Ant System

Marta Almirón

Trabajo presentado como requisito para optar al título de Máster en Ingeniería de Sistemas

Asesor:
D.Sc. Benjamín Barán

Universidad Nacional de Asunción
Diciembre-2000

Indice General

Indice de Figuras

Indice de Tablas

Indice de Pseudocódigos

Capítulo 1

Introducción

1.1 Consideraciones iniciales

El problema de la *explosión combinatoria* se encuentra en situaciones donde dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de estos, permitiendo una vasta cantidad de posibilidades. Los intentos por tratar con el problema han encontrado muchos obstáculos, por ejemplo, no es suficiente contar con un conocimiento experto para manejarlos de manera efectiva. Así mismo, no es suficiente confiar en la velocidad de procesamiento de una computadora, con el objeto de buscar la solución del problema de manera exhaustiva, en el conjunto de posibilidades dado su rápido crecimiento (exponencial) con el tamaño del problema. Situaciones de este tipo ocurren en problemas de inversión financiera, manejo de inventarios, diseño de circuitos integrados, manejo de recursos hidráulicos, mantenimiento de sistemas, por citar solo algunos ejemplos prácticos.

En la actualidad, la investigación de este tipo de problemas se ha dirigido hacia el diseño de buenas heurísticas, es decir, algoritmos eficientes con respecto al tiempo de cómputo y al espacio de memoria, y con cierta verosimilitud de entregar una solución buena, esto es, relativamente cercana a la óptima, mediante el examen de un subconjunto relativamente pequeño de soluciones del dominio de definición del problema.

La observación de la naturaleza ha sido una de las principales fuentes de inspiración para la propuesta de nuevos paradigmas computacionales. La gran variedad de problemas de optimización que ésta presenta, tales como: la supervivencia, la evolución de las especies, la búsqueda del camino crítico, entre otros, sugirieron diversos algoritmos computacionales que se inspiran en procesos naturales.

La clave para tratar con tales problemas es ir un paso más allá de la aplicación directa de la destreza y del conocimiento del experto, surgiendo así la Inteligencia Artificial como una técnica alternativa para la resolución de problemas de dimensiones y complejidad crecientes.

1.2 Técnicas de Inteligencia Artificial

En la naturaleza, los seres vivos se enfrentan a problemas que deben resolver con éxito para desarrollarse, como obtener más luz del sol, o conseguir alimento para su subsistencia. La sapiencia de la naturaleza para resolver estos problemas ha sido siempre admirada e imitada en muchos campos de la ciencia. Una de ellas, la computación, ha creado técnicas de Inteligencia Artificial con el objeto de plasmar en sistemas computacionales la esencia misma de la naturaleza observable.

En el área de la Inteligencia Artificial (IA) se pueden observar, a grandes rasgos, dos enfoques diferentes en función del objetivo [Her99b]:

- La concepción de IA como el intento de desarrollar una tecnología capaz de suministrar al ordenador capacidades de razonamiento o discernimiento similares, o aparentemente similares, a las de la inteligencia humana.
- La concepción de IA como investigación relativa a los mecanismos de inteligencia humana (por extensión, investigación relativa a la vida y al universo), que emplea el ordenador como herramienta de simulación para la validación de teorías.

El primer enfoque es por lo general el más práctico, se centra en los resultados obtenidos, en la utilidad y no tanto en el método. En este enfoque se encuadran, por ejemplo, los Sistemas Expertos. El segundo enfoque está orientado a la creación de un sistema artificial que sea capaz de realizar los procesos cognitivos humanos. Desde este punto de vista, no es tan importante la utilidad del sistema creado (qué hace), como el método empleado (cómo lo hace).

En cuanto a los medios utilizados, se identifican también dos enfoques:

- Enfoque *Top-Down*, o Enfoque simbólico. Se simulan directamente las características inteligentes del ser humano.
- Enfoque *Bottom-Up*, o Enfoque subsimbólico. Se caracteriza por crear sistemas con capacidad propia de aprendizaje. Esto se puede obtener a nivel de individuo, imitando el cerebro, o a nivel de especie, imitando la evolución, o el comportamiento de diversas especies para la subsistencia.

Una posible clasificación de los paradigmas de Inteligencia Artificial es presentada a continuación [Her99b]:

- Enfoque simbólico *(Top-Down)*
  - Sistemas Expertos *(Expert System)*
  - Lógica Difusa *(Fuzzy Logic)*
- Enfoque subsimbólico *(Bottom-Up)*
  - Computación Evolutiva *(Evolutionary Computation)*
    - Templado Simulado *(Simulated Annealing)*
    - Algoritmos Genéticos *(Genetic Algorithms)*
    - Programación Genética *(Genetic Programming)*
    - Estrategias Evolutivas *(Evolutionary Strategies)*
    - Clasificadores Genéticos *(Genetic Clasifiers)*
    - Programas Evolutivos *(Evolutionary Programming)*
    - Algoritmos Miméticos *(Memetic Algorithms)*
  - Redes Neuronales Artificiales *(Artificial Neural Networks)*
  - Vida Artificial *(Artificial Life)*

El **Sistema Experto** [Ign91] es un sistema capaz, en cierta medida, de reemplazar a un experto en un dominio relativamente estrecho. Un sistema experto pretende dar a la computadora un cierto conocimiento, una cierta habilidad sobre una materia en particular. En la actualidad, los sistemas expertos son utilizados en el campo de la medicina, la investigación petrolífera, en el estudio de factibilidad de créditos y en la predicción de subidas o caídas de acciones en el mercado, detección de fraudes de seguros [SR97], entre otros. *Eliza* fue uno de los primeros sistemas expertos. Éste es un "programa terapista", creado por el pionero de la inteligencia artificial Joseph Weizenbaum en el MIT. Una persona puede comunicarse con este programa como si lo estuviera haciendo con un profesional terapista. Los objetivos perseguidos por un sistema experto son básicamente los siguientes:

- Adquisición de conocimientos, concernientes tanto al saber como al saber hacer. Estos conocimientos deben ser presentados en forma lo más próxima posible a aquella empleada por los expertos del dominio.
- Explotación completa de los conocimientos adquiridos para deducir nuevos conocimientos o para responder a una pregunta. Agregar y suprimir fácilmente nuevos conocimientos.

Para alcanzar estos objetivos, un sistema experto utiliza [Ple95]:

- Un lenguaje de expresión de los conocimientos que permita expresar de manera comprensible por el sistema, los conocimientos provistos por los expertos.
- Una base de conocimientos que contenga la experteza necesaria para que el sistema funcione. Esta experteza se clasifica en dos tipos diferentes: el conocimiento establecido y el conocimiento de deducción. De este modo, la base de conocimientos de un sistema experto está constituida por dos conjuntos que se denominan base de hechos y base de reglas respectivamente. La base de hechos contiene los hechos establecidos en tanto que la base de reglas contiene las reglas de deducción. La base de conocimiento de un sistema experto puede contener un número muy importante de conocimientos.
- Un motor de inferencia, que es un programa capaz de aplicar las reglas a los diferentes hechos establecidos para deducir nuevos hechos. En el tiempo de

> vida del sistema, nuevas verdades pueden aparecer y por tanto enriquecer la base de hechos, pero también pueden descubrirse nuevas reglas de razonamiento que se agregarán a la base de reglas.

La **Lógica Difusa** [BNW96] fue introducida por primera vez en 1965 por Lotfi A. Zadeh, profesor de informática en la Universidad de California en Berkeley. A cierto nivel, la lógica difusa puede ser vista como un lenguaje que permite trasladar sentencias sofisticadas del lenguaje natural a un lenguaje matemático formal. La mayoría de los fenómenos que encontramos cada día son imprecisos, es decir, tienen implícito un cierto grado de difusidad en la descripción de su naturaleza. Esta imprecisión puede estar asociada con su forma, posición, momento, color, textura, o incluso en la semántica que describe lo que son.

En muchos casos, el mismo concepto puede tener diferentes grados de imprecisión en diferentes contextos o tiempo. Un día cálido en invierno no es exactamente lo mismo que un día cálido en primavera. La definición exacta de cuando la temperatura va de templada a caliente es imprecisa - no podemos identificar un punto simple de templado, así que emigramos a un simple grado, la temperatura es ahora considerada caliente. Este tipo de imprecisión o difusidad asociado continuamente a los fenómenos es común en todos los campos de estudio: sociología, física, biología, finanzas, ingeniería, oceanografía, psicología, etc.

En 1994, la teoría de la lógica difusa [Cha98] se encontraba en la cumbre, pero esta idea no es nueva, sus orígenes se remontan hasta 2.500 años atrás. Aún Aristóteles consideraba que existían ciertos grados de veracidad y falsedad. Platón había considerado ya grados de pertenencia.

En el siglo XVIII el filósofo y obispo anglicano irlandés George Berkeley y el filósofo escocés David Hume, describieron que el núcleo de un concepto atrae conceptos similares. Hume en particular, creía en la lógica del sentido común, el razonamiento basado en el conocimiento que la gente adquiere en forma ordinaria mediante vivencias en el

mundo. En Alemania, Immanuel Kant, consideraba que solo los matemáticos podían proveer definiciones claras, y muchos principios contradictorios no tenían solución. Por ejemplo la materia podía ser dividida infinitamente y al mismo tiempo no podía ser dividida infinitamente. Particularmente la escuela americana de la filosofía llamada pragmatismo, fundada por Charles Sanders Peirce, cuyas ideas se fundamentaron en estos conceptos, fue el primero en considerar vaguedades, más que falso o verdadero, como forma de acercamiento al mundo y a la forma en que la gente funciona.

La idea de que la lógica produce contradicciones fue popularizada por el filósofo y matemático británico Bertrand Russell, a principios del siglo XX. Estudió las vaguedades del lenguaje, concluyendo con precisión que la vaguedad es un grado. El filósofo austríaco Ludwing Wittgenstein estudió las formas en las que una palabra puede ser empleada para muchas cosas que tienen algo en común. La primera lógica de vaguedades fue desarrollada en 1920 por el filósofo Ja Lukasiewicz, quien visualizó los conjuntos con un posible grado de pertenencia con valores de 0 y 1, después los extendió a un número infinito de valores entre 0 y 1. En los años sesentas, Lofti Zadeh inventó la lógica difusa, que combina los conceptos de la lógica y de los conjuntos de Lukasiewicz mediante la definición de grados de pertenencia.

La lógica difusa ha emergido como una herramienta provechosa para controlar procesos industriales complejos, aparatos electrónicos del hogar y de los hospitales, sistemas de la diagnósticos y otros sistemas expertos, en general para los procesos muy complejos, donde no hay un modelo matemático simple [NAT96, BRB^{+}99] y para los procesos altamente no lineales.

La **Computación Evolutiva** [BHS97], agrupa paradigmas muy relacionados, basándose en la imitación de varios procesos naturales que intervienen en la evolución de las especies (selección natural, mutaciones, cruzamientos). Tratan de resolver complejos problemas de búsqueda, optimización, aprendizaje, predicción o clasificación. A continuación un breve resumen de cada uno de estos paradigmas.

El **Templado Simulado** [FS93], cuyo origen está en los procedimientos físicos de solidificación controlada, que consisten en calentar un sólido hasta que se funde, y seguidamente, ir enfriándolo de forma que cristalice en una estructura perfecta, sin malformaciones locales.

En el Templado Simulado se parte de una solución inicial que se va modificando a cada iteración. A medida que avanza el algoritmo, éste será cada vez más exigente con las soluciones aceptadas. En la analogía con la solidificación física, se trata de tener un cuerpo (problema) a alta temperatura (admitiendo soluciones muy diversas a pesar de ofrecer malos resultados) e ir disminuyendo la temperatura (aumentando la exigencia del algoritmo) de forma que termine por solidificarse según la forma deseada (ofreciendo el mejor resultado).

En la década de los ochenta se propuso que esta simulación podría aplicarse para obtener las soluciones factibles de un problema de optimización. En un problema de minimización, los métodos tradicionales de búsqueda de soluciones emplean una estrategia descendente, en la que la búsqueda siempre va en una dirección que produce una mejora. Una estrategia de este tipo puede dar lugar a alcanzar mínimos locales en lugar del mínimo global (dependiendo de la naturaleza del espacio de soluciones factibles y de la técnica aplicada). Las soluciones obtenidas por estas estrategias descendentes dependen fuertemente de las soluciones iniciales consideradas [Ber98].

El Templado Simulado posibilita que la búsqueda no sea siempre descendente, siendo este evento gobernado por una función de probabilidad (o temperatura) que cambia durante el proceso de ejecución del algoritmo.

La base para que sea posible este comportamiento diferenciador de los métodos tradicionales en la búsqueda de soluciones está basada en la estadística termodinámica. Una posible aplicación del algoritmo puede llevarse a cabo generando una perturbación y calculando el cambio de energía resultante. Si la energía ha disminuido el sistema se mueve a un nuevo estado. Si la energía ha aumentado, el nuevo estado se acepta de acuerdo con

una probabilidad dada. El proceso se repite un determinado número de iteraciones a cada una de las temperaturas, después de lo cual, la temperatura disminuye hasta alcanzar un estado de equilibrio.

Algunas de las aplicaciones del recocido simulado son el diseño electrónico de sistemas, diseño de redes de computadoras, diseño de circuitos integrados [HB97], optimización de costes de producción [MVO98], así como problemas de asignación de tiempos y espacios.

El Templado Simulado es una herramienta válida para la resolución de problemas combinatorios. Las desventajas se presentan por los altos tiempos de cálculo necesarios para acercarse al óptimo y la dificultad para ajustar adecuadamente los parámetros que controlan el algoritmo.

Los **Algoritmos Genéticos** [Gol89] fueron introducidos por John Holland en 1970 inspirándose en el proceso observado en la evolución natural de los seres vivos.

Los biólogos han estudiado con detalle los mecanismos de la evolución, y aunque quedan parcelas por entender, muchos aspectos están bastante explicados.

De manera muy general, se puede decir que en la evolución de los seres vivos el problema al que cada individuo se enfrenta es la supervivencia. Para ello, cuenta con las habilidades innatas provistas en su material genético. A nivel de los genes, el problema es el de buscar aquellas adaptaciones beneficiosas en un medio hostil y cambiante.

Debido en parte a la selección natural, cada especie gana una cierta cantidad de “conocimiento”, el cual es incorporado a la información de sus cromosomas. De esta manera, la evolución tiene lugar en los cromosomas, en donde está codificada la información del ser vivo.

La información almacenada en el cromosoma varía de unas generaciones a otras. En el proceso de formación de un nuevo individuo, se combina la información cromosómica de los progenitores aunque la forma exacta en que se realiza es aún desconocida.

Aunque muchos aspectos están todavía por discernir, existen unos principios generales ampliamente aceptados por la comunidad científica. Algunos de estos son:

1. La evolución opera en los mismos cromosomas en lugar de operar en los individuos a los que representan.
2. La selección natural es el proceso por el que los cromosomas con "buenas estructuras" se reproducen más a menudo que los demás.
3. En el proceso de reproducción tiene lugar la evolución mediante el cruzamiento de los cromosomas de los progenitores. Llamamos cruzamiento a este proceso en el que se forma el cromosoma del descendiente. También son de tener en cuenta las mutaciones que pueden alterar dichos códigos.
4. La evolución biológica no tiene memoria en el sentido de que en la formación de los cromosomas únicamente se considera la información del período anterior.

Los Algoritmos Genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en un string (vector) a modo de cromosoma. A tal efecto se introduce una función de evaluación de los cromosomas, que se conoce como *fitness* y que está basada en la función objetivo del problema. Igualmente, se introduce un mecanismo de selección de manera que los cromosomas con mejor evaluación sean escogidos para reproducirse más a menudo que los de peor *fitness*.

Los algoritmos desarrollados por Holland en base a lo arriba expuesto inicialmente eran sencillos pero dieron buenos resultados en problemas considerados difíciles [Hol75].

Los Algoritmos Genéticos están basados en integrar e implementar eficientemente dos ideas fundamentales:

- Las representaciones simples, como strings binarios, de las soluciones del problema.
- La realización de transformaciones simples para modificar y mejorar estas representaciones.

Para llevar a la práctica el esquema anterior y concretarlo en un algoritmo, hay que especificar los siguientes elementos:

- una representación cromosómica;
- una población inicial;
- una medida de evaluación *(fitness);*
- un criterio de selección / eliminación de cromosomas;
- una o varias operaciones de cruzamiento;
- una o varias operaciones de mutación

A continuación se comentan estos elementos. En los primeros trabajos las soluciones se representaban por strings binarios, es decir, listas de 1s y 0s. Este tipo de representaciones ha sido ampliamente utilizada incluso en problemas en donde no es muy natural.

La población inicial suele ser generada en forma aleatoria. Sin embargo, últimamente se están utilizando métodos heurísticos para generar soluciones iniciales de buena calidad.

Respecto a la evaluación de los cromosomas, se suele utilizar la calidad como medida de la bondad según el valor de la función objetivo en el que se puede añadir un factor de penalización para controlar la infactibilidad.

La selección de los padres viene dada habitualmente mediante probabilidades calculadas según su *fitness*. Uno de los procedimientos más utilizado es el denominado “de la ruleta”, en donde cada individuo tiene una sección de una ruleta circular cuyo tamaño es directamente proporcional a su calidad (medida de evaluación).

Los operadores de cruzamiento más utilizados son:

- De un punto: Se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus bits.
- De dos puntos: Se eligen dos puntos de ruptura al azar para intercambiar.
- Uniforme: En cada bit se elige al azar un padre para que contribuya con su bit a un hijo, mientras que el segundo hijo recibe el bit del otro padre.

La operación de mutación más sencilla, y una de la más utilizadas, consiste en reemplazar con cierta probabilidad el valor de un bit. Se puede notar que el papel que juega la mutación es el de introducir un factor de diversificación ya que, en ocasiones, la convergencia del procedimiento a buenas soluciones puede ser prematura y quedarse atrapado en óptimos locales. Otra forma obvia de introducir nuevos elementos en una población es cruzar elementos tomados al azar sin considerar su *fitness*.

Los Algoritmos Genéticos han sido aplicados con éxito a numerosos problemas, entre ellos los problemas de optimización, como los citados a continuación:

- Diseño óptimo de redes [BL99, DAS97].
- Diseño óptimo de sistemas de distribución de energía eléctrica [Ber98].
- Optimización del caudal turbinado de una represa hidroeléctrica [BCC98].

En la **Programación Genética** [KB96] los elementos de una cadena (genes) son instrucciones en un lenguaje de programación. Codificando en el código genético operaciones aritméticas básicas se han conseguido aproximadores funcionales que dan buenos resultados. Pese a los avances de John R. Koza en 1992 [Orc1], creando árboles de análisis, aun se está trabajando en conseguir que la programación genética se pueda aplicar sobre un lenguaje de programación común. Se espera que en poco tiempo la programación genética pueda operar plenamente en lenguajes funcionales no tipados.

Las **Estrategias Evolutivas** fueron desarrolladas por Rechenberg [Rec94] y Schwefel [Sch95], y extendidas por Herdy [Her92], Kursawe, Ostermeier, Rudolph,

Schewefel y otros. Fueron diseñadas originalmente con el objeto de resolver problemas discretos y contínuos de una manera eficiente, además de problemas de optimización paramétrica. Durante la década de los ochentas los avances tecnológicos en el diseño de computadoras permitieron la aplicación de algoritmos evolutivos para solucionar difíciles problemas de optimización del mundo real.

Las estrategias evolutivas hacen búsquedas más exploratorias que los algoritmos genéticos. Es una buena solución si tenemos un problema paramétrico del que no conocemos casi nada.

Los **Clasificadores Genéticos** [BGH89] son algoritmos evolutivos cuya finalidad es obtener un sistema clasificador. Normalmente los clasificadores genéticos hacen evolucionar un conjunto de reglas que serán las encargadas de realizar la clasificación, y en ese caso se pueden considerar como un tipo de Programación Evolutiva.

La **Programación Evolutiva** [BHS97] fue introducida por Fogel y extendida por Burgin [Bur69], Atmar [Atm76], Fogel y otros, fue originalmente ofrecida como una tentativa de crear inteligencia artificial. El método fue desarrollado para utilizar máquinas de estados finitos (*Finite State Machine* - FSM) con el objeto de predecir eventos sobre la base de observaciones anteriores. Un FSM es una máquina abstracta que transforma una secuencia de símbolos de entrada en una secuencia de símbolos de salida. La transformación depende de un conjunto finito de estados y de un conjunto finito de reglas de transición de estados. El funcionamiento de un FSM con respecto a su ambiente se puede medir en base a la capacidad de la predicción de las máquinas, es decir, comparando cada símbolo de salida con el siguiente símbolo de entrada y midiendo el valor de una predicción con una función de rentabilidad.

Los **Algoritmos Miméticos** [Orc2] se definen como la hibridización del algoritmo genético, que se encargará de realizar una búsqueda exploratoria, con un algoritmo explotatorio; sólo que no se intercambian los métodos entre sí según la velocidad de convergencia, sino que la búsqueda explotatoria mejora los individuos de la población.

