindad completa es $O(n^3)$. Aún, los algoritmos *first-improvement* pueden ejecutarse más rápido si sólo un número limitado de iteraciones son realizadas o se aplican técnicas adicionales como el uso de *don't look bits* [5]. Adicionalmente, examinando la vecindad en orden aleatorio, puede ser obtenidos diferentes óptimos locales también al empezar de la misma solución inicial.

Búsqueda Tabú aplicada al QAP como algoritmo de búsqueda local

En Búsqueda Tabú[9], se elige una permutación aleatoria π^0 como solución inicial. En la iteración k, la elección de π^{k+1}, la próxima solución visitada, es la mejor solución de $N(\pi^k)$ que es permitida, incluso si π^{k+1} es peor que π^k, la solución actual en la iteración k. Para ser admitida, una solución debe satisfacer las siguientes condiciones (donde t y u son dos parámetros):

- Si $k > t$ y $\pi_r^k \neq i$, $\forall v$, $k - t \leq v \leq k$ (es decir, si el número k de iteraciones realizadas es mayor que t y el elemento i nunca ha estado en la ubicación r durante las últimas t iteraciones), entonces las permutaciones de $N(\pi^k)$ que no coloquen a i en la ubicación r son prohibidas.
- Si $\exists v$, $v \geq k - u$ tal que $\pi_r^v = i$, $\pi_s^v = j$ y si $\pi_r^k \neq i$, $\pi_s^k \neq j$ (es decir, durante los últimas u iteraciones, una solución tiene el elemento i colocado en la ubicación r y la unidad j colocada en la ubicación s), entonces está prohibido ubicar tanto i en el lugar r y j en el lugar s de nuevo (a menos que esta modificación mejore la calidad de la mejor solución encontrada hasta el momento).

[9]En particular esta implementación es la denominada Búsqueda Tabú Robusta, y fue desarrollada por Eric Taillard [70].

El objetivo de los parámetros t y u es evitar que el algoritmo siempre visite las mismas soluciones. En [70], se demuestra que la modificación de u aleatoriamente durante la búsqueda eligiendo su valor 10 % alrededor de n, conduce a un método que es bastante robusto. El parámetro t debe ser mayor que $|N(\pi)|$, en la práctica, valores entre $2n^2$ y $5n^2$ son convenientes. Es evidente que estos valores para t y u pueden ser malos para algunos problemas, pero, en general, producen resultados aceptables. El efecto de t es el de diversificar la búsqueda imponiendo soluciones dadas, aún cuando sus valores son muy altos. Así, este mecanismo puede considerarse como una memoria a largo plazo. Por el contrario, el mecanismo asociado con u puede considerarse como una memoria a corto plazo.

4.4.4. Algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System basado en memoria

El $\mathcal{MAX} - \mathcal{MIN}$ Ant System basado en memoria es similar al algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System presentado en la sección 3.3.2. La principal diferencia está en la manera en que se construyen las soluciones, aquí se incorpora la matriz externa como mecanismo adicional para lograr explícitamente intensificación/diversificación en la búsqueda.

Construcción de soluciones

Las hormigas (artificiales) construyen soluciones válidas para el QAP; asignan cada elemento a una única ubicación y ninguna ubicación es ocupada por más de un elemento. La solución construida corresponde a una permutación $\phi \in \Phi(n)$. La construcción de una solución involucra dos pasos. Primero, tiene que ser elegida una ubicación y luego se debe asignar un elemento libre a esa ubicación[10]. Para escoger la ubicación, se selecciona una ubicación i, aleatoriamente entre aquellas no ocupadas aún. Para el segundo paso se usan los rastros de feromona, τ_{ij} se refiere al deseo de asignar el elemento j en la ubicación i. Para asignar un elemento j a una ubicación desocupada i se utiliza la siguiente regla:

$$j = \begin{cases} T & \text{si } q < q_0 \text{ (Explotación)} \\ R & \text{si } q \geq q_0 \text{ (Exploración)} \end{cases} \tag{4.17}$$

Como ya se mencionó, esta regla es similar a la utilizada por Ant Colony System, con una probabilidad fija q_0 $(0 \leq q_0 \leq 1)$ la hormiga elige el "mejor elemento posible" de acuerdo al conocimiento adquirido (puede ser en base a la memoria externa o en base a los rastros de feromona), mientras que con probabilidad $(1 - q_0)$ realiza exploración

[10]Este orden es inverso al utilizado por la mayoría de las aplicaciones de algoritmos ACO al QAP. Estas implementaciones asocian los rastros de feromona a los arcos que van de elementos a ubicaciones (es decir, para la elección de la ubicación a la cual asignar el elemento), pero no a los arcos que van de ubicaciones a elementos (los cuales son elegidos por alguna regla probabilística o heurística que no está en función de los rastros de feromona).

controlada de nuevas soluciones (también en este caso, puede ser en base a la memoria externa o en base a los rastros de feromona), donde q es una variable aleatoria uniformemente distribuida en el intervalo $[0, 1]$.

Para el caso de la explotación de soluciones, se utiliza la regla:

$$T = \begin{cases} arg\,\text{máx}_{l \in \mathcal{N}_i^k}\{\texttt{memoria[i,l]}\} & \text{si } r < r_0 \\ arg\,\text{máx}_{l \in \mathcal{N}_i^k}\{\tau_{il}\} & \text{si } r \geq r_0 \end{cases} \tag{4.18}$$

En esta regla, con una probabilidad fija r_0 $(0 \leq r_0 \leq 1)$ se escoge aquel elemento que más veces fue asignado a la ubicación actual (frecuencia) ó que más recientemente fue asignado a la ubicación actual (reciente); mientras que con probabilidad $(1 - r_0)$ se elige el elemento más deseable de acuerdo al rastro de feromonas. Donde r es una variable aleatoria uniformemente distribuida en el intervalo [0,1]. Hay que mencionar que existe una diferencia entre el elemento más deseable para una ubicación (aquel que tiene mayor cantidad de rastro de feromonas asociado) y el elemento que más veces fue asignado a una ubicación (aquel que tiene un valor mayor en la matriz `memoria`), ya que los rastros de feromona representan una *distribución de probabilidad*.

