Autor: Sergio R. Cherchyk

Profesor Tutor: Ing. Daniela Díaz

Instituto Universitario Aeronáutico

Ingeniería de Sistemas

Trabajo de Grado

"*Problema del Viajante de Comercio:*

Su implementación mediante un algoritmo de ruteo dinámico y aplicación de Optimización de Colonia de Hormigas"

Noviembre de 2016

Instituto Universitario Aeronáutico

Educación a Distancia

Ingeniería de Sistemas

Tema: Trabajo de Grado

Alumno: Sergio Román Cherchyk

Tutor: Ing. Daniela Díaz

1 Tabla de contenidos

Tabla de figuras

2 Introducción

El presente trabajo se basa en la experimentación empírica de meta-heurísticas empleadas en al ámbito de la Inteligencia Artificial del tipo de Colonia de Hormigas, para la resolución del problema del Viajante de Comercio (TSP) y en la implementación de algoritmos eficientes para la implementación de una aplicación que trabaje en la solución del mismo.

Para la implementación de las distintas instancias empleadas de este problema se emplea un algoritmo que, a diferencia de los métodos tradicionales empleados, permite la fácil diagramación y modificación de escenarios (es decir, de los grafos del problema), aportando una escalabilidad tal que facilita el trabajo con escenarios de gran número de nodos (tanto más es su ventaja cuanto mayor sea el escenario). Este algoritmo además, se adapta perfectamente a los problemas de TSP asimétrico, lo cual será verificado durante las pruebas de este trabajo.

Las mencionadas pruebas, además incluirán la verificación empírica de múltiples variables estocásticas que son el fundamento de estos métodos meta heurísticos.

Finalmente se incluirá y probará una variante original del método *AntSystem,* que consiste en una inicialización sesgada de algunas variables del algoritmo.

3 Descripción y Justificación

Para la resolución del problema del Viajante de Comercio se requiere contar con una matriz de distancias entre los distintos nodos de un grafo, lo que implica contar con una cantidad de datos igual a $(n^2)/2$ en el caso de un TSP simétrico de n^2 para un TSP asimétrico. Esto incrementa notablemente la carga de trabajo de recolección e introducción de datos a medida que el escenario del problema se agranda.

Como primera contribución en este trabajo se introduce un algoritmo que calcula dinámicamente las distancias entre cada nodo, contando solo con la información de las distancias entre nodos vecinos, con lo cual se consigue que el número de datos a manejar tenga incremento lineal del orden de *k*n/2*, donde *k* es una constante que refleja el promedio de rutas directas por nodo que se encuentran en el grafo (llamándose ruta directa a todo arco que une dos nodos sin pasar por uno intermedio).

Esta constante k es en general un número pequeño y se calcula mediante:

$$k = \sum_{x=1}^{n} r_{dir_x}/n$$

donde r_{dir_x} es el número de rutas directas que salen del nodo *x*, y *n* es el número de nodos. Esta fórmula es la misma para TSP simétrico y asimétrico, pero se espera que en un TSP asimétrico los r_{dir_x} sean mayores ya que entre dos nodos cualesquiera puede haber dos arcos, no ya solo uno. Para este trabajo se diagramaron dos escenarios con ciudades argentinas, en el primer caso 23 capitales, y en el segundo unas 100 ciudades elegidas *ad hoc*, quedando la constante k en el primer caso igual a 4,5 y en el segundo caso 2,9. Esto se comprende observando que el primer esquema abarca solo capitales, es decir, las

ciudades más grandes e importantes de cada provincia, de las cuales se espera que tengan una mayor cantidad de accesos directos.

Surge de esto entonces, que este encuadre del problema puede ser una herramienta útil en principio para quienes quieran abordar este tipo de problemas en particular, pero así también podría ser aprovechado en cualquier tipo de situación que requiera calcular distancias entre nodos de un grafo.

En segundo lugar, este método mencionado se ajusta perfectamente y a los problemas TSP asimétricos, lo cual le confiere una utilidad adicional.