La diferencia conceptual entre híbridos y miméticos es muy sutil, pero importante. En los algoritmos híbridos cambiamos de un método a otro cuando nos bloqueamos, de forma que usamos, al principio de la optimización algoritmos genéticos y, al final, algoritmos exploratorios. El algoritmo híbrido consiste en un algoritmo genético que va a buscar los pozos donde puede estar el mínimo global, y el algoritmo exploratorio va a buscar el mínimo. En el mimético empleamos la búsqueda exploratoria para que todos los individuos de la población del algoritmo genético sean mínimos locales. Así, lo que tendremos no será una competición entre elementos de los distintos pozos, sino entre mínimos de un mismo pozo. Esto acelera enormemente la convergencia, ya que evaluamos cada individuo con lo mejor que puede aportarnos. En cambio, aumentan los cálculos del método exploratorio.

El término *Memetic Algorithms* [Mos98] aparece por primera vez en la literatura computacional en 1989 en *"On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms"*. Este trabajo discute una heurística desarrollada por Pablo Moscato y Michael G. Norman, utilizando *Simulated Annealing* para búsqueda local con un juego competitivo y cooperativo entre agentes, mezclado con el uso de un operador de cruzamiento.

Las **Redes Neuronales** *(Neural Networks)* [FS93], son sistemas que intentan simular el cerebro, inspirados en su capacidad de aprender, sea por medio de nuevas instrucciones, o basándose en su experiencia. El elemento más básico del cerebro humano es un tipo específico de célula, que nos provee de las capacidades de recordar, pensar, y aplicar experiencias anteriores a nuestras acciones. Estas células se conocen como neuronas y cada una de estas neuronas se pueden conectar con otras 200.000 neuronas. La potencia del cerebro viene de los números de estos componentes básicos y de las conexiones múltiples entre ellas.

Básicamente se trata de una red de elementos muy sencillos que trabajan en paralelo, existiendo una interconexión masiva entre ellos y diferentes grados de conexión

entre unos y otros. Las Redes Neuronales Artificiales fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de unidades llamadas “neuronas” o “nodos” conectadas unas con otras. Estas conexiones tienen una gran semejanza con las dendritas y axones de los sistemas nerviosos biológicos.

Las redes neuronales han sido utilizadas en áreas donde otros métodos han fallado, como por ejemplo el reconocimiento de patrones complicados, vagos, o incompletos [MP96], para el reconocimiento de minerales en una imagen digitalizada [SWP96]. También han sido utilizadas con éxito en el análisis de riesgos para conceder créditos a una persona determinada [Arr98], en el reconocimiento de caracteres cursivos, utilizada en actividades bancarias para el procesamiento de tarjetas de crédito y cheques, en donde la lectura y correcto reconocimiento de firmas en documentos tienen una significación crucial, agilizando tramites y reduciendo errores. De igual modo es utilizada en algunos aeropuertos de EE.UU. para detectar bombas.

La **Vida Artificial** [Jim99] es una disciplina que estudia la vida natural, recreando los fenómenos biológicos en ordenadores y otros medios artificiales. Complementa el estudio teórico de la biología pero en lugar de tomar organismos aislados y analizar su comportamiento, lo que se intenta es colocar juntos organismos que actúan como los seres vivos.

La Vida Artificial [Lie00] es una biología sintética en analogía con las síntesis químicas. El intento por recrear fenómenos biológicos en un medio alternativo nos llevará a una mejor comprensión de los procesos biológicos así como a la implementación de los principios biológicos en aplicaciones prácticas como en el diseño de ordenadores (*hardware* y *software*), robots móviles, medicina, nanotecnología, etc. Extendiendo los horizontes de la investigación empírica de la Biología más allá del territorio de la vida como la conocemos. La Vida Artificial nos da acceso al dominio de la vida como debería ser y es en este dominio en el que podremos construir teorías generales a cerca de la Biología y en el que descubriremos aplicaciones prácticas y útiles de las herramientas de los seres vivos en la resolución de problemas.

Las publicaciones realizadas sobre Vida Artificial pueden ser clasificadas en las siguientes áreas [Pao00]:

- Comportamiento adaptativo [Rut97].
- Comportamiento social [BDT99b].
- Biología evolucionaria [BLP97].
- Aprendizaje [Bat94].
- Robótica [LMP+98].
- Aplicaciones [CNP95].

El presente trabajo trata sobre una de las más recientes técnicas de Inteligencia Artificial en el área de Vida Artificial, conocida como **Sistema de Hormigas** *(Ant System)* [DMC92, DMC96], algoritmo inspirado en el comportamiento (a nivel de especie) de una colonia de hormigas que optimizan el camino para llegar desde el nido hasta su fuente de comida, utilizando para éste fin, comunicación indirecta por medio de una substancia química denominada feromonas y su capacidad para adaptarse a cambios de ambiente. Se tratará el tema con profundidad en los capítulos siguientes.

Todas estas técnicas si bien ofrecen buenos resultados, requieren de gran esfuerzo computacional. Esto implica que al crecer la complejidad del problema, el tiempo para resolverlos crece correspondientemente al punto de no ser factible la utilización de computadoras secuenciales. Así, se presenta el procesamiento paralelo como una opción para disminuir el tiempo de procesamiento de problemas de gran porte que hasta hoy no pueden ser resueltos con los recursos computacionales secuenciales accesibles en el país.

1.3 Procesamiento paralelo

El procesamiento paralelo y distribuido [BT89] es actualmente un área de intensa actividad en la investigación, motivada por una variedad de factores. Siempre ha existido la necesidad de encontrar solución a problemas de gran porte, pero los avances tecnológicos han permitido el procesamiento paralelo efectivo recién desde hace unos años. Además, la

disponibilidad de ordenadores paralelos de gran alcance está generando interés en nuevos tipos de problemas que no fueron tratados en el pasado. De esta manera, el desarrollo de los algoritmos distribuidos y paralelos está dirigido por esta interacción entre las viejas y nuevas necesidades de cálculo, por un lado, y por el progreso tecnológico por el otro.

El rápido aumento de la velocidad en las comunicaciones, en los últimos años, permite que el procesamiento paralelo se vuelva una opción cada vez más atractiva para la resolución de problemas que consumen mucho tiempo de procesamiento, dada la posibilidad actual de comunicar los procesadores de forma eficiente y barata.

Durante inicios de los 90 la computación paralela era exclusiva de los centros de alto desempeño que contaban con supercomputadoras de numerosos procesadores. Con los avances de la tecnología de redes y las computadoras personales, se generó un gran movimiento de investigación sobre el uso de redes locales en sustitución de las costosas supercomputadoras. Este nuevo esquema es utilizado no solo por países que carecen de grandes computadoras sino como una alternativa eficaz para el cómputo de alto rendimiento. Este salto tecnológico permite a naciones de poca infraestructura competir con grandes potencias de cálculo distribuido en redes ya operativas.

Un super-procesador diez veces más potente que un procesador de uso corriente es mucho más caro que diez procesadores corrientes. Por ello, si paralelizamos un programa, es decir, dividimos la carga computacional entre varios procesadores distintos, vamos a obtener una mejora en la relación entre costo/rendimiento. Con menos inversión en hardware estamos obteniendo más potencia computacional.

Avalon [Ava00] es un ejemplo de la potencia que se puede alcanzar con un sistema paralelo y distribuido. *Avalon* fue montada en Los Alamos National Laboratory y se encuentra entre las *top 500* máquinas más potentes del planeta [Top00]. Cuenta con 140 nodos, cada uno de ellos es un procesador Alpha DEC de 533 MHz trabajando bajo el sistema operativo Linux, obteniendo un rendimiento de 47,7 Gigaflops, con un costo aproximado de 313.000 dólares americanos. Para realizar una comparación, una

supercomputadora paralela con 64 procesadores de 250 MHz y 8 Gigabytes de memoria, *SGI Origin 2000* cuesta alrededor de 1.800.000 dólares americanos, alcanzando un rendimiento de 24 Gigaflops.

Existen varios factores, tanto de índole computacional como de índole puramente físico, que hacen difícil la obtención de un escalamiento en el rendimiento igual o superior al escalamiento en el número de procesadores, es decir, diez procesadores no van a trabajar diez veces más rápido que un solo procesador. Sin embargo, el incremento de velocidad se aproxima al número de procesadores que se integran en el sistema en determinados algoritmos con una paralelización intrínseca muy fuerte. Además, hay gran cantidad de científicos trabajando para acercar el límite práctico al límite teórico, por lo que, a los avances en hardware, hemos de sumar los avances que se producen en algoritmia.

A todo esto hay que añadir que el factor coste sigue siendo determinante para optar por soluciones paralelas aunque no sean óptimas, ya que, aunque no obtengamos el rendimiento máximo teórico, un incremento lineal en la potencia del procesador de una máquina implica un crecimiento exponencial del precio de una máquina, mientras que, con el incremento del número de procesadores, el incremento del precio es mucho menor.

Los sistemas paralelos son bastante complejos de programar, y su verificación es más compleja todavía. Por ello, estos sistemas sólo encuentran su justificación en aquellos casos en los que el incremento de costes en la programación realmente está justificado, porque esa potencia de cálculo es precisa. Para estas aplicaciones, sin embargo, los sistemas paralelos tienen gran éxito, encontrando como únicas barreras para su imposición final la complejidad del desarrollo, pues necesita de programadores realmente preparados, que conozcan a fondo materias tan complejas como mecanismos de sincronización y técnicas para compartir datos. Otro punto que vale la pena mencionar es la escasez de herramientas para hacer más cómoda la verificación de programas paralelos.

Cabe destacar la distinción entre sistemas paralelos y sistemas de cálculo distribuido [BT89]. En líneas generales, un sistema paralelo consiste en varios procesadores situados a

una pequeña distancia uno de otro. Su propósito principal es ejecutar conjuntamente una tarea diseñada para el efecto, la comunicación entre los procesadores es confiable.

Los sistemas distribuidos son diferentes en varias maneras. Los procesadores pueden estar separados a gran distancia unos de otros, y la comunicación entre los procesadores es más problemática. Los retardos de la comunicación pueden ser imprevisibles, y los puentes de comunicaciones no confiables. Además, la topología de un sistema distribuido puede experimentar cambios mientras que el sistema está funcionando, debido a incidentes o reparaciones de los puentes de comunicaciones, así como a la adición o al retiro de procesadores.

Algunos ejemplos de aplicaciones en las que el paralelismo es de gran utilidad son:

- los grandes sistemas gestores de bases de datos;
- operaciones de búsqueda y proceso de datos a gran escala, denominado *data minning*. Este problema tiene grandes implicaciones tanto en las ciencias de punta, como en el proyecto genoma humano, el análisis de datos estadísticos para problemas de bolsa y de análisis de mercados; es decir, mucho dinero en juego;
- algoritmos de cálculo numérico pesados. Este es un área fruto de gran investigación, existiendo gran cantidad de rutinas ya listas para ser empleadas en la bibliografía;
- los gráficos por ordenador a gran escala. Aunque este problema realmente es una subespecialización de los algoritmos de cálculo numérico pesados, problemas como generación de imágenes realistas o efectos especiales son muy pesados en cálculos;
- problemas de simulación. Estos problemas son de gran interés para la industria, ya que ahorran gran cantidad de dinero en experimentos de campo que no necesitan ser realizados por poder ser analizados en etapas de desarrollo del producto mediante complejas simulaciones por ordenador;
- predicción de fenómenos naturales, como una especialización del campo anterior de gran interés en la actualidad. Tanto fenómenos sísmicos, como

grandes fenómenos naturales (inundaciones, tornados, sequías) son centro de estudio en la actualidad, y una parte muy importante de los supercomputadores paralelos son emplados en la actualidad parcial o totalmente para este tipo de tareas;

- problemas de optimización. Si bien algunos métodos de optimización son bastante complejos de paralelizar, los problemas de optimización son lo suficientemente pesados computacionalmente como para que sea de interés su paralelización.

Además, existe una demanda creciente de computadores de elevado rendimiento en las áreas de análisis estructural, investigación de la energía de fusión, diagnóstico médico, simulaciones aerodinámicas, inteligencia artificial, sistemas expertos, automatización industrial, teledetección, defensa militar, ingeniería genética y socioeconomía, entre muchas otras aplicaciones científicas y técnicas. Sin computadores de gran rendimiento, muchos de estos retos al avance de la civilización humana no podrían llevarse a cabo dentro de un período razonable de tiempo [HB90].

La estrategia del procesamiento paralelo ha sido utilizada con éxito en nuestro país, atacando diversos problemas como:

- Optimización topológica de redes [BL99].
- Generación de energía eléctrica [BCC98].
- Flujo óptimo de potencia eléctrica [BCA+96].
- Partición de sistemas [BBR96].

1.4 Revisión Bibliográfica de la Colonia de Hormigas

Ant System (AS) es una innovadora técnica basada en agentes muy simples llamados hormigas, nació con la tesis doctoral de Marco Dorigo (1992) [Dor98] quien en 1996 [DMC96], publicó tres variantes del algoritmo que se diferencian en el momento y la manera de actualizar una matriz que representa las feromonas de los sistemas biológicos:

- *Ant-density*: con actualización constante de las feromonas por donde pasa una hormiga.
- *Ant-quantity*: con actualización de feromonas inversamente proporcional a la distancia entre 2 ciudades recorridas.
- *Ant-cycle*: con actualización de feromonas inversamente proporcional al trayecto completo, al terminar un recorrido. Este último presentó mejores resultados y las siguientes investigaciones se centraron en él.

Recientemente, Dorigo y Gambardella trabajaron en varias versiones extendidas del paradigma *AS*. *Ant-Q* [GD95, DG96] es un híbrido entre *AS* y *Q-learning*, un conocido algoritmo de aprendizaje con realimentación positiva [GD95, DG96]. *Ant Colony System (ACS)* es una extensión de *Ant-Q* publicado en 1997 [DG97], en el que se presentaron mejoras del algoritmo en tres aspectos principales: i) la regla de transición de estados, con la que se ofrece un balance entre la exploración de nuevos caminos y explotación a priori del conocimiento acumulado acerca del problema, ii) la regla de actualización global que permite actualizar la matriz de feromonas solo con el mejor tour encontrado hasta el momento iii) la regla de actualización local que permite a todas las hormigas actualizar la matriz de feromonas al terminar su tour. En dicha publicación se aplica además búsqueda local, utilizando el conocido método 3-opt [JM97] para el problema del cajero viajante, que intenta reducir la longitud encontrada intercambiando los arcos. En particular, este método realiza tres cortes en el tour encontrado e intercambia las ciudades destino, evitando de este modo invertir el sentido de las ciudades visitadas.

Otra variante de *AS*, conocida como *Max-Min Ant System (MMAS)*, fue presentada por Tomas Stützle y Holger Hoos [SH96], permitiendo actualizar la matriz de feromonas, solo a la hormiga con el mejor tour. Esto acelera la convergencia, pero puede llevar a estancamientos en soluciones sub-óptimas. A fin de evitar convergencias prematuras, se propuso poner un límite máximo y otro mínimo dentro de los cuales puede variar la cantidad de feromonas [SH96]. La aplicación de búsqueda local mejora notoriamente los resultados experimentales [SH97].

Como *Ant System* es una clase de algoritmo distribuido que consiste en un conjunto de agentes cooperativos que intercambian información de manera indirecta, el paralelismo es inherente al funcionamiento del algoritmo, es decir, el comportamiento de una hormiga es independiente de todas las demás durante la misma iteración. Se propusieron varias estrategias de paralelización. En [BKS97] fue publicada una implementación paralela síncrona y otra parcialmente asíncrona que se resumen a continuación.

a) En la implementación paralela síncrona, un proceso inicial (master) levanta a un conjunto de procesos hijos, uno para cada hormiga. Después de distribuir la información inicial acerca del problema, cada proceso inicia la construcción del camino y calcula la longitud del tour encontrado. Después de terminar este procedimiento, los resultados son enviados al master, quien se encarga de actualizar el nivel de feromonas y calcular el mejor tour encontrado hasta ahora. Se inicia una nueva iteración con el envío de la nueva matriz de feromonas.

b) En la implementación parcialmente asíncrona, se propone reducir la frecuencia de la comunicación. Para esto, cada hormiga realiza un cierto número de iteraciones del algoritmo secuencial, independientemente de las otras hormigas. Solo después de estas iteraciones locales, se realiza una sincronización global entre las hormigas. Entonces el master actualiza el nivel de feromonas y se inicia el proceso de nuevo.

Como una extensión de éste trabajo, Tomas Stützle presenta otra estrategia de paralelización [Stü98], propone corridas independientes y paralelas de un mismo algoritmo, *(ACO o MMAS)* evitando de éste modo el overhead de comunicación. Cabe destacar que en este modelo de paralelización no existe comunicación entre los procesos que corren en diferentes máquinas. El método funciona solo si el fundamento del algoritmo es aleatorio como en el caso de Ant System. La mejor solución de *k* corridas es elegida al final. En caso de aplicar búsqueda local, Stützle propone dos estrategias pero sin presentar resultados experimentales:

a) Una implementación síncrona al tener un proceso master que es utilizado para actualizar las estructuras de datos y construir soluciones iniciales que son enviadas a los procesadores hijos que se encargan de aplicar búsqueda local sobre ellos. El master recoge las soluciones óptimas locales y de encontrarse un número suficiente de tales soluciones se actualiza la matriz de feromonas antes de construir más soluciones.

b) Una implementación asíncrona en la que un procesador guarda la matriz de feromonas y la actualiza, mientras que uno o varios procesadores pueden usar la matriz de feromonas para construir soluciones y enviar a aquellos procesadores que aplicaran búsqueda local sobre éstos resultados.

Muchos investigadores interesados por la originalidad y el rendimiento del nuevo algoritmo, aplicaron la técnica con excelentes resultados a problemas tan diversos como:

- el paradigma del cajero viajante *(Traveling Salesman Problem)* [DMC96, DG97];
- el problema del ordenamiento secuencial *(Sequential Ordering Problem)* [GD97];
- el problema de ruteo de vehículos *(Vehicle Routing Problem)* [BHS99];
- el problema de asignación cuadrática *(Quadratic Assignment Problem)* [MC99];
- redes de telecomunicaciones *(Telecommunications Networks)* [CD98, BS00].

Esta técnica comienza a tener la madurez tecnológica adecuada para su utilización en problemas reales, como puede verse por la publicación de los libros [BDT99a] y [CDG99].

1.5 Objetivos y Organización del presente trabajo

El presente trabajo propone implementar *Ant System* en un ambiente totalmente asíncrono con una red de PCs, utilizando el problema simétrico del cajero viajante (TSP -

Traveling Salesman Problem) como paradigma de manera a probar el rendimiento del algoritmo propuesto.

Para esto, agentes computacionales muy simples llamados *hormigas* trabajan en cada procesador de una red de computadoras, explorando el espacio de soluciones, comunicando en forma asíncrona a los demás procesadores los resultados más alentadores, consolidando la información recogida en *matrices de feromonas* propias de cada procesador, que servirán para guiar la búsqueda de mejores soluciones.

El trabajo fue dividido en 6 Capítulos. El Capítulo 2 contiene un breve estudio bibliográfico del Problema del Cajero Viajante, y la descripción de problemas tipos a ser utilizados en el presente trabajo. La presentación de las diferentes variantes del algoritmo, con énfasis en las diferencias entre cada una de ellas se encuentra en el Capítulo 3. En el Capítulo 4 se presentan mejoras implementadas al algoritmo *Ant Quantity* y los resultados experimentales obtenidos. *Ant System* en su versión paralela y asíncrona se expone con detalles en el Capítulo 5, además de los resultados experimentales. Finalmente, en el Capítulo 6 se detallan a las conclusiones obtenidas con el presente trabajo, así como trabajos futuros.

El apéndice A contiene el artículo publicado en el marco de XXV Conferencia Latinoamericana de Informática, en Asunción – Paraguay. El apéndice B contiene el artículo publicado en el marco de la XXVI Conferencia Latinoamericana de Informática en México. El código fuente del programa utilizado para la variante *Ant Quantity* en su versión secuencial es finalmente presentada en el apéndice C.

Capítulo 2

El Problema del Cajero Viajante

2.1 Introducción

El Problema del Cajero Viajante *(TSP = Traveling Salesman Problem)* es uno de los problemas de optimización combinatoria más tradicionales y estudiados. Su variante más habitual, el TSP euclídeo y simétrico, consiste en una distribución de *N* ciudades en un plano bidimensional, para lo cual se define $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ como la distancia euclidiana entre la ciudad *i* y la ciudad *j*.

Cada ciudad es alcanzable desde cualquier otra ciudad mediante un camino cuya longitud viene definida por la distancia euclidiana que separa a ambas ciudades. Un cajero se dedica a recorrer estas ciudades comprando y vendiendo productos. Su interés es el de recorrer todas las ciudades empleando para ello el trayecto total más corto posible.

De modo más concreto, el problema consiste en hallar una ruta que, partiendo de una ciudad determinada, recorra todas las ciudades restantes, volviendo a la ciudad origen y minimizando la distancia total del recorrido.

Las restricciones que se han de tener en cuenta en este problema son las siguientes:

- cada ciudad sólo puede ser visitada una única vez en el recorrido;
- el recorrido debe pasar por todas las ciudades;
- el recorrido debe ser único y completo (no puede haber varios recorridos inconexos).

Pueden surgir otras restricciones debidas a la representación elegida. También existen muchas variantes del problema que introducen restricciones de todo tipo. Para el presente trabajo se contemplará únicamente la versión más habitual (euclídeo y simétrico).

Una característica recurrente en estos problemas de optimización combinatoria es el hecho de que son muy fáciles de entender y de enunciar, pero generalmente son difíciles de resolver. Podría pensarse que la solución de un problema de optimización combinatoria se restringe únicamente a buscar de manera exhaustiva el valor máximo o mínimo en un conjunto finito de posibilidades y que usando una computadora veloz, el problema carecería de interés matemático, sin pensar por un momento, en el tamaño de este conjunto, que hace inviable cualquier intento de búsqueda exhaustiva en problemas de tamaño no muy restringidos.