Para el caso de la exploración de nuevas soluciones, se utiliza la regla:

$$R = \begin{cases} arg\,\text{mín}_{l \in \mathcal{N}_i^k}\{\texttt{memoria[i,l]}\} & \text{si } p < p_0 \\ \\ \frac{\tau_{ij}(t)}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}(t)} & \text{si } p \geq p_0 \end{cases} \tag{4.19}$$

En esta regla, con una probabilidad fija p_0 $(0 \leq p_0 \leq 1)$ se elige aquel elemento que menos veces fue asignado a la ubicación actual (frecuencia) ó que menos recientemente fue asignado a la ubicación actual (reciente); mientras que con probabilidad $(1 - p_0)$ se escoge aquel elemento de acuerdo a la regla de selección básica similar a la del algoritmo Ant System (notar que en este caso no se hace uso de información heurística). Donde p es una variable aleatoria uniformemente distribuida en el intervalo [0,1].

Algoritmos resultantes

La elección del tipo de `memoria` para la regla 4.18 y la regla 4.19, dan origen a cuatro variantes del algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System.

Tipo de memoria (Intensificación)	Tipo de memoria (Diversificación)
`frecuencia`	`frecuencia`
`frecuencia`	`reciente`
`reciente`	`frecuencia`
`reciente`	`reciente`

Los algoritmos serán referidos de aquí en más como:

- $\mathcal{MAX} - \mathcal{MIN}$ Ant System frecuencia-frecuencia ($\mathcal{MM}$AS-ff)
- $\mathcal{MAX} - \mathcal{MIN}$ Ant System frecuencia-reciente ($\mathcal{MM}$AS-fr)
- $\mathcal{MAX} - \mathcal{MIN}$ Ant System reciente-frecuencia ($\mathcal{MM}$AS-rf)
- $\mathcal{MAX} - \mathcal{MIN}$ Ant System reciente-reciente ($\mathcal{MM}$AS-rr)

Actualización de rastros de feromona

Después de que todas las hormigas han construido una solución, los rastros de feromona son actualizados de acuerdo a la Ecuación 3.10. Para el caso del QAP, $\Delta\tau_{ij}^{best}$ se define como:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/f(\phi_{best}) & \text{si la mejor hormiga asigno el elemento } j \text{ en la ubicación } i \text{ en su solución} \\ 0 & \text{en otro caso} \end{cases} \tag{4.20}$$

donde $f(\phi_{best})$ es costo de la solución (para la instancia de QAP) que generó la mejor hormiga. La hormiga a la que se le permite añadir feromona puede ser la que generó la mejor solución de la iteración actual, en cuyo caso $f(\phi_{best}) = f(\phi_{ib})$ donde ϕ_{ib} es la mejor solución encontrada en la iteración actual; o la que generó la mejor solución desde el inicio del algoritmo, en cuyo caso $f(\phi_{best}) = f(\phi_{bs})$ donde ϕ_{bs} es la mejor solución encontrada desde el inicio del algoritmo. Los resultados experimentales demuestran que el mejor rendimiento se obtiene incrementando gradualmente la frecuencia de elegir la mejor global para la actualización de feromona respecto de la mejor de la iteración.

De esta forma, si en las mejores soluciones los elementos son puestos frecuentemente en ubicaciones específicas, estas componentes de solución reciben una gran cantidad de feromona y, por tanto, los elementos se colocarán preferentemente en estas ubicaciones en futuras iteraciones del algoritmo. Para la aplicación de $\mathcal{MM}$AS al QAP por lo general se elegirá, la mejor solución global (es decir la mejor desde el inicio del algoritmo) para la actualización de feromona.

Límites de rastros de feromona

Los límites de rastros de feromona para la aplicación de $\mathcal{MM}$AS al QAP son elegidos del mismo modo que en la aplicación de $\mathcal{MM}$AS al TSP [24].

Inicialización y Reinicializaciín de rastros de feromona

La inicialización de rastros de feromona se realizó de acuerdo a la propuesta original del algoritmo $\mathcal{MM}$AS [69].

La reinicializaciín de rastros de feromona no se utilizó en los experimentos realizados, debido a que en pruebas previas los resultados no mejoraban el rendimiento de los algoritmos. Otra de las razones es que los algoritmos propuestos ya introducen un medio de exploración adicional a través de la memoria externa.

4.5. Resumen

Este capítulo, es uno de los más importantes del trabajo final, ya que aquí se detallan los algoritmos ACO propuestos para resolver el QAP. También se incluye una revisión de aplicaciones previas de algoritmos ACO al QAP, tanto versiones tradicionales como versiones relacionadas al uso de memoria externa.

Capítulo 5

Experimentos y Análisis de Resultados

En este capítulo se presentan los resultados obtenidos en las pruebas realizadas a los algoritmos propuestos y la comparación de estos resultados con el algoritmo $\mathcal{MM}$AS-QAP original. Se comienza indicando los problemas de prueba seleccionados y las características de cada uno de ellos; posteriormente se indica la configuración de los experimentos a realizar. Después se detallan los resultados de cada una de las ejecuciones del algoritmo y finalmente se realiza un análisis de los mismos.

5.1. Instancias utilizadas

Las instancias utilizadas son las descritas en la sección 2.4.2, las cuales fueron propuestas en [68]. En particular, se utilizaron un subconjunto de las instancias de tamaño 50. Se puede distinguir entre seis clases diferentes de instancias que difieren en la manera en que se combinan las diferentes variantes de la matriz de distancia y de la matriz de flujo. Las clases son:

- **RandomRandom:** Matriz de distancia aleatoria y flujos aleatorios;
- **RandomStructured:** Matriz de distancia aleatoria y flujos estructurados;
- **RandomStructuredPlus:** Matriz de distancia aleatoria y flujos estructurados con conexiones entre grupos de objetos;
- **GridRandom:** Matriz de distancia basada en una grilla y flujos aleatorios;
- **GridStructured:** Matriz de distancia basada en una grilla y flujos estructurados;
- **GridStructuredPlus:** Matriz de distancia basada en una grilla y flujos estructurados con conexiones entre grupos de objetos.

5.2. Descripción de los experimentos

Los valores de los parámetros para el algoritmo $\mathcal{MM}$AS-QAP estan configurados de acuerdo a [69] excepto que el valor de ρ se establecio en 0.2 ya que obtenía mejores resultados que la configuración de $\rho = 0.8$ propuesta en la literatura; además para todos los experimentos se trabajo con $m = 20$ hormigas (todas las hormigas aplican búsqueda local a la solución que generan). El parámetro de la Ecuación 4.17 se estableció en $q_0 = 0.1$ (valor sugerido en la literatura para un algoritmo similar [24]).

Las cuatro variantes de algoritmos fueron probadas con las instancias de QAP descritas en la sección anterior. Se seleccionaron 4 instancias de cada una de las 6 clases de instancias. Éstas fueron probadas para diferentes valores de los parámetros r_0 y p_0 (los cuales determinan el uso respectivo de la memoria, tanto para intensificar como para diversificar).