Por último, en este trabajo más allá de probar las bondades del sistema antes mencionado, se aprovecha el mismo en la realización de un trabajo comparativo entre distintas meta-heurísticas de tipo optimización de colonia de hormigas (ACO, por sus siglas en inglés u OCH en su versión castellana), en las que se plantea la determinación empírica de la mejor combinación de los parámetros empleados en ellos, incluyendo la introducción de una variante original de uno de estos métodos. De esta forma esperamos que el trabajo contribuya a aportar información de referencia que pueda ser útil en el empleo subsiguiente de estos algoritmos en problemas similares.

4 Objetivos y alcances

Como **primer objetivo** de este trabajo nos proponemos probar un algoritmo de cálculo de distancias dinámico, su viabilidad y escalabilidad. Para esto se elaborarán dos problemas, según se ha mencionado, de distinta magnitud, para comparar las performances en la resolución de los mismos y comprobar la facilidad con la que se puede escalar o modificar un esquema dado. Se aclara que el primer problema (capitales de provincia), está incluido en el segundo (100 ciudades argentinas), por lo cual se tomó al primero como punto de partida y se lo escaló. Lo que esperamos demostrar es la facilidad de la adición de un ciudad extra (en comparación especialmente a el trabajo esperado usando los métodos usuales) y el funcionamiento sin problemas al incorporar asimetría en el esquema.

Como **segundo objetivo** nos proponemos encontrar combinaciones óptimas en los parámetros de las meta-heurísticas empleadas, las cuales luego, como **tercer objetivo** se emplearán para buscar soluciones que se compararán finalmente con el resultado obtenido utilizando un algoritmo de búsqueda voraz[1].

Como **objetivo final**, se introducirá una variante en una de las heurísticas (AS), con la intención de forzar una búsqueda más local cercana a la obtenida con el algoritmo Voraz, para estudiar su comportamiento comparándola con las otras meta-heurísticas empleadas y sacar conclusiones al respecto. Cabe hacer notar que esta variante propuesta no se ha

[1] El algoritmo de Búsqueda Voraz (Greedy algorythm) propone, en un grafo dado, elaborar un recorrido Hamiltoniano partiendo de uno de sus nodos, y continuando hacia el más cercano disponible hasta completar el circuito.

encontrado documentada en toda la bibliografía consultada, por cuanto es original y no se tiene antecedentes de cuáles puedan ser los resultados obtenidos.

Los parámetros a probar, que serán descriptos más en profundidad más adelante, y su descripción se listan en la tabla 1:

α	Peso del índice de feromonas
β	Peso de la heurística
ρ	Índice de persistencia de las feromonas
δ	Coeficiente de PTS
P_{best}	Probabilidad de construir la mejor solución encontrada
κ	Constante de inicialización de feromonas

Tabla 1- descripción de parámetros estocásticos

Estos parámetros se probarán dentro de distintos rangos según cada meta-heurística empleada, tomando como base estudios empíricos comparativos previos y adaptándolos a estos problemas en particular.

Los parámetros se variaran de a uno para mantener una condición *ceteris paribus*, y para cada combinación de parámetros se realizaran 20 ensayos de 1000 iteraciones cada una para el caso de AS[2] y 10 de 1000 para MMAS[3].

[2] Ant System

[3] MinMax Ant System

Finalmente, una vez encontradas las mejores combinaciones se realizaran corridas con esas combinaciones.

Los rangos de variación propuestos para cada parámetro se muestran en la *Tabla 2*:

Parámetro	Valor mínimo	Valor máximo	Incremento
Para AS			
α	1	1	-
β	1	6	1
ρ	0,3	0,7	0,1
Para MMAS			
α	1	1	-
β	2	5	1
ρ	0,58	0,98	0,05
δ	0	1	0,5
P_{best}	0,005	0,5	Intermedio:0,05

Tabla 2- Rangos de variación de los parámetros bajo estudio

5 Marco teórico

5.1 El problema del viajante de comercio

5.1.1 Descripción del Problema

El problema del Viajante de Comercio (TSP, por sus siglas en inglés), ha sido largamente difundido y estudiado en el campo de las ciencias de la computación. Este problema toma su nombre de aquellos días en que los viajantes de comercio, valija en mano, recorrían distintos pueblos en busca de potenciales clientes, tratando en lo posible de minimizar el costo de su recorrido: cuánto más corto, en la mayoría de los casos, mejor.