2.2 El Problema NP-Completo

Cada vez nos sorprendemos más de la velocidad de las computadoras actuales y de sus logros. Estos computadores han permitido la solución de problemas numéricos, tales como, la simulación de procesos gigantescos que involucran no solo aspectos físicos, sino también sociales, además de la manipulación de la información con cientos y miles de variables y restricciones y millones de datos que no podían ser manejados en forma manual hasta hace solo una década [Com1].

Sin embargo, pese a lo que pueda pensarse, las computadoras no realizan ninguna actividad que pudiera calificarse como de una gran inteligencia. Las computadoras pueden únicamente llevar a cabo algoritmos; esto es, secuencias de instrucciones precisas y universalmente entendibles que solucionen cualquier instancia de problemas computacionales rigurosamente definidos. Ejemplos típicos de algoritmos son los métodos que nos enseñan en las escuelas elementales para efectuar operaciones aritméticas sobre enteros decimales; estos son métodos precisos que pueden ser aplicados a cualquier entero, no importa que tan grande sea y, son métodos correctos en el sentido de que garantizan terminar con la solución correcta.

De esta forma, el concepto intuitivo de algoritmo, lo tenemos prácticamente todos y lo podemos entender entonces como sigue: “Un algoritmo es una serie de pasos para resolver un problema”.

Alan Turing demostró que existen problemas matemáticos bien definidos para los cuales no existe un algoritmo que los resuelva, o si existe, su tiempo de resolución crece exponencialmente, lo que elimina toda posibilidad de utilizarlo en problemas de dimensiones considerables.

El rendimiento de diferentes algoritmos, puede ser medido por el tiempo que el algoritmo en cuestión consume antes de producir la solución final. Esta cantidad de tiempo puede variar de una computadora a otra, debido a diferencias en su velocidad de procesamiento y en la forma como el programa es traducido en instrucciones de máquina. En el análisis de algoritmos, el tiempo siempre es expresado en términos del número de pasos elementales (operaciones aritméticas, comparaciones, lecturas, escrituras, etc.) requeridos para la ejecución del algoritmo sobre una computadora hipotética. Dichos pasos elementales tienen una duración unitaria de tiempo. De esta forma, el tiempo requerido para la ejecución de un algoritmo es establecido como el número de pasos elementales o instrucciones que se efectúa antes de producir la solución final.

El número de pasos requeridos por un algoritmo no es siempre el mismo para todas las entradas y frecuentemente es difícil obtener una fórmula precisa que establezca la función de tiempo. Es por ello que se consideran tres medidas, tomando en cuenta todas las entradas de un tamaño n dado [Com1]:

- el mínimo tiempo necesario para la ejecución del algoritmo para todas estas entradas (el tiempo del mejor caso);
- el máximo tiempo necesario (el tiempo del peor caso) y
- el tiempo promedio necesario (el tiempo del caso promedio).

De esta forma la complejidad de un algoritmo es función del tamaño de la entrada.

Los algoritmos polinomiales, cuya complejidad es descrita por una función polinomial, pueden ser ejecutados para entradas grandes en una cantidad de tiempo razonable. Por su parte, los algoritmos exponenciales son aquellos que violan todo acotamiento polinomial para instancias suficientemente grandes. Se les da este nombre porque 2^n es el paradigma de una razón de crecimiento no polinomial. Lo anterior tiene como base la teoría de **NP-Completo** presentada por Cook en 1971. Cook probó que cualquier problema que perteneciera a la clase NP podía ser reducido al problema de satisfactibilidad proposicional a través de una transformación de tipo polinomial. Cook también sugirió una división entre los problemas más difíciles: **NP-Completos** para problemas de decisión y **NP-Duros** para problemas de optimización o búsqueda

El TSP es considerado representativo de problemas cuya resolución pertenece al dominio **NP-Completo**, de extrema complejidad. Esto se debe a que no se conoce ningún algoritmo capaz de generar la solución óptima sin recurrir a la enumeración sistemática de soluciones.

Para un problema de tamaño *N* (número de ciudades) la cantidad de soluciones que se han de explorar viene descrita por un factorial de ***N***. Lo que esto implica se puede apreciar fácilmente en la Tabla 1 que confronta diversos tamaños de problemas con el tamaño del espacio de búsqueda y con el tiempo que requeriría un ordenador capaz de procesar 1 millón de soluciones por segundo. Se produce una explosión combinatoria tal, que resulta impensable obtener una solución óptima a partir de *N*=30. En la Figura 1 se puede observar una gráfica de la complejidad del problema.

Ciudades	Soluciones	Tiempo	Unidad
4	3	0,000003	segundos
5	12	0,000012	segundos
6	60	0,00006	segundos

7	360	0,00036	segundos
8	2520	0,00252	segundos
9	20160	0,02016	segundos
10	181440	0,18144	segundos
11	1814400	1,8144	segundos
12	19958400	19,9584	segundos
13	239500800	3,99168	minutos
14	3113510400	51,89184	minutos
15	4,3589E+10	12,108096	horas
16	6,5384E+11	7,56756	días
17	1,0461E+13	121,08096	días
18	1,7784E+14	5,6393872	años
19	3,2012E+15	101,508969	años
20	6,0823E+16	19,2867041	siglos
21	1,2165E+18	385,734083	siglos
22	2,5545E+19	8,1004157	milenios
23	5,62E+20	178,20914	milenios
24	1,2926E+22	4098,8103	milenios
25	3,1022E+23	98371,4488	milenios
26	7,7556E+24	2459286,21	milenios
27	2,0165E+26	63941441,7	milenios
28	5,4444E+27	1726418926	milenios
29	1,5244E+29	48339729930	milenios
30	4,4209E+30	1,4019E+12	milenios
31	1,3263E+32	4,2056E+13	milenios
32	4,1114E+33	1,3037E+15	milenios
33	1,3157E+35	4,1719E+16	milenios

Tabla 1: Espacio de búsqueda y tiempo requerido

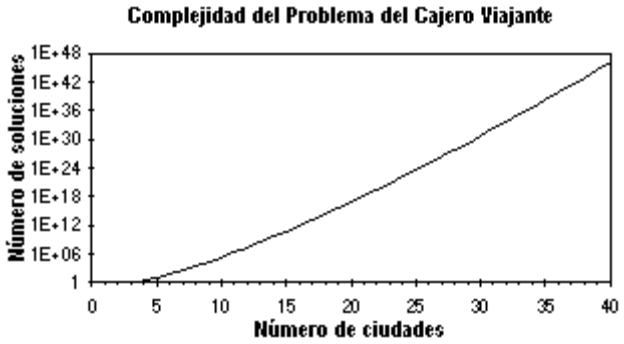


Figura 1: Complejidad del Problema

2.4 Revisión Bibliográfica

El Problema del Cajero Viajante ha sido considerado un paradigma desde hace muchísimos años. Científicos de todas partes han ocupado su tiempo en resolver diferentes instancias del problema.

A continuación se presenta algunos de los trabajos publicados con la aplicación de diferentes técnicas para atacar el problema.

El trabajo de Johnson et al. [JM97] presenta un estudio sobre la optimización local aplicada al problema del cajero viajante. Trata dos métodos específicos, 2-opt y 3-opt que intentan mejorar un tour específico realizando simples movimientos para convertir un tour en otro, esto es, dado un tour factible, el algoritmo realiza repetidas operaciones de una clase, hasta reducir la longitud de tour actual o hasta que se consiga un tour para el cual ya no exista operaciones posibles que la mejoren. Alternativamente, se puede ver esto como un proceso de búsqueda en la vencindad (*neighborhood search*), donde cada tour tiene una vecindad asociada de tours adyacentes, esto significa que pueden ser encontrados con un solo movimiento y esto se realiza hasta encontrar uno mejor o hasta que ya no existan mejores.

El algoritmo 2-opt fue propuesto por primera vez por Croes (1958), aunque los movimientos básicos ya fueron sugeridos por Flood (1956). Este método elimina dos arcos, rompiendo el tour en dos caminos diferentes, que luego son reconectados de otra manera posible, (Figura 2). Nótese que la figura es sólo un ejemplo, si las distancias fueran como aparecen en ella, estos cambios podrían ser contraproducentes y no se llevarían a cabo. En 3-opt, propuesto por Bock (1958), los cambios reemplazan tres arcos del tour actual (Figura 3). Además se realizó un estudio detallado de los métodos mencionados aplicados a técnicas como *Tabu Search, Simulated Annealing, Genetic Algorithms y Neural Networks*.

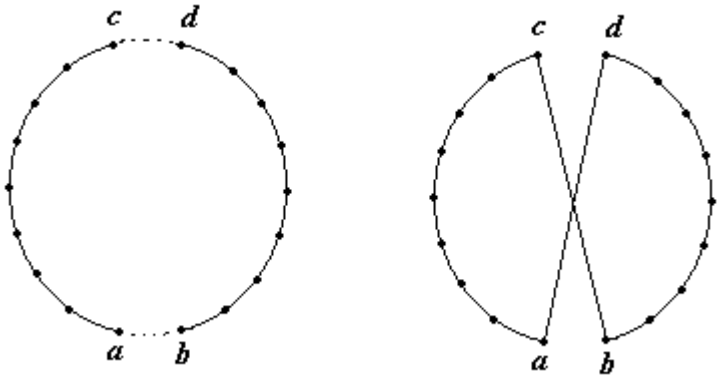


Figura 2: Un movimiento 2-opt
Tour original a la izquierda y el Tour resultante a la derecha

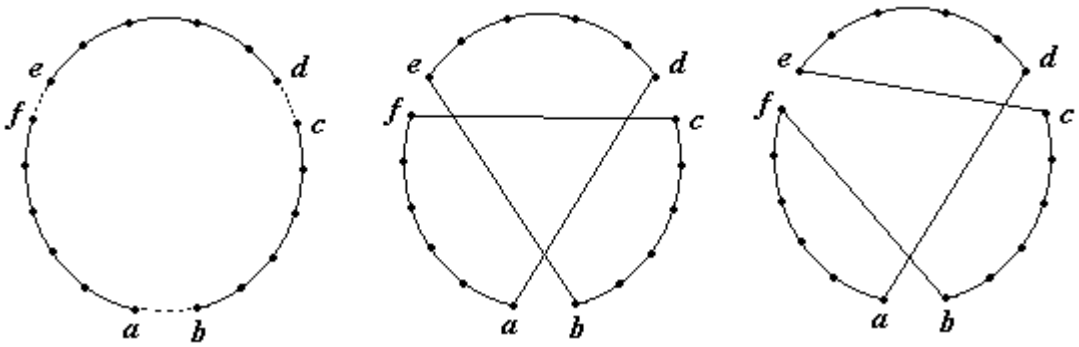


Figura 3: Dos movimientos 3-opt posibles
Tour original a la izquierda y los Tours resultados a la derecha

Tabu Search, como *Simulated Annealing* y varias técnicas discutidas en el trabajo de Johnson et al. [JM97] son motivadas por la observación de que no todas las soluciones locales necesitan ser buenas soluciones. De ésta manera, se desea modificar un algoritmo de optimización puramente local proveyéndole de un mecanismo que lo ayude a escapar de su óptimo local y continuar la búsqueda más allá de ésta. Uno de esos mecanismos podría ser simplemente realizar repetidas corridas de un algoritmo de optimización local, usando una heurística de inicialización aleatoria para proveer diferentes soluciones iniciales.

Una razón para la limitada efectividad de la política de reinicio al azar es que no explota la posibilidad de que la solución óptima pueda estar muy cercana a la solución actual. Si esto fuera verdad, sería mejor reiniciar la búsqueda cerca de la solución encontrada en lugar de realizar una elección aleatoria. Esto es en esencia lo que *tabu search* realiza. Asumiendo que todos los vecinos de la solución actual son examinados a cada paso, *tabu search* alterna entre buscar un óptimo local y, una vez hallado, identificar la mejor solución vecina, la cual es utilizada como punto de inicio para una nueva fase de optimización local. Información de los más recientes movimientos realizados es guardada en una o más listas tabú, y dicha información es utilizada para descalificar nuevos movimientos que revertirían el trabajo de esos movimientos recientes.

Tabu Search se cimenta en varios factores: i) el nivel de aspiración, el cual provee excepciones a la regla tabú, típicamente en situaciones en donde exista alguna garantía que el movimiento supuestamente prohibido no nos regrese a una solución ya vista con anterioridad, ii) la regla de diversificación, que provee algo parecido al reinicio aleatorio, iii) la regla de intensificación, el cual restringe a que la búsqueda se mantenga en la vecindad de una buena solución previamente encontrada

Dada la flexibilidad inherente al método de *tabu search*, no es sorpresa que exista una amplia variedad de dichos algoritmos que han sido propuestos para el TSP. El primer algoritmo *tabu search* implementado para TSP fue descripto por Glover (1986). Resultados limitados para esta implementación y sus variantes fueron reportados por Glover (1989), Knox y Glover (1989), Knox (1989, 1994), y métodos similares fueron estudiados por Troyon (1988) y Malek, Guyruswamy, y Pandya (1989). Estos algoritmos usan 2-opt intercambios como movimientos básicos, pero difieren en la naturaleza de la lista tabú y la implementación de los niveles del criterio de aspiración.

El TSP fue uno de los primeros problemas para el cual *Simulated Annealing* fue aplicado [JM97], sirve de ejemplo para Kirkpatrick et al. (1983) y Cerny (1985). La mayoría de las adaptaciones han sido basadas en el esquema simple presentado en el Pseudocódigo 1, con implementaciones diferentes en sus métodos para generar soluciones

iniciales (tours) y para manipular temperaturas, así como en la definición de *equilibrio, enfriamiento, vecindad* y *aleatoriedad*.

Pseudocódigo 1: Esquema general para un algoritmo *simulated annealing*

1. Generar una solución inicial *S* y un conjunto de soluciones iniciales *S**=*S*.
2. Determinar la temperatura inicial *T*.
3. Mientras no se alcance *enfriamiento* hacer:
   3.1. Mientras no alcance *equilibrio* para esta temperatura, hacer:
      3.1.1. Elegir con *aleatoriedad* una *vecindad S'* de la solución actual.
      3.1.2. Fijar $\Delta = Length(S') - Length(S)$.
      3.1.3. Si $\Delta \leq 0$ (movimiento cuesta abajo):
         Fijar *S* = *S'*
         Si *Length(S)* < *Length(S*)*, fijar *S** = *S*.
      3.1.4. Sino (movimiento cuesta arriba):
         Elegir un número aleatorio *r* uniformemente distribuido entre [0,1].
         Si $r < e^{-\Delta/T}$, fijar *S* = *S'*.
      3.1.5. Fin "Mientras no alcance *equilibrio*".
   3.2. Bajar la temperatura *T*.
   3.3. Fin "Mientras no alcance *enfriamiento*".
4. Retornar *S**.

En la adaptación del *simulated annealing* para el TSP, Kirkpatrick et al. (1983) [JM97] y Cerny (1985) [JM97] sugirieron utilizar una estructura de vecindades basada en movimientos 2-opt. Cerny (1985) también consideró el movimiento más simple en el cual las posiciones de dos ciudades son intercambiadas quedando el segmento entre ellas sin alteraciones, pero los experimentos demostraron que no era un método efectivo.

El número de pasos a cada temperatura (al cual se denomina longitud de temperatura) necesita ser al menos proporcional al tamaño de la vecindad para obtener un tour de calidad considerable. Esto no es una restricción muy onerosa para problemas como particiones de grafos, donde el tamaño típico de la vecindad es O(N) (Kirkpatrick et al., 1983) [JM97]. Para la vecindad del TSP 2-opt el tamaño es proporcional a N^2, de modo que incluso si el número de temperaturas distintas consideradas no crece con N, todavía se cuenta con un algoritmo cuyo tiempo de ejecución es al menos $\Theta(N^2)$ con una gran constante de proporcionalidad.

El uso de *Genetic Algorithms* como método de optimización se inició a principios de 1970 [JM97]. La mejor adaptación de este método para el TSP es el esquema presentado en el Pseudocódigo 2, donde cada ejecución del ciclo consistente en los pasos 3.1 al 3.5 puede ser vista como el paso de una sola generación en el proceso de la evolución. Se puede observar que las operaciones en diferentes soluciones pueden ser realizadas de manera paralela si fuera deseado, por este motivo, muchas veces son conocidos como "algoritmos genéticos paralelos".

Para una adaptación específica del esquema al TSP se necesitan declarar *k* y *k'*, además del método para generar soluciones iniciales (tours), el algoritmo de optimización local *A*, la estrategia de emparejamiento, los operadores naturales de cruzamiento y mutación, la estrategia de selección, y el criterio de convergencia.

Se debe notar que en el esquema presentado en el Pseudocódigo 2 no es lo que se definió como un "algoritmo genético" en las primeras publicaciones sobre el tema como en Holland (1975) [JM97]. En particular, la aplicación de optimización local a las soluciones individuales en los pasos 2 y 3.4 podría ser vista como una acción poco ortodoxa. Al contexto de la motivación biológica original para el método genético, se incorpora el principio de Lamarckian en el que se pueden heredar rasgos sabios. No obstante, tales pasos de optimización local parecen ser esenciales si los resultados a ser obtenidos para el TSP son competitivos.

Pseudocódigo 2: Esquema general para un algoritmo genético de optimización

1. Generar una población $\mathbf{S}$ de *k* soluciones iniciales: $\mathbf{S} = \{S_1, \cdots, S_k\}$.
2. Aplicar el algoritmo *A* de optimización local a cada solución *S* en $\mathbf{S}$, permitiendo que la solución óptima local encontrada reemplace *S* en $\mathbf{S}$.
3. Mientras no haya convergencia, hacer:
   3.1. Seleccionar k' subconjuntos distintos de $\mathbf{S}$, de tamaño 1 o 2, como padres (estrategia de emparejamiento).
   3.2. Para cada subconjunto de 1 elemento, realizar una operación de mutación aleatoria para obtener nuevas soluciones.
   3.3. Para cada subconjunto de 2 elementos, realizar una operación de cruzamiento (posiblemente aleatoria) para obtener una nueva solución que refleje aspectos de ambos padres.
   3.4. Aplicar el algoritmo A de optimización local a cada una de las k' soluciones.
   3.5. Usar una estrategia de selección, elegir *k* sobrevivientes de $\mathbf{S} \cup \mathbf{S}'$, y reemplazar el contenido de $\mathbf{S}$ por los sobrevivientes.
4. Retornar la mejor solución en $\mathbf{S}$.

El primer algoritmo para TSP que se ajusta al esquema presentado en el Pseudocódigo 2 fue presentado por Brady (1985) [JM97]. Brady utilizó 2-opt para optimización local y restringió el emparejamiento (esto es, todos los hijos tienen 2 padres). Para su estrategia de emparejamiento, el identificó padres realizando un emparejamiento al azar entre los elementos de $\mathbf{S}$ (esto implica $k'=k/2$). En el operador de cruzamiento, examinó los tours padres hasta encontrar un par de sub-tours comunes, uno de cada tour padre, que contenían el mismo conjunto de ciudades, pero en un orden diferente. El hijo era obtenido eliminando el sub-tour más largo del tour que lo contenía y era reemplazado por su contraparte más corta.

En sus resultados experimentales, Brady observó que al aumentar *k* los resultados obtenidos mejoraban. Para una instancia de 64 ciudades, obtuvo resultados significativamente mejores.

El mayor salto en la ejecución de algoritmos genéticos vino con el trabajo de Mühlenbein, Gorges-Schleuter, y Krämer (1988) [JM97]. Su algoritmo fue diseñado para iniciar la búsqueda a partir de una implementación paralela, e introduce una estrategia de emparejamiento más sofisticada, además de un operador de cruzamiento mejor. Un tour fue asignado a cada procesador, y cada ciclo empareja su tour (el procesador receptor) con tours de otros procesadores (emisores). Cada tour recibido fue elegido aleatoriamente de una colección de cuatro tours en los procesadores vecinos (su máquina paralela utiliza una grilla para interconectar a sus procesadores), junto con el mejor tour global, se impone una tendencia a favor del tour más corto. Para la operación de cruzamiento se procedió a elegir aleatoriamente sub-tours del procesador emisor de longitud entre 10 y N/2 para luego extendeser al tour completo agregando sucesivas ciudades.

Mühlenbein et al.[JM97] también introdujeron una idea sobre la aceleración de la optimización local de la descendencia: para cada tour descendiente, ellos identificaron todos los sub-tours de cuatro o más ciudades que se presentaban en ambos padres y sellaban todos los arcos que unen a dichas ciudades prohibiendo cualquier movimiento 2-opt que los elimine.

La primera aplicación de *Neural Networks* al TSP [JM97] apareció en el trabajo de Hopfield and Tank (1985). Su método se basaba en una programación de enteros para el TSP descripta en la Figura 4. Aquí $x_{ik} = 1$ significa que la ciudad c_i es la *kth* ciudad en el tour, en dicho caso la suma estaría minimizando la longitud del tour. La primera restricción dice que cada posición contiene precisamente una ciudad y la segunda dice que cada ciudad está en una posición precisa.