- **Probabilidad de usar memoria externa en explotación:** se testeó la influencia del parámetro r_0 en la Ecuación 4.18. Los valores considerados son $\{0.2, 0.5, 0.8\}$, para reflejar una frecuencia de uso limitada, intermedia y alta, respectivamente.
- **Probabilidad de usar memoria externa en exploración:** se testeó la influencia del parámetro p_0 en la Ecuación 4.19. Los valores considerados son $\{0.001, 0.01, 0.1\}$. En este caso, el uso de la memoria debía ser más restringido porque el rastro de feromonas es el principal componente de los algoritmos ACO.

En total se realizaron 216 experimentos (para cada experimento se realizaron series de 30 corridas independientes). El número máximo de iteraciones para cada corrida fue fijado en 1000.

5.2.1. Entorno computacional

Todos los algoritmos fueron programados en Lenguaje `C` y las ejecuciones se desarrollaron en computadoras Intel Core 2 Duo 2.13 GHz con 1 GB RAM, bajo el sistema operativo SUSE Linux 10.2.

5.3. Resultados

En esta sección, se muestran los resultados de las experimentos explicados en la sección anterior, para cada una de las cuatro variantes del algoritmo propuesto (todas las tablas tienen el mismo formato). Para cada combinación de los parámetros p_0 y r_0 se muestra el valor correspondiente al Porcentaje de Error Promedio respecto del mejor valor conocido (este valor de error promedio se calcula con respecto a las cuatro instancias de cada clase utilizadas). A través de los resultados obtenidos se intenta determinar cual es la configuración de parámetros con los que las distintas variantes obtienen los mejores resultados. Los valores que se muestran en las tablas permiten condensar la

Tabla 5.1: Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-ff. Los mejores resultados para cada clase de instancia son indicados en negrita.

p_0	r_0	Grid Random	Grid Structured	Grid StructuredPlus	Random Random	Random Structured	Random StructuredPlus
0.001	0.2	0.1235	0.1504	0.1668	0.0639	0.0964	0.0729
0.01	0.2	0.1215	0.1652	0.1929	0.0814	0.1002	0.1045
0.1	0.2	0.1530	0.2892	0.3321	0.1480	0.4166	0.4385
0.001	0.5	0.1019	0.1051	0.1242	0.0351	0.0587	0.0591
0.01	0.5	0.1046	0.1279	0.1279	0.0511	0.0615	0.0711
0.1	0.5	0.1535	0.2619	0.2894	0.1475	0.3700	0.3646
0.001	0.8	**0.0679**	**0.0442**	0.0616	0.0390	**0.0208**	**0.0150**
0.01	0.8	0.0729	0.0615	**0.0578**	**0.0360**	0.0371	0.0256
0.1	0.8	0.1498	0.2223	0.2455	0.1186	0.2436	0.2206

Tabla 5.2: Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-fr. Los mejores resultados para cada clase de instancia son indicados en negrita.

p_0	r_0	Grid Random	Grid Structured	Grid StructuredPlus	Random Random	Random Structured	Random StructuredPlus
0.001	0.2	0.1192	0.1520	0.1685	0.0645	0.1162	0.1072
0.01	0.2	0.1147	0.1565	0.1770	0.0745	0.0969	0.1024
0.1	0.2	0.1516	0.2894	0.3170	0.1516	0.4194	0.4169
0.001	0.5	0.0895	0.1001	0.1060	0.0438	0.0998	0.0501
0.01	0.5	0.0976	0.0987	0.1049	0.0572	0.0799	0.0601
0.1	0.5	0.1499	0.2602	0.2770	0.1247	0.3081	0.3186
0.001	0.8	**0.0645**	**0.0564**	0.0765	**0.0280**	**0.0220**	0.0447
0.01	0.8	0.0705	0.0690	**0.0745**	0.0375	0.0642	**0.0218**
0.1	0.8	0.1375	0.2163	0.2437	0.0974	0.2258	0.2307

información generada a partir de las 30 corridas para cada configuración de parámetros p_0 y r_0.

Los resultados obtenidos en las Tablas 5.1, 5.2, 5.3 y 5.4 muestran que el incremento del uso de la memoria externa en el proceso de explotación beneficia el rendimiento del algoritmo, respecto del uso de los rastros de feromona (los mejores resultados se obtienen con la configuración $r_0 = 0.8$). En el caso del uso de la memoria en proceso de exploración se confirma una de las suposiciones hechas al principio del capítulo ya que los mejores resultados se obtienen cuando la frecuencia de utilización de la memoria externa para exploración es reducida (la configuración $p_0 = 0.001$ es la que obtuvo los mejores resultados en casi todos los experimentos realizados; y en el resto de los casos el mejor resultado lo obtuvo la configuración $p_0 = 0.01$).

Tabla 5.3: Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-rf. Los mejores resultados para cada clase de instancia son indicados en negrita.

p_0	r_0	Grid Random	Grid Structured	Grid StructuredPlus	Random Random	Random Structured	Random StructuredPlus
0.001	0.2	0.1279	0.1483	0.1780	0.0533	0.1061	0.0815
0.01	0.2	0.1104	0.1499	0.1816	0.0815	0.1053	0.1235
0.1	0.2	0.1634	0.3168	0.3344	0.1537	0.4213	0.4549
0.001	0.5	0.1102	0.1124	0.1295	0.0434	0.0390	0.0396
0.01	0.5	0.1126	0.1252	0.1440	0.0510	0.0606	0.0400
0.1	0.5	0.1613	0.2796	0.3146	0.1545	0.3951	0.3918
0.001	0.8	**0.0751**	**0.0561**	**0.0707**	0.0423	**0.0222**	**0.0285**
0.01	0.8	0.0818	0.0807	0.0874	**0.0358**	0.0513	0.0361
0.1	0.8	0.1530	0.2685	0.2387	0.1382	0.2721	0.2595

Tabla 5.4: Comparación de valores para los parámetros p_0 y r_0 en $\mathcal{MM}$AS-rr. Los mejores resultados para cada clase de instancia son indicados en negrita.

p_0	r_0	Grid Random	Grid Structured	Grid StructuredPlus	Random Random	Random Structured	Random StructuredPlus
0.001	0.2	0.1269	0.1596	0.1791	0.0602	0.1112	0.1021
0.01	0.2	0.1268	0.1655	0.1945	0.0694	0.1226	0.0929
0.1	0.2	0.1570	0.2902	0.3248	0.1511	0.4140	0.3836
0.001	0.5	0.0979	0.1119	0.1155	0.0462	0.0600	0.0662
0.01	0.5	0.1106	0.1331	0.1527	0.0524	0.0603	0.0604
0.1	0.5	0.1532	0.2730	0.2666	0.1371	0.3241	0.3323
0.001	0.8	**0.0716**	**0.0610**	0.0708	**0.0417**	**0.0307**	0.0303
0.01	0.8	0.0750	0.0772	**0.0637**	0.0451	0.0686	**0.0228**
0.1	0.8	0.1490	0.2577	0.2415	0.1211	0.2215	0.2658