Enunciándolo más formalmente, en TSP se nos pedirá hallar un circuito *Hamiltoniano* (cerrado) atravesando *n* ciudades, pasando una vez por cada una de ellas. Esto constituye siempre un problema sin un método general para su solución del tipo *NP-hard*[4]. Hallar entonces una solución óptima requiere una búsqueda exhaustiva de todos los posibles recorridos, lo cual es solo posible en problemas muy acotados, dado que la cantidad de combinaciones es del orden *O(n!)*, siendo n el número de nodos. En consecuencia, un recorrido por solo 10 ciudades requeriría considerar más de 3 millones de combinaciones, mientras que si fuese de sólo 20 se volvería impracticable (20! = 2432902008176640000),

[4]Los problemas NP-duros son problemas que tienen *al menos* la dificultad de un problema NP (nondeterministic polynomial).

y en el caso de 100, que será nuestro escenario más complejo, el número es del orden de 10^{137}.

Matemáticamente, los TSP pueden representarse por medio de un grafo, en el que las locaciones son los nodos y los pesos en los arcos representan las distancias entre estas. La meta es encontrar el recorrido que termine donde empezó, recorriendo todos los nodos, y para el cual la suma de los arcos sea menor.

En este trabajo utilizaremos distintos escenarios basados en ciudades argentinas, con sus distancias reales[5]. Los escenarios son tres, uno de sólo cuatro ciudades, utilizado con fines de testeo y de demostración durante el presente trabajo, uno intermedio con las 23 capitales provinciales del país, utilizado para las pruebas iniciales de referencia, y finalmente un escenario de 100 ciudades a lo largo y a lo ancho de la República.

En la fig.1 se muestra el grafo con cuatro capitales provinciales.

Cabe aclarar en principio que el grafo no constituye un mapa de rutas: los arcos solo muestran la distancia y no muestran la forma real que puede tener una ruta, por lo que la longitud de los mismos puede no estar en proporción con la distancia que representan. De igual modo, para fines prácticos hemos usado grafos en los que si bien las ciudades se encuentran posicionadas de manera aproximada a su ubicación en real, en varios casos se han acercado o alejado los nodos deliberadamente para facilitar su legibilidad (Nota: la interfaz de la aplicación desarrollada para este trabajo si permite ver el grafo final y el recorrido con las ciudades ubicadas de acuerdo a su posición geográfica precisa).

[5] Obtenidas de la página Ruta0.com

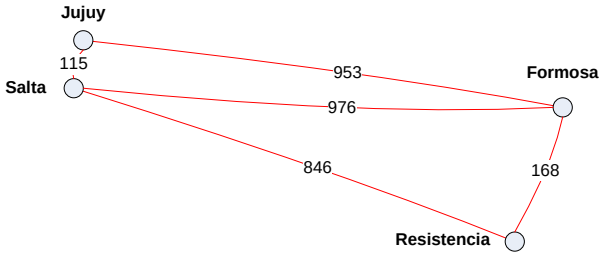

Fig. 1 – Grafo Test

Existen dos tipos de TSP, asimétricos y simétricos. En los simétricos las distancias entre nodos son idénticas en ambos sentidos, mientras que en los asimétricos estas pueden variar. Estos últimos casos contemplan las posibilidades de caminos en un solo sentido, que en la escala de nuestros escenarios pierde sentido real ya que en esta escala casi siempre las rutas son bidireccionales, y si hay alguna diferencia en las distancias en un sentido u otro, esta diferencia es despreciable casi siempre para fines prácticos.