El algoritmo de Hopfiel y Tank intentó encontrar una solución factible a este programa de enteros, cada x_{ij} es representado por una neurona que puede tomar valores

arbitrarios en el intervalo [0,1]. Estas neuronas fueron interconectadas con una red inhibitoria que intentó simultáneamente imponer las restricciones y bajar el costo de la función de energía sustituta. Óptimo local para esta función de energía fue aplicado utilizando un algoritmo de optimización local donde neuronas individuales cambiaban de estado con el objeto de bajar su contribución a la energía total. En el contexto de los otros métodos presentados en el trabajo, los resultados de Hopfield y Tank no fueron prometedores.

$$\text{Minimizar} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} d(c_i, c_j)\cdot\left[x_{i,N}\cdot x_{j,1} + \sum_{k=1}^{N-1} x_{i,k}\cdot x_{j,k+1}\right]$$

$$\text{Sujeto a} \quad \sum_{i=1}^{N} x_{ik} = 1, \qquad 1 \le k \le N$$

$$\text{y} \quad \sum_{k=1}^{N} x_{ik} = 1, \qquad 1 \le i \le N$$

$$\text{donde} \quad x_{ik} \in [0,1], \qquad 1 \le i,k \le N$$

Figura 4: Formulación de una programación de enteros para el TSP

Stützle et al. [SH99] presentaron un análisis del comportamiento del tiempo de ejecución de iterativas búsquedas locales *(Iterated Local Search* ILS) para problemas de optimización. ILS es un simple y poderoso algoritmo metaheurístico, el cual ha demostrado ser uno de los mejores entre los algoritmos de aproximación. Está basado en la observación de que las sucesivas mejoras alcanzadas utilizando búsqueda local son atrapadas fácilmente por un mínimo local. En lugar de reiniciar la búsqueda local a partir de una nueva solución, generada aleatoriamente, es una mejor idea modificar la solución actual ***s***, moviendo el punto de corte ***s'*** más allá de la vecindad buscada por el algoritmo y reinicar la búsqueda a partir del punto ***s'*** hasta alcanzar un nuevo mínimo local ***s''***. Un

criterio de aceptación determina si la búsqueda debe continuar a partir de ***s'*** o ***s''*** (o posiblemente a partir de otra solución).

Se implementaron dos algoritmos de búsqueda local, 2-opt y 3-opt con ILS. Para modificar la solución inicial fue utilizado el movimiento llamado doble-puente (*double-bridge*), el cual es una elección estándar para ILS aplicados al problema del cajero viajante. El mismo realiza cortes al tour actual en 4 arcos apropiadamente elegidos, el resultado de esta operación son 4 sub-tours $s_1 - s_2 - s_3 - s_4$ los cuales están contenidos en la solución en el orden dado. Estos sub-tours son reconectados en el orden $s_4 - s_3 - s_2 - s_1$ para producir una nueva solución e iniciar la búsqueda local.

El movimiento doble-puente es un movimiento 4-opt específico realizado de tal manera a no cambiar la dirección del tour. En los experimentos preliminares fue encontrado que se puede alcanzar un mejor desempeño si los arcos a ser cortados por el doble-puente son elegidos de una manera no completamente aleatoria. En lugar de esto, se procedió de la siguiente manera: en I2-opt y I3-opt fueron elegidos aleatoriamente los arcos *(i,j)*.

El criterio de aceptación se denota como *Better(**s**, **s''**)* y retorna ***s''*** si el tour ***s''*** es más corto que ***s***, de otra manera retorna ***s***. Las soluciones iniciales para todos los algoritmos ILS aplicados fueron generados por la heurística del vecino más cercano *(nearest neighbor*).

Stützle et al. [SGL$^+$00] presenta un interesante trabajo comparando métodos heurísticos inspirados en la naturaleza para resolver el problema del cajero viajante. El TSP ha jugado un papel importante en el desarrollo de muchas heurísticas inspiradas en la naturaleza tales como *Evolutionary Computation*, *Ant Colony Optimization*, *Neural Networks* o *Simulated Annealing*, solo por nombrar algunos. El estado presente del arte en TSP sugiere que se obtienen mejores resultados combinando algoritmos de construcción o modificación con la aplicación de efectivos y rápidos algoritmos de búsqueda local. Cuando se comparan algoritmos para los TSP, es particularmente importante que todos los algoritmos utilicen el mismo método de búsqueda local, porque su elección tiene una

decisiva influencia en el rendimiento final. Además, pequeñas diferencias en la implementación de la búsqueda local pueden causar significativas diferencias en el rendimiento de estos algoritmos híbridos.

Para las comparaciones experimentales Stützle et al. [SGL+00] eligieron 4 algoritmos que demostraron buen rendimiento y han sido reimplementados con el algoritmo 3-opt. En particular, se implementaron dos Algoritmos Genéticos (GA), uno análogo al GA desarrollado por Merz y Freisleben y otro similar al implementado por Walters, *un Ant Colony Optimization* (ACO) implementación de Stützle y Hoos y un *algoritmo Iterated Local Search* (ILS).

El GA implementado por Merz y Freisleben (a continuación GA-DPX) utiliza un operador específico de cruzamiento llamado DPX el cual intenta generar una descendencia que tenga igual distancia entre ambos padres. DPX trabaja como sigue: el contenido del primer padre es copiado a la descendencia y todos los arcos que no tienen en común en los 2 padres son eliminados. Las partes resultantes del tour roto son reconectados usando una heurística (*greedy heuristic*). Para la mutación en GA-DPX se utilizó el movimiento del doble-puente (double-bridge - DB). La mutación DB corta el tour actual en 4 sub-tours $s_1 - s_2 - s_3 - s_4$ (los cuales están contenidos en la solución, en el orden dado) y los reconecta en el orden $s_4 - s_3 - s_2 - s_1$. Se utiliza el método de búsqueda local 3-opt para mejorar las soluciones.

Los GA implementados por Walters [SGL+00] difieren substancialmente de los GA-DPX porque en los GA de Walters los operadores de mutación y cruzamiento puede generar tours no factibles, los cuales son reparados por un ingenioso mecanismo; que se llamará *GA-Repair*. La idea central de reparar el algoritmo es reemplazar los arcos no factibles por arcos factibles cuya longitud sea lo más cercana posible a la longitud del arco original no factible. Otra diferencia es que *GA-Repair* utiliza una población relativamente grande. Walters utiliza un mecanismo de selección de hijos, el cual genera varios hijos por un par de padres y se elige sólo uno o dos de las mejores descendencias. Stützle et al. [SGL$^+$00] no implementaron exactamente este mecanismo, pero con un espíritu similar

generaron un gran número de hijos (tres veces el tamaño de la población *m*, para sus experimentos *m*=100) en cada generación, y seleccionaron padres utilizando un *fitness* basado en rangos.

Otro algoritmo implementado fue *Ant Colony Optimization*, se utilizó la variante *MAX-MIN Ant System* (MMAS) implementado por Stützle y Hoos, que ha obtenido uno de los mejores resultados para muchos problemas TSP.

En *Iterate Local Search* (ILS) se aplica un mecanismo sofisticado, que se denomina *Fitness-Distance-Based-Diversification* (ILS-FDD) para diversificar la búsqueda. ILS-FDD genera una nueva solución inicial para ILS de una manera iterativa, en particular, se obtiene una solución por la generación de un conjunto de soluciones candidatas por el mecanismo estándar ILS y luego es elegida una solución candidata, cuya calidad depende de la distancia de la solución original. Este proceso es repetido hasta que la solución alcance una distancia mínima de la solución original.

La implementación de 3-opt utiliza técnicas estándares de aceleración: i) restringe el conjunto de movimientos examinados a aquellos contenidos en la lista de los 40 vecinos más cercanos; ii) realiza la búsqueda en la vecindad de un punto fijo; iii) utiliza una bandera de bits asociada a cada nodo.

Los resultados experimentales demostraron que para muchas instancias del TSP, ILS-FDD presenta un mejor rendimiento, lo cual se observa al obtener mayores probabilidades de encontrar la solución óptima para el mismo tiempo de ejecución que sus competidores, y para algunas instancias (`att532` y `d1291`) es el único algoritmo que siempre encuentra la solución óptima en un tiempo de ejecución dado. En general, la clasificación jerárquica de los algoritmos depende fuertemente de la instancia. Por ejemplo, *GA-Repair* es el algoritmo que presentó el rendimiento más bajo en la instancia `pcb442`, pero entre los mejores rendimientos en las instancias `rat783` y `pr1002`.

2.4 Problemas utilizados

Las secuencias de coordenadas de los problemas utilizados para el presente trabajo se citan a continuación.

La matriz de distancias para 15 ciudades del Paraguay que se cita a continuación, fue utilizada para probar el algoritmo *Ant Quantity*:

$$\begin{bmatrix}
0 & 102 & 104 & 201 & 168 & 208 & 225 & 262 & 252 & 322 & 316 & 259 & 401 & 494 & 467 \\
102 & 0 & 141 & 257 & 156 & 126 & 131 & 198 & 245 & 379 & 406 & 356 & 500 & 584 & 551 \\
104 & 141 & 0 & 144 & 210 & 265 & 203 & 195 & 146 & 235 & 277 & 274 & 403 & 465 & 418 \\
201 & 257 & 144 & 0 & 301 & 371 & 348 & 338 & 238 & 193 & 156 & 136 & 264 & 329 & 332 \\
168 & 156 & 210 & 301 & 0 & 180 & 287 & 358 & 376 & 445 & 400 & 309 & 448 & 564 & 561 \\
208 & 126 & 265 & 371 & 180 & 0 & 173 & 275 & 363 & 497 & 524 & 455 & 601 & 725 & 665 \\
225 & 131 & 203 & 348 & 287 & 173 & 0 & 102 & 232 & 408 & 532 & 467 & 608 & 668 & 579 \\
262 & 198 & 195 & 338 & 358 & 275 & 102 & 0 & 153 & 346 & 477 & 470 & 598 & 621 & 568 \\
252 & 245 & 146 & 238 & 376 & 363 & 232 & 153 & 0 & 193 & 353 & 374 & 482 & 497 & 421 \\
322 & 379 & 235 & 193 & 445 & 497 & 408 & 346 & 193 & 0 & 208 & 285 & 348 & 321 & 232 \\
316 & 406 & 277 & 156 & 400 & 524 & 532 & 477 & 353 & 208 & 0 & 121 & 139 & 175 & 163 \\
259 & 356 & 274 & 136 & 309 & 455 & 467 & 470 & 374 & 285 & 121 & 0 & 143 & 255 & 270 \\
401 & 500 & 403 & 264 & 448 & 601 & 608 & 598 & 482 & 348 & 139 & 143 & 0 & 153 & 225 \\
494 & 584 & 465 & 329 & 567 & 725 & 668 & 621 & 497 & 321 & 175 & 255 & 153 & 0 & 113 \\
467 & 551 & 418 & 332 & 561 & 665 & 579 & 568 & 421 & 232 & 163 & 270 & 225 & 113 & 0
\end{bmatrix}$$

La solución óptima para este problema es de 2195.

Las siguientes coordenadas (x,y) pertenecen al problema Oliver’30 [WSF89] para 30 ciudades:

{ (54,67) (54,62) (37,84) (41,94) (2,99) (7,64) (25,62) (22,60) (18,54) (4,50) (13,40) (18,40) (24,42) (25,38) (44,35) (41,26) (45,21) (58,35) (62,32) (82,7) (91,38) (83,46) (71,44) (64,60) (68,58) (83,69) (87,76) (74,78) (71,71) (58,69) }

El mejor resultado conocido en la literatura para este problema es de 423,74.

Las siguientes coordenadas (x,y) pertenecen al problema Eilon'50 [WSF89] para 50 ciudades:

{ (37,69) (27,68) (31,62) (42,57) (37,52) (38,46) (42,41) (45,35) (40,30) (32,22) (27,23) (20,26) (17,33) (25,32) (31,32) (32,39) (30,48) (21,47) (25,55) (16,57) (17,63) (5,64) (8,52) (12,42) (7,38) (5,25) (10,17) (5,6) (13,13) (21,10) (30,15) (36,16) (39,10) (46,10) (59,15) (51,21) (58,27) (48,28) (52,33) (52,41) (56,37) (61,33) (62,42) (58,48) (49,49) (57,58) (62,63) (63,69) (52,64) (43,67) }

El mejor resultado publicado en la literatura para este trabajo es de 427,86.

Las siguientes coordenadas (x,y) pertenecen al problema KroA100 [Rei00] para 100 ciudades:

{ (1380,939) (2848,96) (3510,1671) (457,334) (3888,666) (984,965) (2721,1482) (1286,525) (2716,1432) (738,1325) (1251,1832) (2728,1698) (3815,169) (3683,1533) (1247,1945) (123,862) (1234,1946) (252,1240) (611,673) (2576,1676) (928,1700) (53,857) (1807,1711) (274,1420) (2574,946) (178,24) (2678,1825) (1795,962) (3384,1498) (3520,1079) (1256,61) (1424,1728) (3913,192) (3085,1528) (2573,1969) (463,1670) (3875,598) (298,1513) (3479,821) (2542,236) (3955,1743) (1323,280) (3447,1830) (2936,337) (1621,1830) (3373,1646) (1393,1368) (3874,1318) (938,955) (3022,474) (2482,1183) (3854,923) (376,825) (2519,135) (2945,1622) (953,268) (2628,1479) (2097,981) (890,1846) (2139,1806) (2421,1007) (2290,1810) (1115,1052) (2588,302) (327,265) (241,341) (1917,687) (2991,792) (2573,599) (19,674) (3911,1673) (872,1559) (2863,558) (929,1766) (839,620) (3893,102) (2178,1619) (3822,899) (378,1048) (1178,100) (2599,901) (3416,143) (2961,1605) (611,1384) (3113,885) (2597,1830) (2586,1286) (161,906) (1429,134) (742,1025) (1625,1651) (1187,706) (1787,1009) (22,987) (3640,43) (3756,882) (776,392) (1724,1642) (198,1810) (3950,1558) }

El mejor resultado publicado en la literatura para este problema es de 21.285,44.

La función utilizada para la construcción de las matrices de distancias a partir de las coordenadas (x,y) es la siguiente:

Pseudocódigo 3: Función para construir matriz de distancia

```
ConstruirMatriz(x, y, d)
int x [LONGITUD], y[LONGITUD];
int d[LONGITUD][LONGITUD];
{
   int primero, segundo;
   for(primero=0; primero<LONGITUD, primero++)
   {
      d[primero][primero]=0.0;
      for(segundo=0; segundo<primero; segundo++)
         {
            d[primero][segundo] =
            (int) ((sqrt((double) (DIFFSQ(x[primero], x[segundo]) +
                  (DIFFSQ(y[primero],y[segundo])))))) + 0,5);
            d[segundo][primero]=d[primero][segundo];
         }
   }
 }
```

Capítulo 3

Ant System

3.1 Introducción

Ant System (AS) se basa en el comportamiento colectivo de las hormigas en la búsqueda de alimentos para subsistir. Resulta fascinante entender como animales casi ciegos, moviéndose aproximadamente al azar, pueden encontrar el camino más corto desde su nido hasta la fuente de alimentos y regresar. Para esto, cuando una hormiga se mueve, deja una señal odorífera, depositando una substancia denominada feromona, para que las demás puedan seguirla.

En la naturaleza, las feromonas cumplen un importante papel en la organización y supervivencia de muchas especies. Así, representa un sistema de comunicación química entre animales de una misma especie, que transmiten información a través de señales odoríferas acerca del estado fisiológico, reproductivo y social, así como la edad, sexo y parentesco del animal emisor, las cuales son recibidas en el sistema olfatorio del animal receptor, quien interpreta esas señales.

Estas señales odoríferas pueden ser liberadas de una manera activa cuando el animal realiza ciertas conductas específicas como la de marcaje por frotamiento del mentón en conejos o por frotamiento de diversas partes del cuerpo en los gatos. También la liberación de las feromonas puede ser pasiva cuando el animal emisor no

participa activamente como en el caso de la emisión de feromonas en los pezones de conejas para que sus crías que nacen ciegas se guíen hacia ellos y puedan alimentarse.

En muchos mamíferos, los machos marcan el territorio en donde pueden alimentarse y aparearse con las hembras que se encuentren en esa demarcación. A su vez, las hembras comunican su disponibilidad sexual.

Las feromonas, han sido más estudiadas en las llamadas especies inferiores como son las hormigas, las abejas y otros insectos. De hecho, los primeros estudios se realizaron en una polilla y el comportamiento asociado a este sistema de comunicación química. Asimismo, la palomilla del gusano de seda fue el primer animal en el que se estudió el sofisticado sistema olfativo para detectar las feromonas.

Existen feromonas que son volátiles y que cuentan con poca estabilidad y persistencia, pero que tienen una mayor dispersión y un tiempo corto de acción. Tal es el caso de las feromonas secretadas por las hormigas o termitas durante un ataque a su nido. Además cuando una hormiga experimenta una sensación, la emite por todo su cuerpo y todas las hormigas de los alrededores la perciben al mismo tiempo que ella. Una hormiga estresada comunica al instante su pena al entorno, de suerte que las demás hormigas solo tienen una preocupación: que cese el penoso mensaje encontrando un método para ayudar al individuo.

Asimismo, existe otro tipo de feromonas que tienen poca volatilidad y una mayor estabilidad y persistencia, pero su dispersión es menor y cuentan con un mayor tiempo de acción como son los hilos de plata que dejan los caracoles para poder regresar a su guarida y las feromonas secretadas por las hormigas para guiar a las demás en la búsqueda de alimentos.

En principio, una hormiga aislada se mueve esencialmente al azar, pero las siguientes deciden con una buena probabilidad seguir el camino con mayor cantidad de feromonas.

Considere la Figura 5 en donde se observa como las hormigas establecen el camino más corto. En el figura (a) las hormigas llegan al punto en que tienen que decidir por uno de los caminos que se les presenta; en (b) realizan la elección de manera aleatoria, algunas hormigas eligen el camino hacia arriba y otras hacia abajo; en (c) como las hormigas se mueven aproximadamente a una velocidad constante, las que eligieron el camino más corto alcanzarán el otro extremo más rápido que las otras que tomaron el camino más largo, depositando mayor cantidad de feromona por unidad de longitud; en (d) la cantidad de feromona depositada en el trayecto más corto hace que la mayoría de las hormigas elijan este camino, por realimentación positiva.

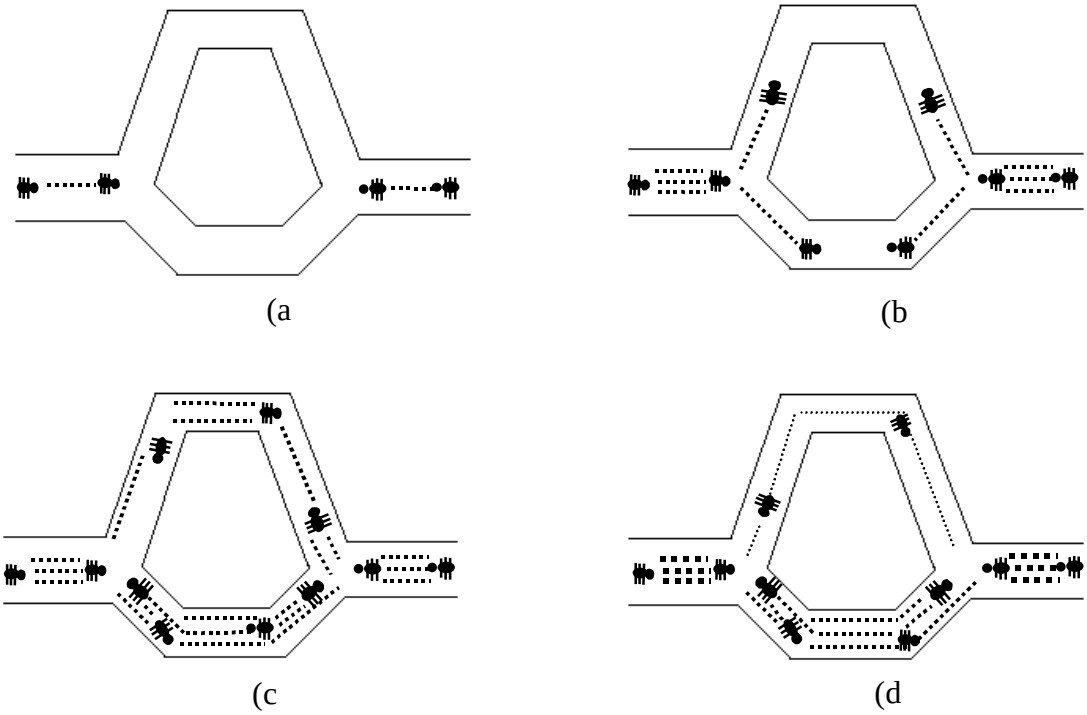


Figura 5: **Comportamiento de las hormigas reales**

La técnica de las hormigas para establecer el camino más corto ha inspirado este novedoso sistema aplicable a numerosos problemas, como fuera mencionado en la sección 2.4.

3.2 Principales variables

Inspirados en el comportamiento de las hormigas, arriba descripto, Dorigo et al. [DMC96] proponen el algoritmo *Ant System (AS)*, presentado a continuación.