		$\mathcal{MM}$AS-ff			$\mathcal{MM}$AS-fr			$\mathcal{MM}$AS-rf			$\mathcal{MM}$AS-rr		
Instancia	M.V.C.[a]	Mejor	Media	Desv.	Mejor	Media	Desv.	Mejor	Media	Desv.	Mejor	Media	Desv.
GR.974820449	550969	**561152**	**564691.60**	1555.56	562101	565209.80	1535.53	563188	565974.43	1558.81	563067	565693.20	**1317.97**
GR.974820461	152116	**160825**	**163704.20**	1571.25	161057	163801.03	1770.99	162757	164978.77	**1235.38**	161891	165067.93	1489.12
GR.974820471	79749	87949	90554.53	1426.41	88073	**90050.97**	**1132.89**	89451	91335.20	1159.43	**87687**	91172.30	1650.25
GR.974820482	12728	14240	**15546.97**	**699.95**	**13962**	15757.63	800.18	14222	16157.03	884.38	14370	16040.53	1115.26
GS.974825925	4243	**4243**	4438.73	196.27	**4243**	4438.90	174.67	**4243**	**4438.17**	**163.00**	**4243**	4441.53	193.01
GS.974825930	16335	**17166**	19445.20	1205.77	17618	**18939.17**	963.11	17689	19289.13	950.87	17605	19104.80	**916.08**
GS.974825936	59994	64477	**68836.13**	2712.33	65188	69098.23	**2274.71**	**64046**	71223.03	3231.61	65084	72255.30	3181.36
GS.974825941	69820	**74208**	**78725.33**	2241.83	75340	79025.63	1952.82	76291	79975.40	1660.21	77052	80216.13	**1652.22**
GSP.974826006	3493	**3493**	3626.57	106.96	**3493**	3627.87	111.75	**3493**	3627.47	104.53	**3493**	**3592.60**	**81.02**
GSP.974826012	12939	14404	15392.17	610.51	14925	15903.07	766.51	**13911**	15426.37	1062.26	14255	**15330.73**	**523.51**
GSP.974826017	46340	**50113**	**54891.47**	2704.78	51784	55439.97	**2433.76**	52242	57250.30	2834.22	50759	56535.00	3322.01
GSP.974826022	65000	69868	74001.90	1909.26	**69755**	**73491.10**	1766.55	72227	75453.50	1814.44	72557	75722.73	**1449.76**
RR.974820375	15946893	16114813	16196832.63	45642.34	**16068979**	**16182558.83**	51002.60	16131322	16235888.93	48369.04	16142810	16221303.63	**35135.88**
RR.974820387	5951176	**6009955**	**6088489.77**	29887.03	6056662	6089937.00	**23015.46**	6042435	6115255.43	33413.42	6063247	6123522.47	27392.10
RR.974820399	1294453	1389492	**1450259.03**	**27762.71**	**1360667**	1463210.33	37235.94	1418167	1475507.77	33664.04	1417646	1478230.53	41285.51
RR.974820410	124931	133978	147384.50	8605.77	**129942**	**145182.80**	**8088.82**	132276	145950.73	8786.64	131409	145377.00	8757.91
RS.974823931	16107	**16315**	35666.33	15686.59	17064	32602.37	14108.78	16522	**29410.03**	**10143.73**	16616	31504.30	12679.89
RS.974823936	186096	193733	239754.67	30290.73	**186110**	248330.60	33518.17	195279	245006.73	31275.77	199155	**238673.60**	**29387.20**
RS.974823941	1528696	1531182	1649198.43	88048.14	1531130	1645610.27	**74867.36**	**1531124**	**1632745.37**	77054.21	1534709	1653011.60	91641.35
RS.974823946	2119306	2184139	**2257268.20**	**47087.62**	2185520	2267927.33	59075.09	**2152075**	2291478.17	68167.59	2168823	2312691.27	69195.98
RSP.974824391	7646	7868	14918.43	6900.66	7888	**12970.70**	6108.94	**7500**	13282.43	6211.18	7649	13013.90	**5862.24**
RSP.974824396	224556	**225187**	**295229.57**	27744.70	249448	304490.77	36010.62	246325	305846.70	33584.50	246480	302334.40	**27634.58**
RSP.974824401	1486604	**1490338**	**1554827.10**	60741.05	1498055	1558397.60	**47027.17**	1496789	1576734.90	79837.26	1493582	1569532.67	48368.77
RSP.974824406	1801276	**1850589**	**1926897.17**	39210.83	1877748	1941797.53	**35777.08**	1873363	1942347.50	40204.64	1852015	1960438.10	49933.67

Tabla 5.5: Resultados obtenidos de cada una de las variantes propuestas.

[a]M.V.C. Mejor Valor Conocido

Una vez establecida la configuración con la que se obtienen los mejores resultados, se presentan los resultados obtenidos de la ejecución de los distintos algoritmos utilizando esas configuraciones particulares de parámetros. Si se observa la Tabla 5.5, se puede deducir que, aunque los algoritmos no son capaces de obtener los mejores valores conocidos para la mayoría de las instancias, el rendimiento alcanzado es bastante razonable teniendo en cuenta la poca diferencia entre los mejores valores obtenidos y los mejores valores conocidos.

Uno de los principales motivos del rendimiento obtenido con los algoritmos propuestos, es que estos utilizan como algoritmo de búsqueda local al algoritmo de mejora iterativa **2-opt** el cual tiene un rendimiento muy inferior si se lo compara con un algoritmo de búsqueda local basado en Búsqueda Tabú [71]. A pesar de esto, los algoritmos propuestos alcanzan buenos resultados para algunas clases de instancias; llegando incluso a encontrar una nueva mejor solución para una instancia en particular. Además, si se compara con los resultados obtenidos por el algoritmo $\mathcal{MM}$AS tradicional el balance es altamente positivo ya que su rendimiento es muy inferior a cualquiera de los cuatro algoritmos propuestos (ver Tabla 5.6).