5.1.2 Usos del TSP

El TSP es un problema teórico que resulta de una idea de la vida real (de ahí su nombre) que por ende puede aplicarse directamente a situaciones de ruteo similares a las que lo motivaron: recorrido de vendedores, reparto de correo, recolección de residuos etc. Más aún, este problema puede adaptarse a la resolución de muchas otras situaciones, he aquí algunos ejemplos para ilustrar esta afirmación, con el propósito de reforzar la argumentación sobre la potencial utilidad de este trabajo, que siendo eminentemente teórico, se relaciona con los usos prácticos de maneras que suelen no ser tan evidentes:

1. **El problema de la programación de impresión**: Una de las mayores y principales aplicaciones de un TSP surge en la programación de una imprenta de un diario con múltiples ediciones. Asumamos como ejemplo que existen cinco pares de cilindros de entre los cuales pasan los rollos de papel y por los cuales ambos lados de una página se imprimen de forma simultánea. Existen además tres formatos: de 4, 6 y 8 páginas, que se utilizan para imprimir las ediciones. El problema de programación consiste en decidir qué formato entrará en cada corrida de la imprenta y la longitud de cada una de estas. Usando el vocabulario del TSP, los costos de cambio de placa son las distancias interurbanas. Existen estudios rigurosos para esta aplicación, para más detalles se puede consultar los trabajos de *Gorenstein* (1970) y *Carter & Ragsdale* (2002) como referencia.
2. **Problema de enrutamiento de micros escolares**: *Ángel et al*, (1972) investigó el problema de la programación de los micros como una variante del TSP con algunas limitaciones adicionales. El objetivo de la programación es la obtención de un patrón de carga del micro de tal manera que el número de rutas se reduzca al mínimo, la distancia total recorrida por todos los micros se mantenga en el mínimo, no haya micros sobrecargados y el tiempo necesario para atravesar cualquier ruta no exceda los máximos permitidos por las políticas establecidas.
3. **Problema de programación de entrevistas**: *Gilbert y de Hofstra,* (1992) encontraron que la **a**plicación TSP puede aplicarse a la programación de las entrevistas entre los corredores turísticos y proveedores de la industria del turismo, que tiene variaciones entre distintos períodos estacionales. A cada corredor le corresponde un vendedor que debe visitar un conjunto de los puestos de vendedores, que están representados por un conjunto de ciudades.
4. **Problema de programación de laminado en caliente**: En la industria del hierro y el acero los pedidos están programados en los altos hornos de tal manera que el costo total de configuración durante la producción puede ser minimizado. Detalles de una aplicación reciente que modela estos problemas pueden encontrarse en un trabajo de *Tang et al*., (2000). En el mismo, las órdenes son tratadas como ciudades y la distancia entre dos ciudades equivale al precio de penalización por el cambio del esquema de producción entre dos órdenes. La solución del modelo permitirá una programación de los hornos.
5. **Problema de planificación de la misión**: El problema de planificación de la misión consiste en determinar un camino óptimo para cada soldado (planificador) para llevar a cabo los objetivos de la misión en el mínimo tiempo posible. El planificador de la misión utiliza una variación del plan estratégico donde hay n planificadores, m objetivos que deben ser

visitados por algunos planificadores, y una ciudad base a la que todos los planificadores deben eventualmente volver. La aplicación de TSP en el planeamiento la misión ha sido estudiado por *Brummit y Stentz*, (1996 y 1998) y *Yu et al*, (2002). Del mismo modo, los problemas de enrutamiento que surgen en la planificación de vehículos aéreos no tripulados (drones), también se pueden modelar como TSP y fueron analizados por *Ryan et al*., (1998).

5.1.3 Métodos típicos de resolución: la matriz de distancias

El primer paso en un resolución típica de un TSP es establecer las distancias entre cada uno de los nodos, lo cual se puede guardar en una matriz *n x n*. Por ejemplo, para el grafo de la Fig1. Tenemos la siguiente matriz resultante:

	Jujuy	Salta	Formosa	Resistencia
Jujuy	x	115	953	961
Salta	115	x	976	846
Formosa	956	976	X	168
Resistencia	961	856	168	x

Tabla 3 – Matriz de distancias para 4 ciudades

En casos como el actual, donde el TSP es simétrico, la cantidad de entradas puede ser reducida a $(n^2)/2$, pero si el TSP es asimétrico deberemos conservar la matriz completa.