Para el presente trabajo, se considera un conjunto de *MAXC* ciudades que deben ser visitadas una sola vez con el objeto de encontrar la longitud mínima de recorrido y se define $b_i(t)$ *(i=1, ...,MAXC)* como el número de hormigas en la ciudad *i* al tiempo *t*. Por consiguiente, el número total de hormigas *MAXH* estará dado por:

$$MAXH = \sum_{i=1}^{MAXC} b_i(t) \qquad (1)$$

Cada hormiga es un simple agente con las siguientes características:

- Elige la ciudad destino calculando una probabilidad que es una función de la distancia entre las ciudades origen y destino y la cantidad de feromona presente en el arco que las conecta.
- Se obliga a cada hormiga a realizar un tour legal, esto es, visitar cada ciudad una sola vez. Viajar a una ciudad ya visitada no está permitido hasta que complete todo el viaje (esto es controlado por una lista tabú).
- Cuando viaja de la ciudad *i* a la ciudad *j*, deposita una cantidad de feromona, en el arco *(ij)*, marcando el camino recorrido.

Para satisfacer la restricción de que una hormiga visite todas las ciudades una sola vez, se asocia a cada *hormiga k* una estructura de datos llamada lista tabú, $\mathbf{tabu_k}$, que guarda las ciudades ya visitadas por dicha hormiga. Una vez que todas las ciudades hayan sido recorridas, el trayecto o tour (ciclo) es completado, la lista tabú se vacía y nuevamente la hormiga está libre para iniciar un nuevo tour. Se define como $\mathbf{tabu_k(s)}$ al elemento *s-esimo* de la lista tabú de la *hormiga k*.

Dado un conjunto de MAXC ciudades, denominamos d_{ij} a la longitud del camino entre las ciudades *i, j*; en el caso del Problema de Cajero Viajante Euclidiano.

El punto de partida para la solución del problema simétrico del cajero viajante, es la matriz de distancias $D = \{d_{ij}$, distancia entre la ciudad *i* y la ciudad *j*$\}$, a partir de la cual se calcula la visibilidad $\eta_{ij} = 1/d_{ij}$. Por su parte, se denota como $\tau = \{\tau_{ij}\}$ a la matriz de feromonas a ser utilizada para consolidar la información que va siendo recogida por las hormigas; en otras palabras, la cantidad de feromona que se va almacenando entre cada par de ciudades (*i j*).

$\tau_{ij}(t)$ especifica la intensidad de las feromonas del arco (*i j*) en el tiempo *t*, y se actualiza según:

$$\tau_{ij}(t+1) = \rho \times \tau_{ij}(t) + \Delta\tau_{ij}(t, t+1) \qquad \textbf{(2)}$$

donde ρ es el coeficiente de persistencia de las feromonas, de forma tal que (1-ρ) representa la evaporación de la substancia entre *t* y *t*+1, mientras que la cantidad de feromona depositada en un arco (*i j*), en dicho intervalo de tiempo, está dada por:

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{MAXH} \Delta\tau_{ij}^{k}(t, t+1) \qquad \textbf{(3)}$$

con $\Delta\tau_{ij}^{k}(t, t+1)$ representando la cantidad de feromona depositada en el arco (*i j*) por la hormiga *k-esima* entre *t* y *t+1*.

Durante la ejecución del algoritmo *Ant System*, cada *hormiga* elige en forma probabilística la próxima ciudad a visitar, realizando un cálculo de probabilidad que es función de la distancia y la cantidad de feromona depositada en el arco que une a las ciudades origen-destino, esto es:

$$p_{ij}(t) = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \times \left[\eta_{ij}\right]^{\beta}}{\sum_{j \notin Tabu_k} \left[\tau_{ij}(t)\right]^{\alpha} \times \left[\eta_{ij}\right]^{\beta}} & \text{si } j \notin \text{Tabu}_k \\ 0 & \text{de otra manera} \end{cases} \quad \textbf{(4)}$$

donde α y β son constantes que expresan la importancia relativa del sendero de feromonas y la distancia entre las ciudades respectivamente. Así, un alto valor de α significa que el sendero de feromonas es muy importante y que las hormigas tienden a elegir caminos por los cuales otras hormigas ya pasaron. Si por el contrario, el valor de β es muy alto, una hormiga tiende a elegir la ciudad más cercana.

En el instante *t*, las *hormigas* se mueven de una ciudad a la siguiente (movimiento llamado: iteración), en donde se encontrarán en el instante *t*+1. Lógicamente, al cabo de (MAX*C* – 1) iteraciones, las hormigas han visitado la última ciudad y están en condiciones de regresar a su ciudad origen, posiblemente para actualizar la matriz de feromonas con la información recogida en el tour completo.

El proceso se repite iterativamente hasta que se cumpla algún criterio de parada. En este trabajo, el proceso termina si el contador de tour alcanza un número máximo de ciclos *NCMAX* (definido por el usuario) o todas las hormigas realizan el mismo tour. En este último caso, es evidente que las hormigas han dejado de buscar nuevas soluciones, lo que constituye un criterio de convergencia del algoritmo (similar a la uniformización de la población de un algoritmo genético [BCC98]).

A continuación se presentan las distintas versiones del algoritmo *Ant System*. Como se había adelantado en el capítulo anterior, se diferencian principalmente, por el momento y la manera de actualizar la matriz de feromonas.

3.3 Distintas versiones de Ant System

3.3.1 Ant Density

En el modelo *Ant Density*, la cantidad Q_1 de feromonas es depositada en el arco *(ij)* cuando la hormiga *k* se mueve de la ciudad *i* a la ciudad *j*. De ésta manera tenemos que en el modelo *Ant Density:*

$$\Delta\tau_{ij}^{k}(t,t+1)=\begin{cases} Q_1 & \text{si la hormiga } k\text{-}esima \text{ camina por el arco } (i,j) \\ 0 & \text{de otra manera} \end{cases} \quad \textbf{(5)}$$

A partir de esta definición, es claro que, al viajar una hormiga desde la ciudad *i* a la ciudad *j*, el incremento en la intensidad de feromonas del arco *(i j)* es independiente de la distancia.

3.3.2. Ant Quantity

En el modelo *Ant Quantity*, la cantidad de feromonas depositada en el trayecto es inversamente proporcional a la distancia a d_{ij} y por lo tanto, caminos más cortos son más deseables. Para esto, se realiza el siguiente cálculo de $\Delta\tau_{i,j}^{k}$:

$$\Delta\tau_{ij}^{k}(t,t+1)=\begin{cases} \dfrac{Q_2}{d_{ij}} & \text{si la hormiga } k\text{-}esima \text{ camina por el arco } (ij) \\ 0 & \text{de otra manera} \end{cases} \quad \textbf{(6)}$$

donde Q es una constante y d_{ij} es la longitud del arco $(i\ j)$.

3.3.3 Ant Cycle

En el modelo *Ant Cycle* la cantidad de feromonas depositada en el trayecto es proporcional a la distancia del tour completo encontrado y por lo tanto, es de esperar una apreciable capacidad de búsqueda de soluciones globales. Para esto, se realiza el siguiente cálculo de $\Delta\tau_{ij}^{k}$:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q_3}{L_k} & \text{si la hormiga } k\text{-}esima \text{ camina por el arco } (ij) \\ 0 & \text{de otra manera} \end{cases} \qquad \textbf{(7)}$$

donde Q_3 es una constante y L_k es la longitud del tour completo realizado por la hormiga *k* para recorrer las MAXC ciudades.

3.4 Pseudocódigos

La presente sección presenta los algoritmos de las diferentes versiones de *Ant System*. En la sección 3.4.1 se introduce el pseudocódigo de dos versiones muy similares, *Ant Density* y *Ant Quantity*, con una breve explicación de cómo trabaja el algoritmo. En la sección 3.4.2 se introduce el pseudocódigo de la versión *Ant Cycle*, distinta a las anteriores en el momento y la manera de actualizar la matriz de feromonas.

3.4.1 Ant Density y Ant Quantity

A continuación se presenta el pseudocódigo de las versiones *Ant Density* y *Ant Quantity* [CDM92].

Pseudocódigo 4: *Ant Density* y *Ant Quantity* de Dorigo et al [CDM92].

1. Fase de inicialización

   $t = 0$

   Inicializar contador de ciclos *NC*

   Para cada arco ***(ij)***:

   valor inicial de $\tau_{ij}(t) = c$ /**c*= constante positiva pequeña*/

   $\Delta\tau(ij) = 0$

   Colocar ***MAXH*** hormigas en ***N*** ciudades

2. Colocar la ciudad origen en lista ***tabu***$_k$ de cada hormiga

3. Repetir hasta llenar ***tabu***$_k$

   Para cada hormiga:

   Elegir próxima ciudad a ser visitada según ecuación (4)

   Mover la hormiga a la próxima ciudad

   Insertar la ciudad en ***tabu***$_k$

   Calcular:

   $\Delta\tau_{ij}(t,t+1) = \Delta\tau_{ij}(t,t+1) + Q_1$ en el caso de *Ant Density* o

   $\Delta\tau_{ij}(t,t+1) = \Delta\tau_{ij}(t,t+1) + Q_2 / d_{ij}$ en el caso de Ant Quantity

   Para cada arco ***(ij)***

   Actualizar τ_{ij} según ecuación (2)

4. **Guardar el camino más corto hasta ciclo** *NC*: $L_0^{NC} = min\{L_0^{NC-1}; min_k\{L_k\}\}$

5. **Aumentar contador de ciclos** *NC*

   Si (***NC*** < ***Nc$_{max}$***) y (no todas las hormigas realicen el mismo tour)

   Vaciar ***tabu***$_k$

   Ir a la fase 2

   Sino

   Imprimir camino más corto L_0^{NC}

   Fin

En términos comunes el algoritmo trabaja de la siguiente manera. En el tiempo cero, *t=0,* la fase de inicialización se realiza, las hormigas son colocadas en las ciudades de origen (para nuestros experimentos corresponde una hormiga por cada ciudad). Se inicializa el valor de feromonas para cada arco con una constante positiva pequeña, por ejemplo 0,1. Cada hormiga posee su lista tabú, el primer elemento de cada lista es igual a la ciudad origen de cada hormiga.

Cada hormiga elige la próxima ciudad a visitar calculando la probabilidad como una función del nivel de feromonas en el arco *(i j)* que da información acerca de cuantas hormigas han pasado por ese trayecto y la visibilidad que da información acerca de la distancia hasta la ciudad destino, las ciudades más cercanas son más deseables.

Cada vez que una hormiga realiza un movimiento, la feromona depositada en el arco *(i j)* es sumada a la cantidad de feromonas depositada en el mismo arco en el pasado.

Después de *MAXC-1* movimientos, la lista tabú está llena, se calcula y guarda el camino más corto hallado por las *MAXH* hormigas y se vacían las listas tabú. Las hormigas están listas para empezar de nuevo su viaje. El proceso es iterado hasta que el contador de ciclos *NC* alcance el número máximo de ciclos definido por el usuario NC_{max} o todas las hormigas realicen el mismo tour, en consecuencia dejan de buscar nuevas soluciones.

3.4.2 Ant Cycle

A continuación se presenta el pseudocódigo de *Ant Cycle* [CDM92], conocido simplemente como *Ant System* por obtener mayor rendimiento que los algoritmos presentados en la sección anterior.

Pseudocódigo 5: *Ant Cycle* de Dorigo et al [CDM92].

1. Fase de inicialización

   Inicializar contador de ciclos *NC*

   Para cada arco ***(i.j)***:

   valor inicial de $\tau_{ij}(t) = c$ /**c*= constante positiva pequeña*/

   $\Delta\tau(ij) = 0$

   Colocar ***MAXH*** hormigas en ***N*** ciudades

2. Colocar la ciudad origen en lista $\boldsymbol{tabu}_k$ de cada hormiga

3. Repetir hasta llenar $\boldsymbol{tabu}_k$

Para cada hormiga:

Elegir próxima ciudad a ser visitada según ecuación (4)

Mover la hormiga a la próxima ciudad

Insertar la ciudad en $\boldsymbol{tabu}_k$

4. Repetir para cada hormiga $\boldsymbol{k}$

Regresar a la ciudad origen

Calcular la longitud L_k **del ciclo**

Guardar el camino más corto hasta ciclo NC**:** $L_0^{NC} = min\{L_0^{NC-1}; min_k\{L_k\}\}$

Para cada arco *(i,j)*

Calcular $\Delta\tau_{ij}$ según ecuación (3)

5. Para cada arco ***(i,j)***

Actualizar τ_{ij} según ecuación (2)

$\Delta\tau_{ij} = 0$

6. Aumentar contador de ciclos *NC*

Si $\boldsymbol{NC} < \boldsymbol{Nc_{max}}$ y (no todas las hormigas realicen el mismo tour)

Vaciar $\boldsymbol{tabu}_k$

Ir a la fase 2

Sino

Imprimir camino más corto L_0^{NC}

Fin

Se puede observar que *Ant Cycle* trabaja de manera idéntica a los algoritmos anteriores, excepto en el momento y la manera de actualizar la matriz de feromonas. *Ant Density* y *Ant Quantity* actualizan la matriz de feromonas con cada movimiento de las hormigas, es decir, cuando una hormiga viaja de la ciudad *i* a la ciudad *j*, una cantidad de feromonas es depositada en ese trayecto, sin embargo en *Ant Cycle*, las hormigas deben terminar el recorrido y calcular la distancia encontrada para después depositar la feromona en el

trayecto recorrido. Como la cantidad de feromonas a ser depositada es dividida por la longitud del recorrido, es evidente que el tour más corto recibirá mayor cantidad de feromonas y por tanto hará más deseable el camino para las siguientes hormigas. La diferencia en la cantidad de feromonas depositada por cada versión es detallada en las secciones 3.3.1 al 3.3.3.

3.5 Evolución de la matriz de feromonas

La matriz de feromonas es utilizada por las hormigas para acumular el conocimiento que van adquiriendo acerca del problema. Al iniciar las corridas, la matriz es inicializada con una constante pequeña que para nuestros experimentos es 0,1. Las matrices que se presentan a continuación a modo de ejemplo son el resultado del algoritmo *Ant Quantity*, para el problema de 15 ciudades cuyos datos se presentaron en la sección 2.4.

Después de 10 ciclos la matriz presenta zonas en donde la cantidad de feromonas inicial ha disminuido considerablemente porque las hormigas no eligieron esos caminos y por lo tanto no fue depositada nueva feromona, además cierta cantidad fue evaporada; mientras que en otros sectores ha aumentado notoriamente, marcando ya desde el principio ciertos caminos como deseables para las siguientes hormigas.

Después de 50 ciclos se puede notar que muchos caminos fueron reforzados considerablemente en la cantidad de feromonas que contenían. Así mismo, aparecen nuevos caminos marcados como más deseables que otros. En las matrices siguientes se puede observar prácticamente lo mismo, si bien el algoritmo ya ha encontrado varias la solución óptima aún no ha llegado a converger.

Después de 825 ciclos el algoritmo fue capaz de encontrar la solución óptima en 30 ciclos consecutivos. La matriz es presentada sin decimales con el único objeto de observar con facilidad los caminos con mayor cantidad de feromonas, además de aquellos caminos que por contener una cantidad tan pequeña de feromonas pueden ser considerado cero. Los caminos que forman la solución óptima están marcados con un recuadro, es evidente que al

tratarse de un problema simétrico (*ij=ji*), el camino de regreso también contiene gran cantidad de feromonas, que en la matriz aparece marcado con un fondo gris, demostrando como la matriz de feromonas guarda el conocimiento adquirido durante el proceso.

0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0	0.1
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0

Matriz de feromonas inicial *t*=0

0	**98.6**	**17.1**	3.32	5.18	0.09	0.09	0.09	0.09	0.36	0.09	2.25	0.09	0.27	0.09
0.09	0	0.09	0.47	31.4	7.78	**61.2**	0.09	0.09	0.34	0.09	0.09	0.09	0.09	0.09
101	0.09	0	17.1	0.52	0.44	0.09	0.09	4.36	0.49	0.09	0.42	0.09	0.28	0.09
0.09	1.21	**30.6**	0	2.02	0.09	0.88	0.35	0.85	4.44	5.03	**49.1**	0.09	0.09	0.09
5.59	1.86	0.09	9.79	0	**50.2**	0.76	0.09	0.6	0.09	0.33	0.71	0.5	0.25	0.09
7.73	**12.6**	1.88	2.59	**22.1**	0	16.2	0.09	0.09	1.04	1.19	0.72	0.88	0.35	1.24
0.5	12.9	0.09	0.35	0.09	1.15	0	**117**	0.09	0.09	0.09	0.09	0.09	0.24	0.09
0.09	0.09	0.09	1.75	0.09	8.96	**20.1**	0	**58.5**	0.09	0.28	0.09	0.09	0.09	0.09
0.09	0.09	**49.6**	0.09	2.89	0.09	0.09	11.1	0	**19.5**	0.35	0.35	0.09	0.47	0.09
1.58	0.59	0.09	7.96	3.53	1.97	1.48	0.36	**19.1**	0	**18.8**	0.41	0.63	0.36	3.73
0.94	0.09	0.09	1.35	0.57	0.44	0.44	0.09	1.15	4.7	0	35	36	0.09	15.5
0.09	0.09	0.09	22.9	0.69	0.29	0.09	0.3	0.09	0.09	41.3	0	**39.4**	0.09	0.09
0.32	0.09	0.09	1.46	0.71	0.09	0.09	0.4	0.28	2.01	11.6	9.87	0	61.6	0.09
0.27	0.09	0.29	0.37	0.25	0.09	0.09	0.09	0.09	2.69	2.67	0.09	**16.1**	0	**88.5**
0.09	0.09	0.32	3.45	0.77	0.37	0.09	0.09	0.09	27.3	9.43	1.81	0.9	34	0

Matriz de feromonas después de 10 ciclos

0	**385**	**69**	15.13	23.96	0.05	0.05	0.42	1.14	0.43	0.36	2.71	0.85	0.16	0.05
16.03	0	0.05	3.04	55.01	**119**	**221**	0.05	0.05	0.20	0.05	0.05	0.05	0.05	0.22
411	1.22	0	71.01	2.73	0.54	0.87	0.05	6.84	0.29	0.05	0.25	0.05	0.17	0.05
0.05	1.01	**111**	0	9.27	1.30	4.00	1.19	1.22	7.99	11.65	**219**	1.74	0.54	0.26
31.18	7.50	2.99	42.36	0	**141**	7.03	1.74	2.36	1.30	1.84	9.43	2.72	0.77	1.63
16.39	48.50	5.84	7.56	**116**	0	**60.58**	2.61	0.50	2.82	2.70	2.70	2.86	1.78	3.45
0.70	47.39	2.19	0.84	1.20	7.04	0	**446**	0.88	0.85	0.53	0.23	0.19	0.29	0.05
0.68	0.84	1.42	7.94	2.27	26.16	**87.26**	0	**224**	0.71	0.80	0.60	0.05	0.05	0.05
0.41	0.05	**187**	0.43	7.08	2.02	0.05	**48.19**	0	**83.23**	0.67	0.87	0.20	0.48	0.29
2.92	4.47	0.05	29.53	19.70	8.52	6.38	1.18	**89.42**	0	40.83	0.98	3.10	0.71	16.60
1.53	0.23	0.05	10.54	2.03	0.56	0.39	0.05	1.19	15.05	0	**101**	**222**	0.42	31.67
0.71	0.05	0.64	**94.79**	5.11	0.17	0.25	0.18	0.72	0.36	**242**	0	**73.96**	0.05	0.30
0.39	0.19	0.45	2.73	2.12	0.18	0.17	0.24	0.30	5.29	39.71	33.76	0	**266**	0.05
0.16	0.05	0.53	0.22	2.18	0.16	0.05	0.05	0.22	5.79	1.97	0.35	**55.55**	0	**379**
0.05	0.05	3.10	5.08	2.65	0.69	0.71	0.05	0.26	**121**	41.76	9.52	2.21	**106**	0

Matriz de feromonas después de 50 ciclos

0	**728**	**209**	18.89	34.21	0.36	0.74	0.82	0.98	0.61	0.21	0.45	0.26	0.22	0.01
112	0	0.01	4.29	28.06	**416**	**283**	0.01	0.01	0.42	0.01	0.01	0.10	0.09	0.18
698	0.17	0	**191**	6.84	1.68	6.12	0.01	19.65	0.94	0.70	0.09	0.20	0.19	0.01
0.10	1.26	**173**	0	11.04	1.38	6.13	1.26	2.21	6.57	13.94	**517**	1.73	1.23	0.22
91	20.33	11.27	**78**	0	**179**	50.79	4.28	5.00	2.22	2.56	17.83	4.30	2.51	2.96
13.74	**69**	8.56	8.17	**328**	0	**88**	9.08	2.20	3.56	3.47	5.84	3.12	2.23	4.28
2.15	**131**	9.89	3.25	8.89	16.99	0	**762**	2.57	2.84	0.25	0.39	1.05	0.44	0.05
4.01	0.71	0.28	23.16	3.65	36.82	**279**	0	**355**	2.96	1.71	1.88	0.96	0.46	0.26
0.14	0.51	**340**	0.13	12.56	2.47	0.01	**148**	0	**142**	0.87	1.82	0.03	0.58	0.25
4.40	6.60	0.01	61.06	31.24	13.71	13.93	2.70	**232**	0	31.83	0.77	5.86	3.66	39.85
0.28	0.23	0.65	37.90	3.70	0.23	0.72	0.94	1.16	12.20	0	**113**	**564**	1.50	18.35
3.68	0.35	1.19	**140**	7.24	0.11	2.11	0.20	0.56	0.05	**625**	0	**48**	0.84	0.40
0.20	0.22	1.17	2.27	3.72	0.12	0.17	0.28	0.14	4.93	54.83	**69**	0	**551**	0.21
0.37	0.01	0.24	0.22	2.48	0.02	0.20	0.03	0.03	3.01	2.58	0.60	**90**	0	**794**
0.52	0.26	6.31	4.34	6.92	0.58	1.33	0.51	0.49	**292**	**61**	9.49	1.00	**146**	0