5.4. Análisis Estadístico

Los algoritmos fueron comparados usando el porcentaje de error promedio respecto de los mejores valores conocidos (los cuales nos fueron provistos por los autores de [68]). Para las comparaciones de los algoritmos se uso el análisis de varianza (ANOVA)[1] de un factor, para chequear la significancia estadística de las diferencias observadas en el rendimiento de los algoritmos (a todas las distribuciones se les aplico la prueba de Kolmogorov-Smirnov[2], para establecer si las distribuciones eran normales; los resultados obtenidos establecieron la normalidad de los valores e hicieron posible la aplicación de ANOVA). Además se utilizó el método de Tukey[3] para encontrar que valores de la media eran significantemente diferentes unos de otros.

Los resultados de la Tabla 5.7 demuestran que existen diferencias significativas entre los resultados obtenidos para las clases GridRandom, GridStructured y GridStructured-Plus, si se comparan los cuatro algoritmos propuestos. Mientras que existen diferencias significativas respecto de todas las clases de instancias si se comparan los cinco algorit-

[1]El análisis de varianza sirve para comparar si los valores de un conjunto de datos numéricos son significativamente distintos a los valores de otro o más conjuntos de datos. El procedimiento para comparar estos valores está basado en la varianza global observada en los grupos de datos numéricos a comparar. Típicamente, el análisis de varianza se utiliza para asociar una probabilidad a la conclusión de que la media de un grupo de valores es distinta de la media de otro grupo de valores.

[2]La prueba de Kolmogorov-Smirnov, es una prueba no paramétrica que se utiliza para determinar la bondad de ajuste de dos distribuciones de probabilidad entre sí

[3]El método de Tukey, es un procedimiento de comparación múltiple de un solo paso y es la prueba estadística usada generalmente en combinación con ANOVA para encontrar cuales de las medias son significantemente diferentes unas de otras. Este método compara todos los posibles pares de medias, y esta basado en una distribución q de *rango studentizado* (esta distribución es similar a la distribución t a partir de un *t-test*).

Tabla 5.6: Resultados obtenidos para el algoritmo $\mathcal{MM}$AS tradicional

Instancia	M.V.C.	Mejor	Media	Desviación
GR.974820449	550969	566199	568836.53	1240.89
GR.974820461	152116	166314	168439.40	1334.37
GR.974820471	79749	92267	94885.57	1263.84
GR.974820482	12728	17359	19013.83	630.34
GS.974825925	4243	4323	5001.87	547.36
GS.974825930	16335	23775	26857.73	1542.04
GS.974825936	59994	76511	80664.87	1880.30
GS.974825941	69820	81804	85497.47	1369.79
GSP.974826006	3493	3559	3977.83	364.92
GSP.974826012	12939	19133	22234.57	1218.31
GSP.974826017	46340	63057	67222.47	2020.51
GSP.974826022	65000	78345	80865.07	1828.07
RR.974820375	15946893	16254598	16313543.47	38768.09
RR.974820387	5951176	6110821	6187181.97	40712.65
RR.974820399	1294453	1526587	1606548.50	34857.15
RR.974820410	124931	138665	166053.37	13081.35
RS.974823931	16107	16903	36610.07	13377.60
RS.974823936	186096	225472	330673.80	66181.90
RS.974823941	1528696	1678382	1870993.43	129055.59
RS.974823946	2119306	2352924	2549577.60	76146.47
RSP.974824391	7646	9053	17615.80	10037.79
RSP.974824396	224556	238352	409955.63	72943.84
RSP.974824401	1486604	1694817	1836669.93	75954.48
RSP.974824406	1801276	1944434	2056365.43	50968.85

Tabla 5.7: Comparación de los mejores valores encontrados con los algoritmos $\mathcal{MM}$AS-ff, $\mathcal{MM}$AS-fr, $\mathcal{MM}$AS-rf, $\mathcal{MM}$AS-rr y $\mathcal{MM}$AS junto con los respectivos p-valores obtenidos de la prueba ANOVA de un factor.

Algoritmo	Grid Random	Grid Structured	Grid StructuredPlus	Random Random	Random Structured	Random StructuredPlus
$\mathcal{MM}$AS-ff	0.0679	0.0442	0.0616	0.0390	0.0208	0.0150
$\mathcal{MM}$AS-fr	0.0645	0.0564	0.0765	0.0280	0.0220	0.0447
$\mathcal{MM}$AS-rf	0.0751	0.0561	0.0707	0.0423	0.0222	0.0285
$\mathcal{MM}$AS-rr	0.0716	0.0610	0.0708	0.0417	0.0307	0.0303
(1) p-valor	**0.000027**	**0.0425**	**0.0051**	0.5816	0.7236	0.8605
$\mathcal{MM}$AS	0.1286	0.1734	0.1944	0.0740	0.1025	0.1024
(2) p-valor	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**	$\approx$ **0**

mos (los cuatro algoritmos propuestos y el algoritmo original). Estas diferencias surgen como resultado del bajo rendimiento del algoritmo $\mathcal{MM}$AS tradicional frente a los algoritmos propuestos.

Los p-valores[4] de la fila (1) son obtenidos del ANOVA entre los cuatro algoritmos propuestos ($\mathcal{MM}$AS-ff, $\mathcal{MM}$AS-fr, $\mathcal{MM}$AS-rf y $\mathcal{MM}$AS-rr); mientras que los p-valores de la fila (2) son obtenidos del ANOVA entre los cuatro algoritmos propuestos y el algoritmo $\mathcal{MM}$AS original (ver Tabla 5.7).

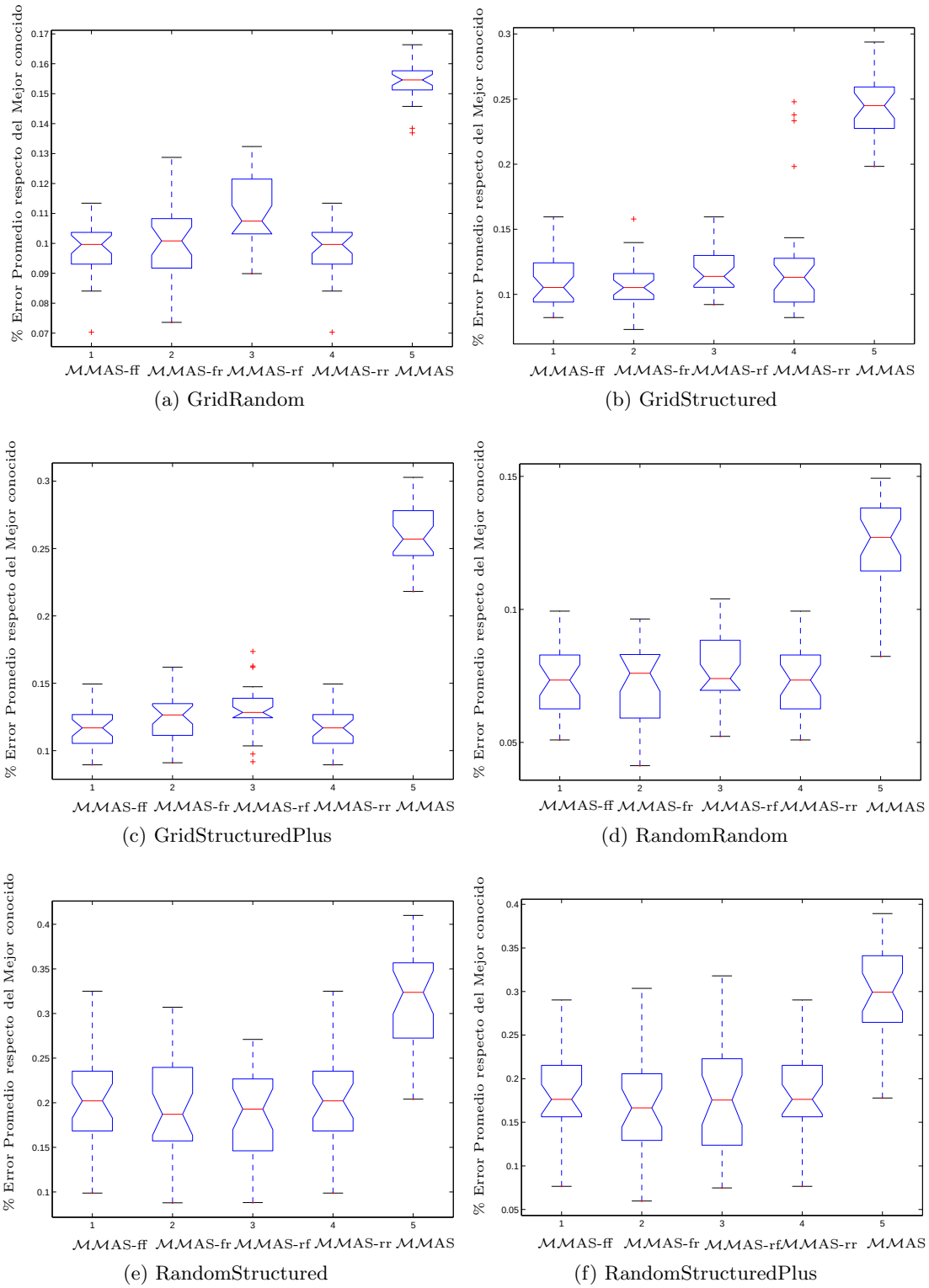
En la Figura 5.1 se muestran los diagramas de caja (Boxplot)[5] correspondientes a los valores obtenidos de la ejecución de los experimentos. Se puede observar, que para todas las clases de instancias los cuatro algoritmos propuestos obtienen mejores resultados comparados con el algoritmo $\mathcal{MM}$AS original. Esto se refleja en el hecho de que los mejores valores obtenidos con el algoritmo $\mathcal{MM}$AS son iguales o peores (para todas las clases de instancias) que los valores medios obtenidos por cada uno de los cuatro algoritmos propuestos.

Por otra parte si se comparan los resultados obtenidos entre los cuatro algoritmos propuestos, se observa que para las clases de instancias GridRandom, GridStructured y GridStructuredPlus el algoritmo $\mathcal{MM}$AS-rf obtiene resultados levemente inferiores a los obtenidos con los tres algoritmos restantes. Estas observaciones estan soportadas por los resultados obtenidos a través del método de Tukey, que afirman que existen diferencias significativas entre las medias obtenidas por el algoritmo $\mathcal{MM}$AS-rf y las medias obtenidas por los tres algoritmos restantes para esas clases de instancias.

En la Figura 5.2 se muestran gráficamente los resultados de la aplicación del método de Tukey a los resultados obtenidos por los algoritmos. De la Figura 5.2a se deduce que existen diferencias significativas entre el algoritmo $\mathcal{MM}$AS-rf y los algoritmos ($\mathcal{MM}$AS-ff, $\mathcal{MM}$AS-fr y $\mathcal{MM}$AS-rr), así como también respecto del algoritmo $\mathcal{MM}$AS original. En la Figura 5.2b no existen diferencias entre los algoritmos propuestos, pero claramente si existen diferencias entre los algoritmos propuestos y el algoritmo $\mathcal{MM}$AS original. En la Figura 5.2c se tiene que existen diferencias significativas entre el algoritmos $\mathcal{MM}$AS original y los cuatro algoritmos propuestos, además existen diferencias significativas entre el algoritmo $\mathcal{MM}$AS-rf y los algoritmos $\mathcal{MM}$AS-ff y $\mathcal{MM}$AS-rr. Para el caso de las Figuras 5.2d, 5.2e y 5.2f solo se observan diferencias significativas entre el algoritmo $\mathcal{MM}$AS original y los cuatro algoritmos propuestos.

[4]p-valores para la hipótesis "Las distribuciones de porcentaje de error promedio respecto del mejor valor conocido de las soluciones son iguales" para todos los algoritmos. El nivel de significancia con el cual se rechaza la hipótesis nula es 0.05.

[5]Los diagramas de caja son una manera conveniente de mostrar gráficamente grupos de datos numéricos a través de cinco valores (el valor mínimo, el cuartil inferior, la mediana, el cuartil superior, y el valor máximo). Un diagrama de caja también puede indicar cuales de las observaciones, pueden ser considerados valores aislados.