Matriz de feromonas después de 200 ciclos

	29	**05**	3.46	1.60	.79	.05	.29	.11	.27	.23	.82	.66	.05	
178	0	0	3.58	14.61	**553**	**175**	0	0.29	0.35	0.33	0.35	0.01	0.19	0.01
666	0.01	0	**265**	6.66	1.67	3.79	0.56	23.62	1.56	1.49	0.01	0.37	0.29	0.12
0.36	1.89	**170**	0	10.76	0.91	8.58	1.49	0.98	4.63	7.41	**588**	0.24	1.18	0.18
123	47.78	14.86	65.69	0	**149**	85.28	2.68	2.62	2.19	1.41	17.04	2.50	2.28	2.72
8.93	55.08	7.25	5.78	**392**	0	**97**	9.01	2.81	2.89	4.00	4.88	3.08	2.14	3.14
3.65	**180**	9.50	3.44	11.79	34.28	0	**735**	2.19	2.13	0.85	0.48	0.79	0.64	0.12
4.00	0.78	0.42	32.01	3.22	18.66	**380**	0	**353**	2.03	1.55	3.07	0.70	0.59	0.16
0.45	0.04	**316**	0.01	10.68	0.90	0	**217**	0	**153**	0.30	2.39	0.45	0.36	0.02
5.36	3.94	0.30	70.16	27.40	10.82	15.19	2.28	**280**	0	16.63	1.04	5.95	5.56	51.70
1.09	0.02	1.08	56.00	4.62	0.18	0.21	0.89	0.11	2.28	0	84.81	**652**	0.70	8.43
1.82	0.09	0.12	**118**	9.18	0.21	0.80	0.40	0.04	0.78	**749**	0	23.23	0.73	0.79
0.01	0.17	1.01	0.94	5.62	0.03	0.58	0.22	0.26	2.03	45.53	86.39	0	**598**	0.09
0.10	0	0.18	0.12	2.04	0	0.25	0.02	0.09	1.43	7.62	0.38	**91**	0	**863**
0.08	0.04	10.28	1.27	8.57	0.31	1.07	1.29	0.53	**336**	44.42	5.39	1.02	**149**	0

Matriz de feromonas después de 400 ciclos

0	**686**	**390**	7	9	1	2	0	1	0	0	1	0	1	0
232	0	4	5	12	**638**	55	0	0	0	0	0	0	0	0
579	0	0	**322**	13	1	0	1	28	1	2	0	0	0	0
1	2	**148**	0	10	1	6	2	0	3	1	**636**	0	0	0
144	50	19	44	0	**117**	**112**	1	4	3	4	15	3	2	3
3	44	3	5	**407**	0	**118**	13	4	4	2	4	2	2	1
7	**230**	7	2	15	44	0	**661**	1	3	0	0	1	0	0
2	1	1	36	5	8	**457**	0	**325**	2	2	1	0	1	0
0	0	**273**	1	11	0	0	**272**	0	**145**	0	3	1	0	0
7	2	2	86	23	6	14	1	**303**	0	5	1	5	7	54
1	0	1	67	4	0	0	1	0	1	0	50	**689**	0	3
6	1	0	82	8	0	1	0	0	1	**811**	0	7	0	0
1	0	0	0	6	0	1	0	0	0	27	**90**	0	**620**	1
0	0	0	0	2	0	0	0	0	0	5	1	85	0	**886**
0	0	13	0	9	0	0	2	0	**350**	40	2	1	**129**	0

Matriz de feromonas después de 825 ciclos

Capítulo 4

Mejoras propuestas para Ant Quantity

4.1 Problema propuesto

Para probar el desempeño del algoritmo se utilizó el problema del cajero viajante en Paraguay. Se consideraron las 15 ciudades mostradas en la Figura 6, cuyos datos fueron presentados en la sección 2.4.

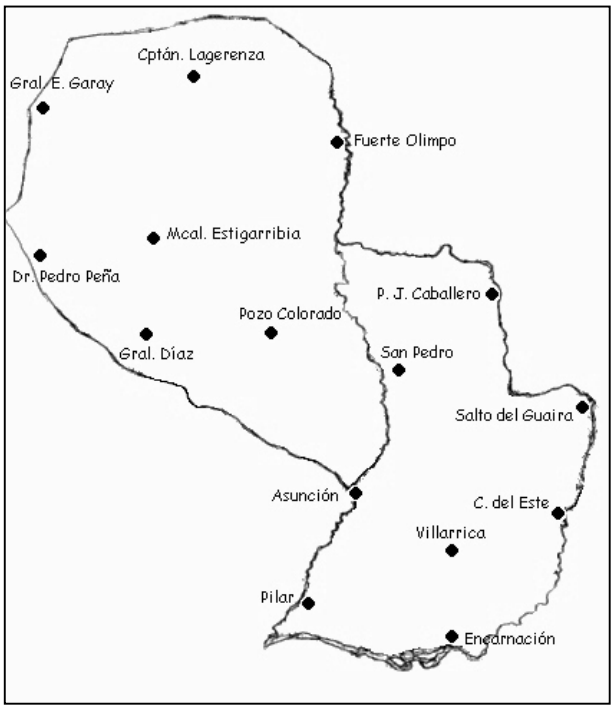


Figura 6: Mapa del Paraguay con las 15 ciudades para el TSP.

Se planea viajar en avión y por lo tanto las distancias se miden en línea recta, minimizando la distancia total recorrida, saliendo y llegando al mismo aeropuerto (ciclo completo).

Para que el cajero encuentre la mejor solución del problema debe evaluar 15! (factorial de 15) caminos posibles o sea 1.307.674.368.000 posibilidades partiendo de cualquier ciudad. Claramente, un análisis exhaustivo está fuera de las posibilidades de un computador personal, aunque existen diversos métodos heurísticos que permiten encontrar buenas soluciones, pero no necesariamente la óptima.

4.2 Mejoras introducidas

La mejora que se propone al algoritmo *Ant Quantity* [DMC96], consiste en inicializar la matriz de feromonas conforme:

$$T_{ij}(0) = f_{min} + \left[\frac{f_{max} - f_{min}}{d_{max} - d_{min}} \right] * \left(d_{max} - d_{ij} \right) \qquad \textbf{(8)}$$

donde:

f_{max} es una constante definida como valor máximo de feromonas correspondiente a la menor distancia.

f_{min} es una constante definida como valor mínimo de feromonas correspondiente a la mayor distancia.

d_{max} mayor distancia de la matriz de datos d_{ij}.

d_{min} menor distancia de la matriz de datos d_{ij}.

Con esto, se privilegian los caminos más cortos en la probabilidad de que una hormiga elija cierta ciudad y como se verá en las próximas secciones, esta propuesta mejora notoriamente los resultados experimentales.

Como se puede apreciar en la Figura 7, a una mayor distancia le corresponde una cantidad menor de feromonas en *t=0*. Para los experimentos presentados en la próxima sección se considera f_{max} =1000 y f_{min} =0.1.

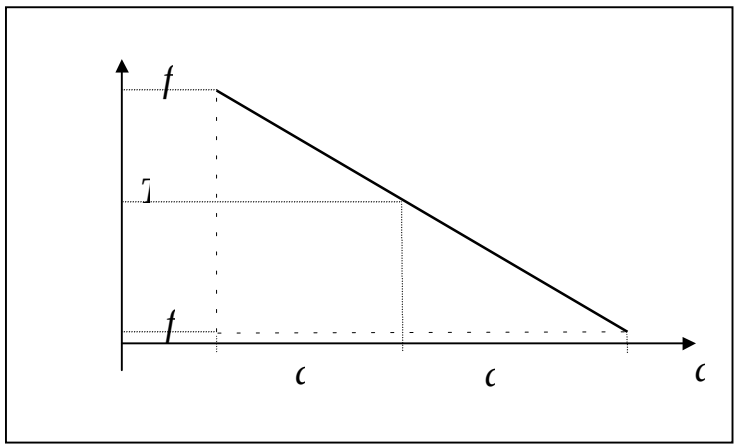


Figura 7: Escalamiento lineal de feromonas con respecto a la distancia para inicializar la matriz de feromonas.

4.3 Resultados experimentales

Para la implementación de los dos algoritmos AS, se utilizó como plataforma computacional una workstation DEC 3000 modelo 300 con procesador ALPHA de 150 MHz con 32 MB de memoria RAM y operando bajo el sistema operativo OSF/1 versión 2.0.

Para comparar el desempeño de los dos algoritmos se han realizado 10 corridas con cada posible combinación de los parámetros α, β y ρ. Cada parámetro posee un conjunto de valores con los que se realizaron las experimentaciones, como se muestra a continuación:

$\alpha \in \{0.5; 1.0; 2.0\}$

$\beta \in \{2.0; 5.0; 10.0\}$

$\rho \in \{0.5; 0.9; 0.99\}$

Por consiguiente, de lo expuesto arriba, existen 27 posibles combinaciones de los parámetros. En la Figura 8 se pueden apreciar los mejores resultados de cada corrida. Para resultados individuales se puede afirmar que el método de la matriz de feromonas escalada propuesto en el presente trabajo, converge a la solución óptima en menor tiempo y con menor dispersión, si bien ambos métodos llegaron a un tiempo mínimo de 0.049998 segundos, el método de Dorigo et al. no converge a una buena solución (2225 km de distancia), mientras que el método propuesto en este trabajo converge a la solución óptima (2195 km. de distancia).

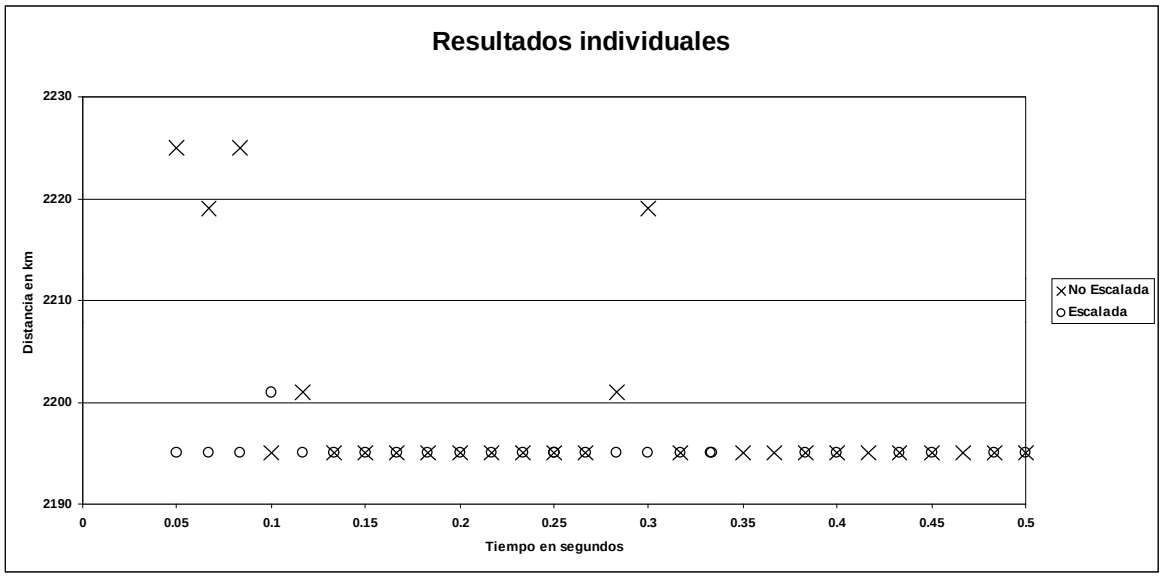


Figura 8: Gráfico con los mejores resultados de cada corrida

La Figura 9 grafica los mejores promedios de 10 corridas (distancia del tour en función del tiempo de cómputo para encontrar esta solución). En dicha figura, se puede apreciar que:

a. La solución más rápida de 0.1866 segundos encuentra un tour de 2255.7 km y corresponde al método propuesto en este trabajo.
b. La solución óptima de 2195 km es encontrada con el método propuesto en 0.2383 segundos con α=2, β=10, ρ=0.9 y en 0.3299 segundos con α=1, β=5, ρ=0.9.

c. Por su parte, el método de Dorigo et al. solo encuentra la solución óptima en 0.8149 segundos.

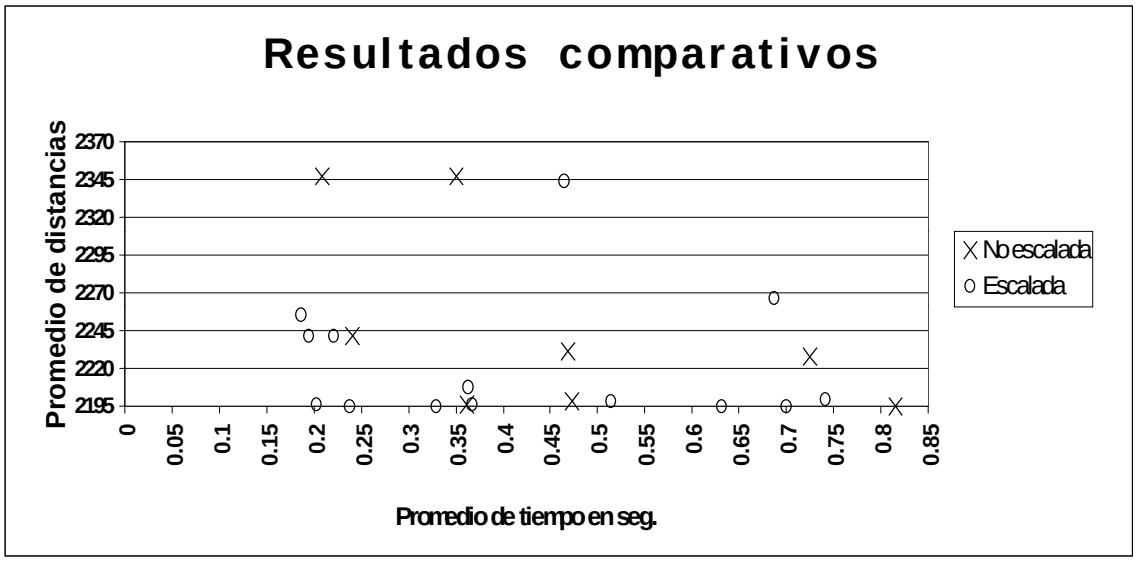
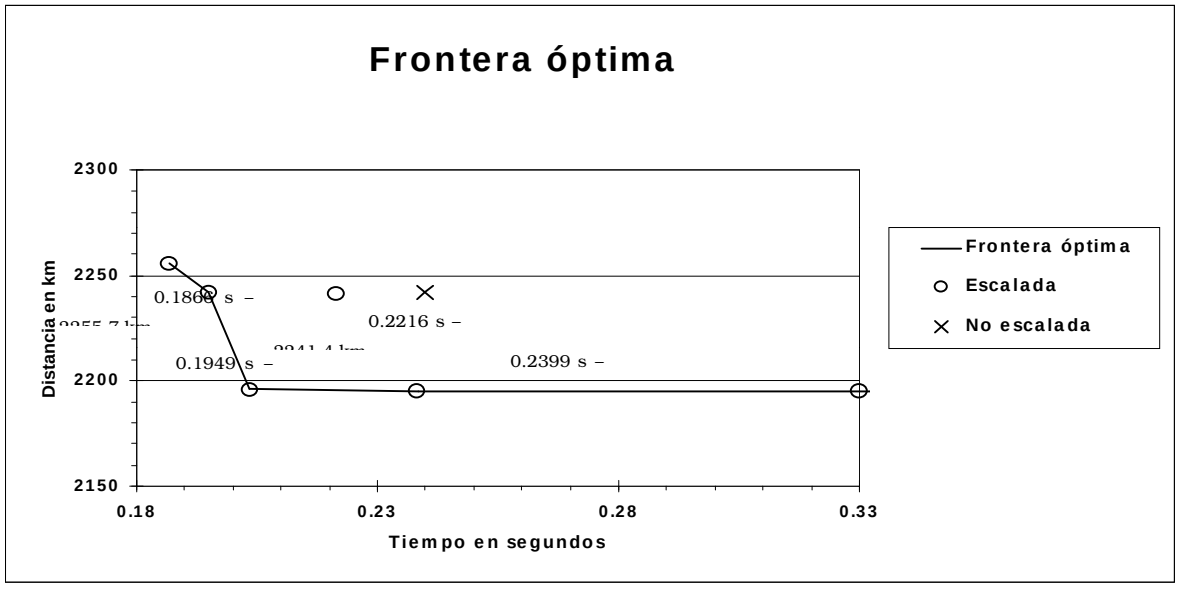


Figura 9: Gráfico con los mejores resultados promediados para 10 corridas sucesivas.

Los mejores promedios en tiempo y distancia están graficados en la Figura 10 formando una curva Pareto con los puntos óptimos (en el sentido de menor distancia y/o

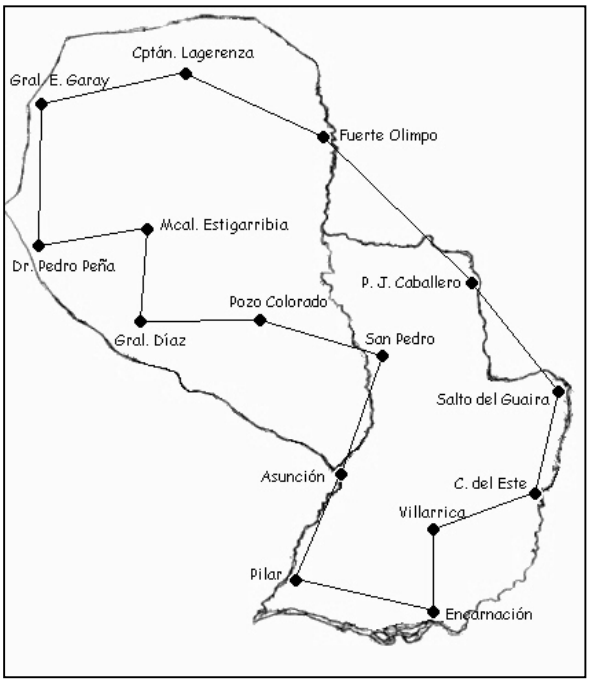


tiempo de cómputo).

Figura 10: Gráfico con los mejores promedios de tiempo y distancia

Se observa que el mejor resultado (0.2399 s - 2242.1 km) en tiempo y distancia obtenido por el algoritmo de Dorigo et al. no forma parte de la frontera óptima, que está totalmente definida por los resultados encontrados con el método aquí propuesto.

En la Figura 11 se presenta el mapa del Paraguay con el mejor tour encontrado. Este tour representa una distancia de 2195 km, partiendo de cualquier ciudad y regresando a la misma después de haber recorrido todas las demás ciudades. Esta solución, promediada para 10 corridas y utilizando el método propuesto, fue encontrada con α=2, β=10, ρ=0.9



con un tiempo de 0.2383 s. Sin embargo, con α=2, β=10, ρ=0.9 sólo hubo 1 corrida que llegó a la solución óptima, 2195 km en 0.04999 s.

Figura 11: Mapa del Paraguay con la solución óptima

En conclusión, el método de inicialización de la matriz de feromonas aquí propuesto mejora el algoritmo originalmente propuesto por Dorigo et al. tanto en tiempo de procesamiento como en la calidad de la solución.

Capítulo 5

Ant Cycle Paralelo

5.1 Paralelismo Asíncrono del Ant Cycle

La complejidad computacional del algoritmo secuencial impide su aplicación a problemas de gran envergadura. Por otra parte, *Ant System* tiene características que lo hacen especialmente apropiado para su paralelización y distribución entre los diversos procesadores de una red de computadoras en un contexto asíncrono. En efecto, el método utiliza la interacción de muchos agentes relativamente simples llamados *hormigas*, pero básicamente independientes entre sí en cada tour, que cooperan intercambiando información en forma asíncrona para el logro de un objetivo común.

Cada hormiga va construyendo su propio tour, con la única restricción de no viajar a una ciudad ya visitada con anterioridad. En consecuencia, el paralelismo está implícito en el mismo algoritmo. Consecuentemente, el presente trabajo propone que cada procesador realice la computación del problema para un cierto número de ciclos, obteniendo resultados parciales que podrán ser transmitidos a los otros procesadores en forma asíncrona, colaborando todos en la solución del problema global, pero sin necesidad de perder tiempo en sincronizar los procesadores, dado que la nueva información (útil para actualizar la matriz de feromonas) solo se usa si está disponible.

Con esta propuesta, el asincronismo elimina los tiempos muertos producidos por la espera en la sincronización de la comunicación, extremadamente perjudiciales cuando se trabaja con una red de computadoras cuyo tráfico no se puede controlar totalmente. En

consecuencia, el presente trabajo propone que las hormigas migren de un procesador a otro en forma asíncrona, respetando políticas migratorias similares a las utilizadas con los Algoritmos Genéticos Paralelos [BCC98, BC97]; en otras palabras, políticas definidas por los siguientes parámetros:

El intervalo de migración: que establece cada cuantos ciclos cierta cantidad de hormigas migrarán de un procesador a otro.

La tasa de migración: que indica cuantas hormigas han de comunicarse al otro procesador cuando se cumpla el intervalo de migración.