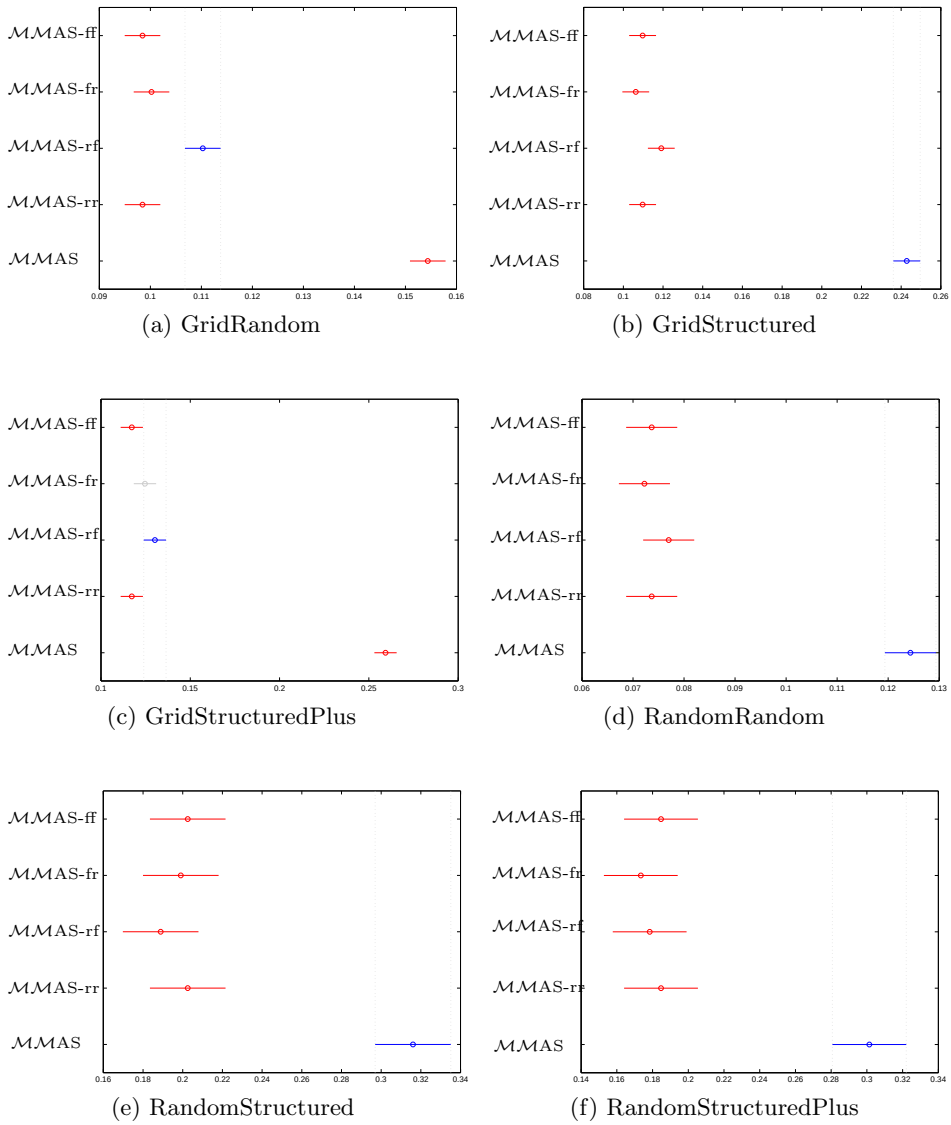


(a) GridRandom (b) GridStructured

(c) GridStructuredPlus (d) RandomRandom

(e) RandomStructured (f) RandomStructuredPlus

Figura 5.1: Boxplots del test de ANOVA aplicado a las seis clases de instancias. El eje y representa el porcentaje de error promedio respecto del mejor valor conocido. Sobre el eje de las x se muestran las respectivas variantes de los algoritmos incluidos el algoritmo $\mathcal{MM}$AS-QAP tradicional.



(a) GridRandom (b) GridStructured

(c) GridStructuredPlus (d) RandomRandom

(e) RandomStructured (f) RandomStructuredPlus

Figura 5.2: Representación gráfica de los resultados obtenidos con el método de Tukey. El punto medio de cada linea representa la media de los valores.

5.5. Resumen

Este capítulo, presentó los resultados obtenidos por los algoritmos propuestos sobre el conjunto de instancias seleccionados. Se realiza un estudio comparativo, así como también el análisis estadístico correspondiente. Estos permiten concluir que el rendimiento alcanzado por los algoritmos propuestos es superior al algoritmo tradicional en el cual están basados.

Capítulo 6

Conclusiones

En este capítulo se hace un resumen de los resultados conseguidos a lo largo de este trabjo final. Además de este resumen, se describen las dificultades en la realización del proyecto, terminando con un apartado donde se comentan las posibles extensiones futuras que se podrían realizar para obtener mejores resultados tanto en la resolución de los problemas aquí descritos como en una amplia gama de problemas de la vida real.

6.1. Resumen de resultados

Como resultados de la realización de este trabajo de fin de carrera, se puede indicar que se ha logrado implementar un algoritmo competitivo para resolver el Problema de Asignación Cuadrática. Entre los resultados más prominentes también se debe mencionar que se encontró una nueva mejor solución conocida para una instancia de QAP:

- **RandomStructuredPlus.974824391.n50.K10.m10.A100.00.B1.00.sp10.00.dat**

la anterior solución tenia un valor de 7646 y la nueva tiene un valor de 7500 [1]. La nueva mejor solución fue obtenida con el algoritmo $\mathcal{MM}$AS-rf.

Otro de los resultados más distinguidos fue la posibilidad de publicación de un artículo en un congreso internacional **"HM2009: 6th International Workshop on Hybrid Metaheuristics"** celebrado en Udine, Italia entre los días 16 y 17 de Octubre de 2009. Este artículo forma parte del volumen 5818 de Springer Verlag's Lecture Notes in Computer Science [4].

6.2. Conclusiones

En este trabajo final se propusieron variantes del algoritmo $\mathcal{MAX}-\mathcal{MIN}$ Ant System para resolver el Problema de Asignación Cuadrática. Las mismas han sido desarrollados tomando conceptos propios de otra metaheurística llamada Búsqueda Tabú.

[1] $\phi = (38, 20, 19, 14, 10, 49, 36, 9, 11, 3, 22, 1, 27, 24, 47, 46, 39, 15, 13, 7, 44, 34, 31, 2, 32, 45, 16, 23, 21, 17, 40, 8, 42, 6, 18, 12, 28, 4, 5, 29, 41, 25, 37, 35, 26, 48, 30, 0, 43, 33)$.

Los algoritmos propuestos introducen modificaciones en la manera de construir las soluciones por las hormigas, en el cual además de la matriz de feromonas se tiene una memoria externa que almacena información específica recolectada durante la ejecución del algoritmo.

En el Capítulo 5, se presentaron los resultados de experimentos con los mencionados algoritmos y en ellos se puede observar que la calidad de los resultados es considerablemente mejor que los resultados obtenidos con el algoritmo $\mathcal{MAX} - \mathcal{MIN}$ Ant System tradicional aplicado al mismo problema. Esto está soportado por un análisis estadístico realizado sobre los resultados obtenidos.

6.3. Trabajo Futuro

El conjunto completo de instancias de tamaño 50 contiene 136 instancias, en este trabajo se utilizaron un total de 24 instancias (4 de cada clase) para la realización de los experimentos debido al alcance del presente trabajo final. Sin embargo como trabajo futuro se planea realizar un estudio experimental exhaustivo con la totalidad de las instancias para poder determinar posibles relaciones entre el tipo de instancias y la configuración óptima de los parámetros involucrados en los algoritmos. Además se planea un análisis más riguroso en la determinación de la frecuencia del uso de la memoria externa, tanto en el proceso de explotación como en el proceso de exploración.