El criterio de selección: que determina la política a seguir para la selección de las hormigas que han de migrar. Por ejemplo que sean elegidas al azar. Sin embargo, con el fin de ayudar a los demás procesos, se elegirán las hormigas que hayan obtenido las mejores soluciones (tour con menores distancias).

Cuando cada proceso recibe a las hormigas migrantes, éstas se ubican en sus ciudades de origen, donde se aplica un criterio de selección similar al utilizado con los Algoritmos Genéticos [Gol89] a fin de mantener constante el número original de hormigas por cada ciudad. En otras palabras, las hormigas que hayan encontrado mejores soluciones, tienen mayor probabilidad de sobrevivir, simulando la selección natural en el que él más fuerte sobrevive (concepto similar al operador de selección, utilizado con los Algoritmos Genéticos [BCC98]).

La manera más fácil de entender al operador de selección consiste en imaginar una ruleta (Figura 12), en la que el número máximo de partes en que se divide la ruleta es igual al número de hormigas en la ciudad, y el número de casillas que cada parte ocupa es inversamente proporcional a la distancia encontrada por cada una de las hormigas. Esto se debe a que buscamos minimizar la distancia. En la figura fácilmente se puede observar que la hormiga 2 tiene mayor probabilidad de sobrevivir frente a las demás y que la hormiga 3 tiene la menor probabilidad debido a que encontró una mayor distancia.

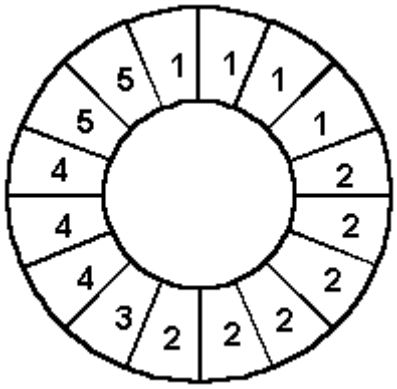


Figura 12: **Ruleta**

N	P	NC
1	0,2500	4
2	0,3750	5
3	0,0625	2
4	0,1875	3
5	0,1250	2
∑	1,0000	

16

N: Identificador de las hormigas ubicadas en la misma ciudad origen

Una vez seleccionadas las hormigas que sobrevivirán al siguiente tour utilizando la ruleta, se actualiza la matriz de feromonas τ_{ij} del procesador en cuestión. Con esto, un buen resultado influye no solo en el procesador que encontró dicho resultado, sino que tiene mayor probabilidad de actualizar inclusive la matriz de feromonas de otros procesadores que la recibieron.

A continuación se presenta el pseudocódigo de la versión paralela implementada que utiliza un proceso *Master* que se encarga de administrar la implementación paralela (Pseudocódigo 6) incluyendo el lanzamiento de los procesos *Esclavos* (Pseudocódigo 7), para que éstos realicen los cálculos propiamente dichos.

Pseudocódigo 6: Proceso Master

1. Fase de inicialización

   Levantar procesos esclavos

   Enviar parámetros a cada esclavo

   Fin= Falso

2. Repetir mientras no sea ***Fin***

   Recibir solución de los esclavos

   Guardar mejor solución

   Si todos los procesos terminaron

   Fin= Verdadero

3. Eliminar procesos esclavos

Cada procesador del sistema distribuido disponible corre un proceso esclavo que básicamente se encarga de resolver el problema utilizando *Ant System*, comunicando sus buenos resultados a los demás procesadores y recibiendo de estos los mejores resultados, intentando con esto una interacción sinérgica entre las mejores hormigas de los diversos procesadores. A continuación se presenta el pseudocódigo de cada proceso esclavo.

Pseudocódigo 7: Proceso Esclavo

1. Fase de inicialización

   Recibir parámetros

   Inicializar variables

2. Repetir mientras $\boldsymbol{NC} < \boldsymbol{NC_{max}}$ y (no todas las hormigas realicen el mismo tour)

   Mover hormigas hasta que cada una complete su tour

   Calcular distancia L_K recorrida para cada hormiga *k*

   Escoger hormigas migrantes

   Enviar hormigas migrantes a otros procesos

   Recibir hormigas migrantes de otros procesos

   Ubicar cada hormiga migrante en su ciudad origen

   Seleccionar las hormigas que sobrevivirán

   Actualizar matriz de feromonas τ

Guardar camino más corto L_0^{NC}

NC = *NC*+**1**

3. Enviar mejor solución al master.

5.2. Variantes implementadas

La Tabla 2 muestra 4 variantes de las estrategias de paralelización que han sido implementadas, entre otras alternativas. Básicamente, se realizaron experiencias modificando la manera en que se eligen las hormigas que viajarán a los demás procesadores (solo las mejores o en forma aleatoria) y que hormigas serán las que actualicen la matriz de feromonas (todas, o solo las mejores).

Viajan / Actualizan	Selección Aleatoria	Sólo las mejores
Todas	AS1	AS2
Las mejores	AS3	AS4

Tabla 2: Versiones implementadas

Para enviar hormigas a los demás procesadores de la red en forma aleatoria (versiones AS1 y AS3), se eligen entre las que encontraron mejores resultados, de este modo, se envían por ejemplo 3 hormigas con sus resultados al procesador 1, otras 3 distintas al procesador 2 y así sucesivamente hasta enviar a todos los procesadores sus respectivas hormigas. En el caso de enviar las hormigas con mejores resultados (versión AS2 y AS4), se eligen un número *k* determinado por el usuario, por ejemplo *k*=5, y estas 5 hormigas se transmiten a todos los procesadores hijos (las mismas *k* hormigas son transmitidas a los demás procesadores).

La matriz de feromonas de un procesador puede ser actualizada por la totalidad de las hormigas sobrevivientes después de la selección (versiones AS1 y AS2), o se pueden elegir solo a las hormigas sobrevivientes con mejores resultados en dicho procesador

(versiones AS3 y AS4). Se hicieron experimentos permitiendo que solo el 10, 20, 30 y 60% de las hormigas (con los mejores resultados) actualicen la matriz de feromonas de un procesador.

5.3 Comparación de las diversas versiones implementadas

La Tabla 3 muestra los resultados obtenidos con las versiones AS1 y AS2 con el problema de 30 ciudades presentado en el Capítulo 2. Claramente, se observa mejores promedios (mayor calidad en las soluciones), a medida que aumenta la cantidad de procesadores.

Nro.de Procesadores	2	4	6
Versión AS1			
Mejor solución	425.649	426.544	425.649
Solución promedio	430.142	429.088	427.217
Desviación estándar	1.632	2.112	1.901
Versión AS2			
Mejor solución	425.820	425.820	425.820
Solución promedio	428.059	427.960	427.543
Desviación estándar	2.216	1.922	1.795

Tabla 3: Versiones AS1 y AS2.

La Tabla 4 compara los resultados obtenidos con las versiones AS3 y AS4, utilizando 5 procesadores, para resolver el problema Oliver30. Esta tabla permite observar que no justifica actualizar la matriz de feromonas con más de 10 a 20% del número total de hormigas, dado que solo se incrementa el esfuerzo computacional sin ninguna mejora apreciable (de hecho empeoran los resultados con el número de hormigas). En consecuencia, se sugiere utilizar k=10% del total de las hormigas, dado que valores menores no conducen a una actualización suficiente de la matriz de feromonas.

Puede notarse en la Tabla 4 que la versión AS3 encuentra 2 veces la solución óptima, por lo que se la puede considerar ligeramente superior a la versión AS4. Dada la superioridad de la versión AS3, ésta fue modificada permitiendo evaporación sólo en los caminos por donde las hormigas han pasado en la iteración actual. Esta nueva versión, denotada AS3.1, presenta un mejoramiento adicional para el problema Oliver30. En consecuencia, se realizaron experimentos con el problema Eilon50 [WSF89], en cuyo caso se encuentra la solución óptima sólo cuando el 10% de las hormigas actualizan la matriz de feromonas, corroborando lo arriba mencionado.

Nro de Hormigas que actualizan feromonas	1	3 10%	6 20%	9 30%	18 60%
Versión AS3					
Mejor solución	424.692	**423.741**	**423.741**	423.912	424.635
Solución promedio	433.236	427.104	426.314	425.954	427.188
Desviación estándar	7.731	4.338	4.428	2.286	3.193
Versión AS4					
Mejor solución	**423.741**	424.692	423.912	423.912	425.649
Solución promedio	435.916	427.771	425.892	425.776	426.930
Desviación estándar	9.201	4.740	2.080	1.898	1.095

Tabla 4: Versiones AS3 y AS4.

Versión AS3.1 – Problema de 30 ciudades					
Nro de Hormigas que actualizan feromonas	1	3	6	9	18
Mejor solución	456.525	**423.741**	**423.741**	**423.741**	**423.741**
Solución promedio	480.557	423.847	423.809	423.758	423.758

Desviación estándar	21.222	0.216	0.084	0.051	0.051
Versión AS3.1 – Problema de 50 ciudades					
Nro de Hormigas que actualizan feromonas	1	5	10	15	30
Mejor solución	723.491	**427.855**	428.721	429.778	436.956
Solución promedio	745.941	429.817	431.812	434.680	444.917
Desviación estándar	12.422	2.188	2.340	2.579	5.268

Tabla 5: Versión AS3.1 para los problemas de 30 y 50 ciudades.

En resumen, de todas las versiones presentadas, la AS3.1 es la que encontró los mejores resultados experimentales, además, el método aplicado para la paralelización del algoritmo aquí propuesto mejora la calidad de la solución al aumentar el número de procesadores, esto se debe a que dispone de mayor información del problema enviada por otros procesadores.

5.4 Resultados experimentales

Para comparar el desempeño de los algoritmos secuencial y paralelo, se hicieron pruebas con los problemas KroA100 [Rei00], Oliver30 y Eilon50 [WSF89]. Se realizaron corridas en una red de 6 computadoras personales con procesadores AMD K6-2 de 350 MHz, y 128 MB de memoria RAM. Estas computadoras personales están conectadas por medio de un switch Ethernet IBM 524, formando una red de área local (LAN) con procesadores corriendo el sistema operativo LINUX RED HAT 6.0.

Las implementaciones fueron realizadas en lenguaje de programación ANSI C utilizando librerías de PVM (*Parallel Virtual Machine*) [GBD+94].

En la tabla 6 se observan los tiempos promedios obtenidos para cada problema y en la Figura 13 se grafica el tiempo promedio de 10 corridas con uno, dos cuatro y seis procesadores para el problema Oliver30, sólo a modo de ejemplo.

Problema	30 Ciudades			50 Ciudades			100 Cuidades		
Nro. Proce-sadores	Tiempo	Aceleración	Eficiencia	Tiempo	Aceleración	Eficiencia	Tiempo	Aceleración	Eficiencia
1	183	Ref. (1)	Ref. (100%)	1002.40	Ref. (1)	Ref. (100%)	6355.70	Ref. (1)	Ref. (100%)
2	94.4	1.94	96.93	484.90	2.07	103.36	3362.20	1.89	94.52
4	47.6	3.84	96.11	243.00	4.13	103.13	1689.70	3.76	94.04
6	32.7	5.60	93.27	162.40	6.17	102.87	1128.40	5.63	93.87

Tabla 6: Tiempos obtenidos para cada problema, Aceleración y Eficiencia

Como ejemplo de los resultados experimentales obtenidos utilizando la red de computadoras personales arriba descripta, la Figura 14 muestra la aceleración promedio en 10 corridas utilizando 2, 5 y 6 procesadores, en este ejemplo, para la versión AS2. Estos resultados se mantienen muy similares para todas las versiones. Claramente, existe una buena escalabilidad con el número de procesadores.

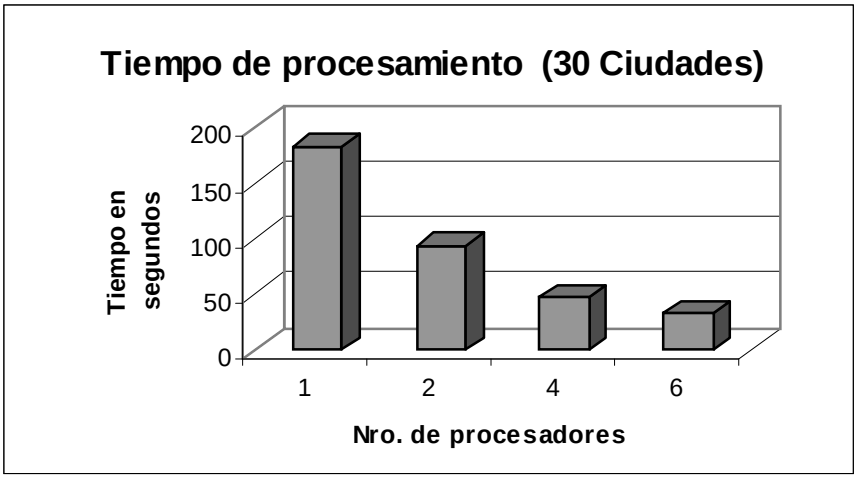


Figura 13: Tiempo de procesamiento para el problema de 30 ciudades

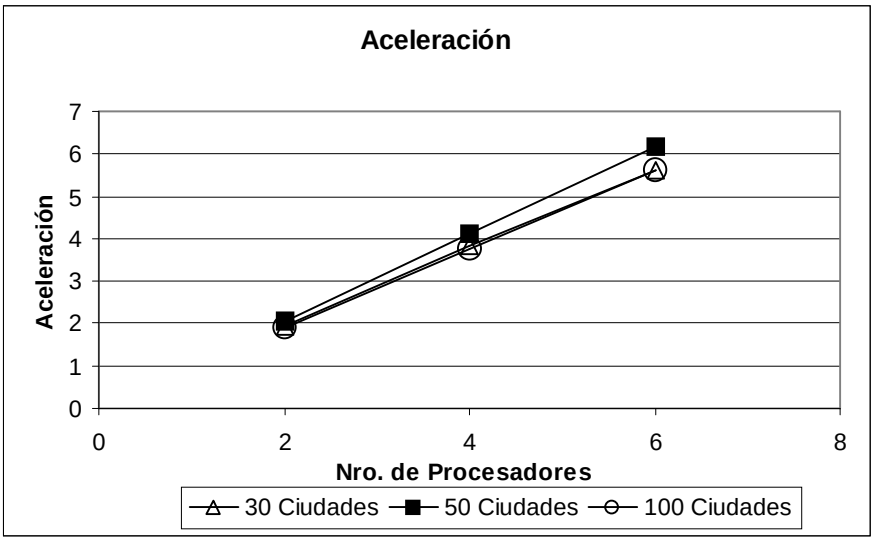


Figura 14: **Aceleración experimental con versión AS2**

En lo relativo a la eficiencia, la Figura 15 demuestra que la paralelización del algoritmo es sumamente benéfica y sencilla, logrando en algunos casos, eficiencias superiores al 100%. En consecuencia, es de esperar que la presente propuesta pueda ser utilizada con un número creciente de procesadores sin mayores complicaciones.

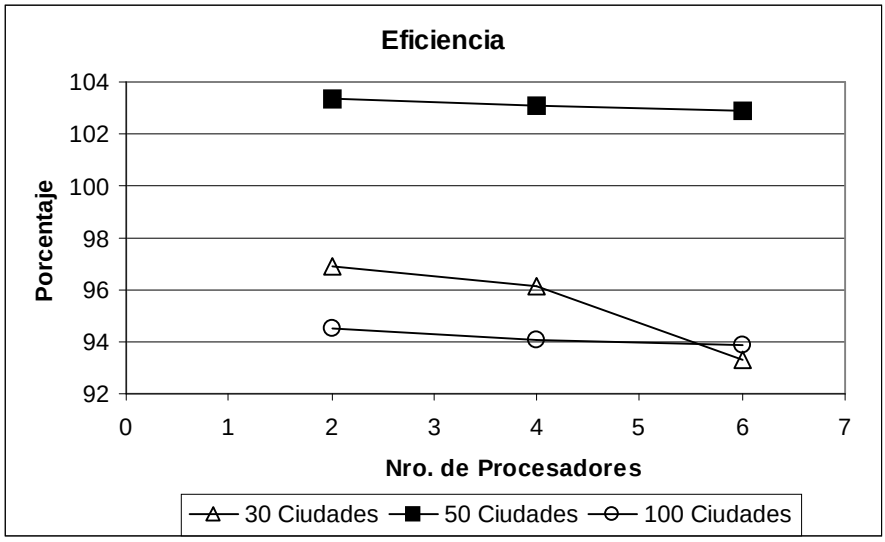


Figura 15: **Eficiencia experimental con versión AS2**

5.5 Porque se utiliza PVM

Así como MPI (*Message Passing Interface*) es el estándar de paso de mensajes en la industria, PVM (*Parallel Virtual Machine*) es el estándar *de facto* del mundo científico. De hecho, en el área de la Física Computacional, PVM es una biblioteca ampliamente usada [GBD+94].

PVM es una herramienta diseñada para solucionar una gran cantidad de problemas asociados con la programación paralela. Para ello, crea una abstracción, que es la máquina paralela virtual, empleando los recursos computacionales libres de todas las máquinas de una red de PCs. Con esto, se dispone de todas las ventajas económicas asociadas a la programación distribuida, ya que empleamos los recursos de hardware de dicho paradigma, pero programando el conjunto de máquinas como si se tratara de una sola máquina paralela.

La máquina paralela virtual es una máquina que no existe, pero un API (*Application Program Interface*) apropiado nos permite programar como si existiese. El modelo abstracto que nos permite usar el API del PVM consiste en una máquina multiprocesador completamente escalable, es decir, que podemos aumentar y disminuir el número de procesadores.

El uso de PVM tiene muchas ventajas, algunas de ellas se citan a continuación [GBD+94]:

- **Precio**. Un conjunto de ordenadores de mediana o baja potencia es muchísimo más barato que el computador paralelo de potencia equivalente. Existen factores fundamentales como la lentitud de la red frente a la velocidad del bus del computador paralelo, que hacen necesarios más ordenadores de pequeña potencia que los teóricos para igualar el rendimiento. Sin embargo, aun teniendo esto en cuenta, la solución es mucho más barata. Además, PVM no necesita de máquinas dedicadas, el *daemon* del PVM corre como un proceso más, se puede

emplear en el proceso los tiempos muertos de los procesadores de todas las máquinas de una red. Por ello, si ya se cuenta con una red Linux montada, el costo de tener un supercomputador paralelo es potencialmente cero, ya que disponemos de las máquinas y la biblioteca PVM es software libre, por lo que no hay que pagar para usarla.

- **Disponibilidad**. Cualquier instituto de educación puede contar con más o menos una docena de máquinas. Esa docena de máquinas 486 que ya no corren ni la última versión del Word para Windows, pueden ser utilizadas para instalar Linux, el PVM y añadirlo al supercomputador paralelo virtual que conforman las máquinas de la red ya disponible.

- **Tolerancia a fallos**. Si por cualquier razón falla uno de los ordenadores que conforman nuestro PVM y el programa que lo usa está razonablemente bien hecho, la aplicación puede seguir funcionando sin mayores problemas. Siempre hay alguna razón por la que alguna máquina puede fallar, la aplicación debe continuar haciendo los cálculos con el hardware disponible.

- **Heterogeneidad**. Podemos crear una máquina paralela virtual a partir de ordenadores de cualquier tipo. El PVM abstrae la topología de la red, la cantidad de memoria de cada máquina, el tipo de procesador y la manera de almacenar los datos. Además, se puede encontrar con facilidad PVM para PowerPC con AIX, Sun con Solaris y PC 80x86 con Linux.

5.6 Comparación con trabajos anteriores que implementan paralelismo

En la sección 2.3 se presenta una cronología de los trabajos publicados sobre *Ant System* aplicados al paradigma del cajero viajante. Cabe destacar que no se mencionaron los trabajos de *Ant System* aplicados a otros problemas, como ruteo de vehículos, asignación cuadrática, ordenamiento secuencial, telecomunicaciones, entre otros.

La técnica de Ant System ha ido madurando desde su nacimiento en 1992, con la investigación del mismo Dorigo y los aportes de distintos autores. Se presentaron varias estrategias de paralelización. En la Tabla 7 se pueden observar las principales diferencias entre ellas, en la cantidad de hormigas utilizadas para cada proceso esclavo, que proceso es el encargado de actualizar la matriz de feromonas y el tipo de comunicación existente entre procesos.

Bullnheimer presentó dos estrategias para el algoritmo *Ant Cycle*:

- en la implementación síncrona, el master levanta un proceso esclavo para cada hormiga que es la encargada de encontrar una solución para el problema. Una vez terminado el recorrido, calcula la distancia y comunica su resultado al master, donde se actualiza la matriz de feromonas con toda la información recibida de los procesos esclavos;
- en la implementación parcialmente asíncrona, los esclavos trabajan de manera independiente por varios ciclos antes de comunicar sus resultados al master, es en este momento en donde se realiza una sincronización de los procesos para la actualización de la matriz de feromonas.

Autores	Algoritmo	Hormigas	Matriz de Feromonas	Paralelismo
Bullnheimer [BKS97]	*Ant Cycle*	Una hormiga por cada proceso esclavo.	Es actualizada por el Master.	Síncrono. Los esclavos comunican al Master sus resultados después de cada ciclo.
				Parcialmente asíncrono. Los esclavos comunican al Master sus resultados después de varios ciclos. Cada proceso corre de manera independiente, la sincronización se lleva a cabo cuando el Master recibe los resultados y actualiza la Matriz de Feromonas.
Stützle [Stü98]	*Max-Min Ant System*	Cada proceso corre con una cantidad igual de hormigas.	Cada proceso cuenta con su Matriz de Feromonas y se actualiza independientemente una de otra.	No existe comunicación entre procesos. El mejor resultado es elegido al final.