Bibliografía

[1] A. Acan. An external memory implementation in ant colony optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *ANTS 2004*, volume 3172 of *LNCS*, pages 73–84. Springer, Heidelberg, 2004.

[2] A. Acan. An external partial permutations memory for ant colony optimization. In G. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *LNCS*, pages 1–11. Springer-LNCS, 2005.

[3] R. K. Ahuja, J. B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27(10):917–934, 2000.

[4] F. Arito and G. Leguizamón. Incorporating Tabu Search principles into ACO algorithms. In M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf, editors, *Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 2009.

[5] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, Oxford, UK, 1996.

[6] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK, 1997.

[7] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[8] G. Brassard and P. Bratley. *Fundamentals of algorithmics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[9] D. E. Brown, C. L. Huntley, and A. R. Spillane. A parallel genetic heuristic for the quadratic assignment problem. In *Proceedings of the third international conference on Genetic algorithms*, pages 406–415, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[10] R. E. Burkard, S. Karisch, and F. Rendl. Qaplib - A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10(4):391–403, 1997.

[11] R. E. Burkard and J. Offermann. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Mathematical Methods of Operations Research (Zeitschrift für Operations Research)*, 21(4):B121–B132, 1977.

[12] R. E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17(2):169–174, 1984.

[13] R.E. Burkard and U. Fincke. Probabilistic asymptotic properties of some combinatorial optimization problems. *Discrete Applied Mathematics*, 12(1):21–29, 1985.

[14] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41(1-4):327–341, 1993.

[15] D. T. Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, 46(1):93–100, 1990.

[16] J.-L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[17] J. W. Dickey and J.W. Hopkins. Campus building arrangement using topaz. *Transportation Science*, (6):59–68, 1972.

[18] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1992.

[19] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, chapter 2, pages 11–32. McGraw-Hill, London, UK, 1999.

[20] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[21] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[22] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, 91-016, Dip. di Elettronica, Politecnico di Milano, Italy, 1991.

[23] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[24] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[25] Z. Drezner. A New Genetic Algorithm for the Quadratic Assignment Problem. *INFORMS JOURNAL ON COMPUTING*, 15(3):320–330, 2003.

[26] Z. Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, 33(5):475–480, 2005.

[27] Z. Drezner. The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research*, 160(2):416–422, 2005.

[28] Z. Drezner, P. M. Hahn, and E. D. Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 139(1):65–94, 2005.

[29] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. In *In P. Pardalos and H. Wolkowicz (Editors), Quadratic assignment and related problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 173–187, 1994.

[30] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[31] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[32] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(2):305–313, 1962.

[33] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[34] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[35] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, 2003.

[36] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1990.

[37] P.-P. Grassé. Recherches sur la biologie des termites champignonnistes (*Macrotermitinae*). *Annales des Sciences Naturelles, Zoologie*, 11(6):97–171, 1944.

[38] P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *bellicositermes natalensis* et *cubitermes sp.* la théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[39] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1999.

[40] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, MI, 1975.

[41] D. S. Johnson and L. A. Mcgeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[42] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search. *Journal of Computer and System Sciences*, 37:79–100, 1988.

[43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[44] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76, 1957.

[45] Y. Li and P. M. Pardalos. Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, 1(2):163–184, 1992.

[46] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[47] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.

[48] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[49] V. Maniezzo, A. Colorni, and M. Dorigo. The ant system applied to the quadratic assignment problem. Technical report, IRIDIA/94-28, IRIDIA, Université Libre de Bruxelles, Belgium, 1994.

[50] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. In *IEEE Transactions on Evolutionary Computation*, pages 4(4):337–352, 2000.

[51] A. Misevicius. A new improved simulated annealing algorithm for the quadratic assignment problem. *Information Technology and Control*, 4(17):29–38, 2000.

[52] A. Misevicius. A modification of tabu search and its applications to the quadratic assignment problem. *Information Technology and Control*, 2(27):12–20, 2003.

[53] A. Misevicius. An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowledge-Based Systems*, 17(2-4):65–73, 2004.

[54] A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30(1):95–111, 2005.

[55] A. Misevicius. A fast hybrid genetic algorithm for the quadratic assignment problem. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation: GECCO 2006*, pages 1257–1264, New York, NY, USA, 2006. ACM.

[56] A. Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4):497–514, 2003.

[57] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 416–421, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[58] V.Ñissen. Solving the quadratic assignment problem with clues from nature. *IEEE Transactions on Neural Networks*, 1(5):66–72, 1994.

[59] Christopher E. Nugent, Thomas E. Vollmann, and John Ruml. An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. *Operations Research*, 16(1):150–173, 1968.

[60] I.H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–628, 1996.

[61] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[62] J. M. Pasteels, J.-L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies. (i): Trail recruitment to newly discovered food sources. *Experientia. Supplementum*, 54:155–175, 1987.

[63] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[64] J. Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. *Computers and Operations Research*, 21(8):855–865, 1994.

[65] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3(1):37–50, 1961.

[66] T. Stützle. $\mathcal{MAX} - \mathcal{MIN}$ for the quadratic assignment problem. Technical report, AIDA-97-4, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 1997.

[67] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report, AIDA-99-03, FG Intellektik, FB Informatik, TU Darmstadt, 1999.

[68] T. Stützle and S. Fernandes. New benchmark instances for the QAP and the experimental analysis of algorithms. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 199–209, Berlin, Germany, 2004. Springer-Verlag.

[69] T. Stützle and H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[70] É. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[71] É. D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.

[72] É. D. Taillard. Fant: Fast ant system. Technical report, IDSIA-46-98, IDSIA, Lugano, Switzerland, 1998.

[73] É. D. Taillard and L. M. Gambardella. Adaptive memories for the quadratic assignment problem. Technical report, IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.

[74] D. M. Tate and A. E. Smith. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 1(22):73–83, 1995.

[75] S. Tsutsui. *c*AS: Ant colony optimization with cunning ants. In T. P. Runarsson *et al.*, editor, *Proc. of the 9th Int. Conf. on Parallel Problem Solving from Nature (PPSN IX)*, volume 4193 of *LNCS*, pages 162–171. Springer, Heidelberg, 2006.

[76] W. Wiesemann and T. Stützle. Iterated ants: An experimental study for the quadratic assignment problem. In M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, editors, *ANTS 2006*, volume 4150 of *LNCS*, pages 179–190. Springer, Heidelberg, 2006.

[77] Mickey R. Wilhelm and T. L. Ward. Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, 19(1):107–119, 1987.