Almirón [ACB99, BA00]	*Ant Cycle*	Una hormiga por cada ciudad.	Cada proceso cuenta con su Matriz de Feromonas y se actualiza con información del mismo proceso y/o con las informaciones recibidas de otros procesadores.	Comunicación totalmente asíncrona. Cada proceso envía las hormigas con mejores distancias a sus pares. Las hormigas con mejores resultados (sean nativas o migrantes) tienen mayor probabilidad de actualizar la Matriz de Feromonas local.

Tabla 7: Comparación con trabajos anteriores sobre *Ant System* Paralelo

Stüzle por su parte, implementó la versión paralela de otro algoritmo conocido como *Max-Min Ant System*, cuyas características se detallan en la sección 2.3. El mismo, propone corridas paralelas y completamente independientes del mismo algoritmo en varios procesadores, al final se elige el mejor resultado obtenido de todos los procesos. El método resulta por las características aleatorias del algoritmo, realizando más ciclos es posible encontrar mejores resultados. Esta técnica no emplea comunicación entre los procesos, por lo tanto no existe la cooperación entre ellos.

La principal contribución del presente trabajo al algoritmo *Ant System* es ofrecer la primera implementación totalmente asíncrona además de la versatilidad del método, ya que puede ser aplicado a las distintas variantes de *Ant System* para resolver una gran variedad de problemas. En efecto, la esencia de *Ant System* son los agentes autónomos llamados hormigas y la matriz de feromonas. Con estos elementos resulta relativamente fácil la implementación paralela de las demás versiones aplicada a otros tipos de problemas.

Capítulo 6

Conclusiones

Se ha implementado una novedosa técnica basada en el comportamiento de las hormigas para establecer el camino más corto desde su nido hasta la fuente de alimento y regresar. Simples agentes computacionales, llamados hormigas, colaboran intercambiando información para alcanzar un objetivo común, la optimización del camino. Al moverse de una ciudad a otra, la hormiga deja una señal, marcando el camino recorrido en una matriz de feromonas. Esta información es utilizada por las hormigas que le siguen, con lo cual, los caminos más transitados son más deseables.

Con la técnica, *Ant Quantity*, propuesta por Dorigo et al. [DMC92, DMC96] se resolvió el problema del cajero viajante para 15 ciudades del Paraguay. Se propuso en el algoritmo original iniciar la matriz de feromonas, realizando un escalamiento a partir de la matriz de distancias (dato del problema), mejorando, con esto el desempeño del algoritmo.

Al implementar el algoritmo *Ant Quantity* original y la propuesta del capítulo 4, fue posible concluir lo siguiente:

- La solución óptima para un mismo conjunto de parámetros α, β, y ρ, fue hallada en 0.238 segundos en promedio para el método propuesto, frente a 0.815 segundos para el algoritmo de Dorigo et al. (realizando 10 corridas para cada algoritmo).

- Se realizaron 27 experimentos (cada uno con 10 corridas para ambos algoritmos). En 16 de ellos se encontró la solución óptima utilizando el algoritmo AS con la

mejora propuesta en el presente trabajo. Sin embargo, con el algoritmo de Dorigo et al. sólo en 10 oportunidades, de las 27 llevadas a cabo.

- Si se considera el promedio más rápido de corrida en los 27 experimentos, se observa que la propuesta de Dorigo et al. requiere de 0.208 s para encontrar un tour promedio de 2346.8 Km, mientras el método con inicialización de feromonas propuesto, requiere solo de 0.187 s para encontrar un mejor tour promedio de 2255.7 Km.

En resumen, la propuesta de iniciar la matriz de feromonas encuentra la solución óptima con mayor frecuencia y en menores tiempos de ejecución. Debido a los excelentes resultados experimentales obtenidos, se continuó trabajando con la novedosa técnica *Ant System*, especialmente en lo referente a su paralelización en sistemas distribuidos, sobre la base de agentes móviles llamados *hormigas*.

Ant System tiene características que lo hacen especialmente apropiado para su paralelización y distribución entre diversos procesadores. En efecto, cada hormiga es un agente autónomo que construye su propio tour, con la restricción de no viajar a una ciudad ya visitada con anterioridad.

El presente trabajo presentó diversas alternativas de paralelización asíncrona del algoritmo *Ant System*, analizando diversas políticas de migración de hormigas y actualización de la matriz de feromonas, en el contexto totalmente asíncrono de una red de computadoras.

Resultados experimentales demuestran una buena eficiencia, que eventualmente superó el 100% para los problemas mayores, consiguiéndose muy buena escalabilidad, lo que permite postular su utilización eficiente con un número creciente de procesadores y en problemas de mayor tamaño y complejidad, además de ser un método versátil, aplicable a cualquier variante de *Ant System*.

Como una posible extensión del presente trabajo, se propone utilizar un mayor número de computadores, posiblemente una red heterogénea de PCs, con el objeto de probar su escalabidad y rendimiento con problemas de mayores dimensiones. Así mismo, aplicar optimización local al algoritmo paralelo con el objeto de mejorar la calidad de los resultados obtenidos.

Además se propone implementar la misma estrategia de paralelización utilizada en el presente trabajo para otras variantes de *Ant System* de reciente publicación, aplicadas a resolver problemas tan variados como:

- Problema de la Asignación Cuadrática (*Quadratic Assignment Problem*) [MC99].
- Problema del Ruteo de Vehículos (Vehicle Routing Problem) [BHS99].
- Problema de la Ejecución Puntual de Tareas con Pesos (*Total Weigthed Tardiness Problem*) [BSD00].

En conclusión, Ant System se presenta como un promisor mecanismo de optimización capaz de resolver satisfactoriamente problemas combinatorios de notable complejidad y gran porte, aprovechando al máximo la accesibilidad a un número creciente de computadores personales heterogéneos, interconectados en red.

Bibliografía

[ACB98] Almirón M, Chaparro E. y Barán B., "Optimización basada en sistema de hormigas con heurística de inicialización", *IX Panel de Informática – Expomática'98* – Asunción – Paraguay, 1998.

[ACB99] Almirón M, Chaparro E. y Barán B., "Sistema distribuido de hormigas para el problema del cajero viajante" *XXV Conferencia Latinoamericana de Informática – CLEI'99*, Asunción – Paraguay, 1999.

[Arr98] Arriaga R., "Desarrollo de un sistema basado en redes neuronales para apoyar el análisis de riesgo de créditos de consumo*"*. *XXIV Conferencia Latinoamericana de Informática – CLEI'98*, Quito – Ecuador, 1998.

[Atm76] Atmar J. W., "Speculation on the evolution of intelligence and its possible realization in machine form"'. *Ph.D. dissertation*, New Mexico State University, Las Cruces, 1976.

[BA00] Barán B. y Almirón M., "Colonias distribuidas de hormigas en un entorno paralelo asíncrono", XXVI Conferencia Latinoamericana de Informática – CLEI'00, México, 2000.

[Bat94] Batali J., "Innate Biases and Critical Periods: Combining Evolution and Learning in the Acquisition of Syntax". *Proceedings of the Fourth Artificial Life Workshop* R. Brooks and P. Maes (eds). The MIT Press, 1994.

[Ber98] Bernal J., "Aplicación de algoritmos genéticos al diseño óptimo de sistemas de distribución de energía eléctrica". *Tesis doctoral*. Universidad de Zaragoza, Departamento de Ingeniería Eléctrica. España, 1998.

[BBR96] Barán B., Benítez D. y Ramos R., "Partición de Sistemas de ecuaciones para su resolución distribuida". *XXII Conferencia Latinoamericana de Informática – PANEL'96*, Santafé de Bogotá – Colombia, 1996.

[BC97] Barán B. y Chaparro E., "Algoritmos Asíncronos combinados en un Ambiente Heterogéneo de Red", *XXIII Conferencia Latinoamericana de Informática*, Valparaíso - Chile, 1997.

[BCA+96] Barán B., Cardozo F., Atlasovich J. y Schaerer C., "Solving the Point of Collapse Problem using a Heterogeneous Computer Network". *International Conference on Information Systems, Analysis and Synthesis – ISAS'96*. Orlando – Florida, Estados Unidos, 1996.

[BCC98] Barán B., Chaparro E. y Cáceres N., "A-Teams en la optimización del caudal turbinado de una represa hidroeléctrica". *Sixth Ibero-American Conference on Artificial Intelligence – IBERAMIA'98*, Lisboa – Portugal, 1998.

[BDT99a] Bonabeau E., Dorigo M. y Theraulaz T., *From Natural to Artificial Swarm Intelligence*. New York: Oxford University Press, 1999.

[BDT99b] Bullock S., Davis J. N., y Todd P. M., "Simplicity rules the roost: Exploring birdbrain parental investment heuristics". En D. Floreano, J.-D. Nicoud, F. Mondada (Eds.) *Fifth European Conference on Artificial Life (ECAL99)*, EPFL Lausanne, Switzerland, Heidelberg: Springer-Verlag, 1999.

[BHS97] Bäck T., Hammel U. y Schwefel H., "Evolutionary Computation: Comments on the History and Current State". *IEEE Transactions on Evolutionary Computation* – Estados Unidos, Vol. 1, No. 1, pp. 3-17, 1997.

[BHS99] Bullnheimer B., Hartl R. y Strauss C., "An improved ant system algorithm for the vehicle routing problem". *Annals of Operations Research* (Dawind, Feichtinger y Hartl (eds.): Nonlinear Economic Dynamics and Control, 1999.

[BKS97] Bullnheimer B., Kotsis G. y Strauss C., "Parallelization Strategies for the Ant System", *Reporte Técnico POM 9/97*, Universidad de Viena, Viena – Austria, 1997.

[BL99] Barán B. y Laufer F., "Topological Optimization of Reliable Networks using A-Teams". *Focus Symposium: "Arquitecture, Tools and Algorithm for Networks, Parallel and Distributed System"* en el marco de la World Multiconference on Systemics, Cybernetics and Informatics SCI'99, Orlando – Florida, Estados Unidos, 1999.

[BLP97] Bignone F. A., Livi R. y Propato M., "Symbolic and Global Dynamics in Coupled Genetic Networks: Evolution of Artificial Cell Populations at the Edge of and within Chaotic Transient Behaviour". *Fourth European Conference on Artificial Life*, Phil Husbands y Inman Harvey (Editores), MIT Press. 1997.

[BRB+99] Barán B., Rojas A., Britez D. y Barán L., "Measurement and Analysis of Poverty and Welfare using Fuzzy Sets". *International Conference on Systemics, Cybernetics and Informatics SCI'99*, Orlando - Florida, Estados Unidos, 1999.

[BS00] Barán B. y Sosa R., "A New approach for AntNet routing". *Ninth International Conference on Computer Communications and Networks.* Las Vegas, Nevada.2000.

[BSD00] den Besten M., Stützle T. y Dorigo M., "Ant Colony Optimization for the Total Weighted Tardiness Problem". *The Sixth International Conference on Parallel Problem Solving for Nature (PPSN-VI)*, Paris-Francia, 2000.

[BT89] Bertsekas D. Y Tsitsiklis J., *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, A Division of Simon & Schuster Englewood Cliff, New Jersey, 1989.

[Bur69] Burgin G., "`On playing two-person zero-sum games against nonminimax players". *IEEE Transactions on Syst. Sci. Cybern.*, Estados Unidos, Vol. SSC-5, No. 4, pp. 369-370, 1969.

[CD98] Di Caro G. y Dorigo M., "Mobile Agents for Adaptive Routing". *Proceedings of the 31st Hawaii International Conference on System*, Hawaii, 1998.

[CNP95] Calabretta R., Nolfi S., y Parisi D., "An Artificial life model for predicting the tertiary structure of unknown proteins that emulate the folding process". En F. Moran, A. Moreno, J.J. Merelo, and P. Chacon (Eds.), *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life*, Berlin Heidelberg, Springer Verlag, 1995.

[DAS97] Dengiz B., Altiparmak F. y Smith E., "Local Search Genetic Algorithm for Optimal Design of Reliable Networks". *IEEE Transactions on Evolutionary Computation*, Estados Unidos, Vol. 1, No. 3, pp. 179-188, 1997.

[DG96] Dorigo M. y Gambardella L., "A study of some properties of Ant-Q". *IV International Conference on Parallel Problem from Nature*, Berlin – Alemania: Springer-Verlag, pp. 656-665, 1996.

[DG97] Dorigo M. y Gambardella L., "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". *IEEE Transactions on Evolutionary Computation*, Estados Unidos, Vol 1, No. 1, pp. 53-66, 1997.

[DMC92] Dorigo M., Maniezzo V. y Colorni A., "An Investigation of some properties of an Ant algorithm". *Proceedings of the Parallel Problem solving from Nature Conference* (PPSN 92), Bruselas – Bélgica, 1992.

[DMC96] Dorigo M., Maniezzo V. y Colorni A., "The Ant System: Optimization by a colony of cooperating agents", *IEEE Transaction on Systems, Man, and Cybernetics*-Part B, Estados Unidos, Vol 26, No. 1, pp. 1-13, 1996.

[FS93] Freeman J. A. y Skapura D. M., *Redes neuronales: Algoritmos, aplicaciones y técnicas de programación*, Addison-Wesley Iberoamericana, S.A. y Ediciones Díaz de Santos S.A., Wilmington, Delaware, Estados Unidos, 1993.

[GBD+94] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R. y Sunderam V, "PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing". *Massachusetts Institute of Technology*, Estados Unidos, 1994.

[GD95] Gambardella L. y Dorigo M., "Ant-Q: A reinforcement learning approach to the traveling salesman problem" *12th International Conference on Machine Learning*. San Francisco, pp. 252-260, 1995.

[GD97] Gambardella L. y Dorigo M., "HAS-SOP: An Hybrid Ant System for the Sequential Ordering Problem", *Technical Report IDSIA11-97*, Lugano - Suiza, 1997.

[Gol89] Goldberg D.E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison – Wesley, Reading, MA, 1989.

[HB97] Hasteer G. y Banerjee P., "Simulated annealing based parallel state assigment of finite state machines". *IEEE International Conference on VLSI Design 1997 – VLSI in Multimedia Applications*. Bangalore – India, 1997.

[Her92] Herdy M., "Reproductive isolation as strategy parameter in hierarchically organized evolution strategies". *Parallel Problem Solving from Nature II, Amsterdam*, The Netherlandas: Elsevier, pp. 207 - 217, 1992.

[Hol75] Holland J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[Ign91] Ignizio J., *Introduction to Expert Systems: The Development and Implementation of Rule-Expert Systems*, McGraw-Hill, 1991.

[JM97] Johnson D.S. y McGeoch L.A., "The traveling salesman problem: a case study in local optimization", *Local Search in Combinatorial Optimization*, Eds. New York: Wiley: New York, 1997.

[HB90] Hwang K. y Briggs F., *Arquitectura de computadoras y procesamiento paralelo*. McGraw-Hill, Inc. Estados Unidos, 1990.

[KB96] Keller R. y Banzhaf W., "Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes". *Genetic Programming 1996: Proc. 1st Annu. Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., 1996.

[LMP+98] Lund. H., Miglino O., Pagliarini L., Billard A.y Ijspeert A., "Evolutionary Robotics - A Children's Game". *Proceedings of IEEE 5th International Conference on Evolutionary Computation.* IEEE Press. 1998.

[MC99] Maniezzo V. y Colorni A. "The Ant System applied to the Quadratic Assignment Problem". *IEEE Transactions on Knowdledge and Data Engineering*, Estados Unidos, 1999.

[MP96] Mareco C. y Paccanaro A., "Recognizing matching among terms: a neural networks approach". *XXII Conferencia Latinoamericana de Informática - PANEL'96*. Santafé de Bogotá - Colombia, 1996.

[MVO98] Martínez-Alfaro H., Valdez H. y Ortega J., "Using simulated annealing to minimize operational costs in the steels making industry". *IEEE International Conference on Systems, Man, & Cybernetics.* Estados Unidos, 1998.

[NAT96] Noroozian M., Andersson G. y Tomsovic K., "Robust, near time-optimal control of power system oscillations with fuzzy logic". *IEEE Transactions on Power Delivery* - Estados Unidos, Vol.11, No. 1, 1996.

[Ple95] Plemenos D., "Inteligencia Artificial y Sistemas Expertos". *Université de Limoges*, Francia, 1995.

[Rec94] Rechenberg I., Evolutionsstrategie'94. *In Werkstatt Bionik und Evolutionstechnik*. Stuttgart, Germany: Frommann-Holzboog, Vol. 1, 1994.

[Rut97] Rutowska J.C., "What's Value Worth? Constraining Unsupervised Behaviour Acquisition". *Fourth European Conference on Artificial Life*, Phil Husbands y Inman Harvey (Editores), MIT Press. 1997.

[Sch95] Schwefel H., *Evolution and Optimum Seeking*. New York: Wiley, (Sixth-Generation Computer Technology Series), 1995.

[SGL+00] Stützle T., Grün A., Linke S. y Rüttger M., "A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem". *The Sixth International Conference on Parallel Problem Solving for Nature (PPSN-VI)*, Paris – Francia, 2000.

[SH96] Stützle T. y Hoos H., "Improvements on the Ant System: Introducing Max-Min Ant System". *International Conference on Artificial Neural Networks and Genetic Algorithms*, Viena – Austria, 1996.

[SH97] Stützle T. y Hoos H., "Max-Min Ant System and Local Search for Combinatorial Optimization Problems". *2nd International Conference on Metaheuristics* – MIC97, Francia, 1997.

[SH99] Stützle T. y Hoos H., "Analyzing the Run-time Behaviour of Iterated Local Search for the TSP" *III Metaheuristics International Conference – MIC'99*, Angra dos Reis – Brasil, 1999.

[SR97] Sternberg M. y Reynolds G., "Using Cultural Algorithms to Support Re-Engineering of Rule-Based Expert Systems in Dinamic Performance

Environments: A Case Study in Fraud Detection". *IEEE Transactions on Evolutionary Computation*, Estados Unidos, Vol 1, No. 4, pp. 225-243, 1997.

[Stü98] Stützle T., "Parallelization Strategies for Ant Colony Optimization", *Proceedings of Parallel Problem Solving from Nature – PPSN-V*, Springer Verlag, Vol. 1498, pp. 722-731, 1998.

[SWP96] Solar M., Watkins F., y Perez P., "Red neural para reconocer minerales en una imagen digitalizada". *XXII Conferencia Latinoamericana de Informática – PANEL'96.* Santafé de Bogotá – Colombia, 1996.

[WSF89] Whitley D., Starkweather T. y Fuquay D., "Scheduling Problems and Travelling Salesman: the Genetic Edge Recombination Operator", *Procedure of the 3rd International Conference on Genetic Algorithm*, Morgan Kaufmann, 1989.

Referencias a Páginas Web

[Ava00] "Avalon". http://cnls.lanl.gov/avalon/. Los Alamos National Laboratory. Estados Unidos. 2000.

[BNW96] Bauer P., Nouak S. y Winkler R., "A brief course in Fuzzy Logic an Fuzzy Control", http://www.flll.uni-linz.ac.at/pdw/fuzzy/fuzzy.html. 1996.

[Cha98] Chapa Vergara S., "Lógica Difusa", http://delta.cs.cinvestav.mx/ red/logica/node195.html. México, 1998.

[Com1] "Complejidad Computacional", http://campus.mor.itesm.mx/~optamiza/ opti9901/capa2/complejidad.html.

[Dor98] Dorigo M., "Ant Colony System", http://iridia.ulb.ac.be/dorigo/ACO/ ACO.html. IRIDIA, Université Libre de Bruxelles, Belgium. 1998.

[Lie00] Liekens A., "Alife Online", http://alife.org/. 2000.

[Mel98] Melo A. I., "A través de feromonas los animales mantienen comunicación", http://www.uam.mx/organo-uam/documentos/V-II/ii31-12.html. 1998.

[Rei00] Reinelt G., "TSP", http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/TSPLIB.html. Universität Heidelber, Institut für Angewandte Mathematick, Germany. 2000.

[Her99a] De la Herrán Gascón M., "GAIA: Computación Evolutiva", http://www.aircenter.net/gaia/ce_c.htm. 1999.

[Her99b] De la Herrán Gascón M., "Notas sobre Computación Evolutiva" http://www.aircenter.net/gaia/ceno_c.htm. 1999

[Jim99] Jiménez F., "Vida Artificial", http://complex.us.es/~jimenez/CA/ac/node19.html. 1999.

[Mos98] Moscato P., "Memetic Algorithms'Home Pages", http://www.ing.unlp.edu.ar/cetad/mos/memetic_home.html.

[Orc1] Orcero D. S., "Programación Genética", http://www.biocristalografia.df.ibilce.unesp.br/irbis/disertacion/node215.html.

[Orc2] Orcero D. S., "Algoritmos Miméticos", http://www.biocristalografia.df.ibilce.unesp.br/irbis/disertacion/node224.html.

[Pao00] Di Paolo E., "Artificial Life Bibliography of On-line Publications", http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/alife.html. 2000.

[Top00] "TOP 500 Supercomputer". http://www.top500.org/lists/TOP500List.php3?Y=2000&M=06**.** Presentado por University of Mannheim y University of Tennessee. 2000.