Se aprecia que para un grafo tan sencillo la cantidad de datos con la que contamos es completamente manejable y el hallazgo de una solución ideal es muy simple. Pero, como se mencionó anteriormente, las cosas empeoran a medida que se aumentan los nodos, en orden factorial para la cantidad de recorridos posibles, y en orden exponencial (cuadrático) para la cantidad de rutas a considerar.

5.1.4 Método de enrutamiento dinámico

Un forma de eliminar los inconvenientes planteados con el uso del método anterior cuando el número de nodos es muy grande, puede conseguirse con el empleo de algún método por el cual se pueda calcular las distancias entre cada nodo contando solo con la información de las distancias entre nodos adyacentes. Este tipo de paradigma es similar al del ruteo dinámico en redes de comunicaciones. Protocolos de enrutamiento tales como el **RIP** o el **OSPF** pueden aprender los mejores caminos entre nodos mediante el intercambio de información entre enrutadores conectados directamente. Uno de los protocolos más sencillos empleado para el ruteo dinámico en redes **IP** es el **RIP**. Este protocolo, así como su versión mejorada el **RIP2**, son protocolos de **vector distancia**, ya que eligen el mejor camino basados en la ruta con el menor número de saltos entre origen y destino. Básicamente el **RIP** funciona enviando oportunamente sus tablas de rutas a todos sus vecinos, en la que indica los destinos que este conoce y su distancia, en saltos, a los mismos. Con esta información cada encaminador va reconfigurando su propia tabla de rutas, y luego de un tiempo de intercambio, cada nodo termina con un conocimiento completo de la red, alcanzando un estado que se denomina de *convergencia*.

Siguiendo este mismo concepto, y agregando la información de la distancia en kilómetros entre cada nodo adyacente, el algoritmo empleado para el cálculo de la matriz de distancias emplea información de rutas directas entre ciudades (llamándose directos a aquellos caminos entre dos nodos en los que no hay ningún otro nodo en el medio y son a su vez la ruta más corta posible entre estos dos nodos). Esta información se guarda en una base de datos y es consultada al arrancar el programa, luego de lo cual cada nodo, secuencialmente, envía su tabla de ruteo a cada vecino. Una vez finalizado el intercambio entre todos los nodos, cada uno calcula las nuevas rutas y comienza una nueva iteración. Este proceso se detiene cuando en una iteración, después de los intercambios, ningún nodo

encuentra ningún mejor camino a los que ya tiene para actualizar su propia tabla de ruteo, con lo que se considera que hemos alcanzado el estado de convergencia del grafo.

Veamos esto usando como ejemplo el grafo sencillo de cuatro nodos de la Fig. 1. En este caso tenemos las siguientes tablas de rutas:

Desde Jujuy	
Destino	**Distancia**
Salta	115
Formosa	953

Desde Formosa	
Destino	**Distancia**
Jujuy	953
Resistencia	168
Salta	976

Desde Resistencia	
Destino	**Distancia**
Salta	846
Formosa	168

Desde Salta	
Destino	**Distancia**
Jujuy	115
Formosa	976
Resistencia	846

Tabla 4

Nótese que en este esquema tan sencillo nos encontramos desde el inicio con dos nodos que ya tienen rutas directas hacia todos los demás nodos, condición que se va tornando menos probable a medida que agregamos más nodos al esquema. En este caso, tanto Jujuy como Resistencia reciben la información suficiente como para calcular sus mejores rutas posibles, por lo que luego se llega a la convergencia en la primera iteración y será en la segunda cuando luego de intercambiar información con sus vecinos ninguno de los nodos

tenga posibilidad de mejorar su propia tabla y el proceso se detendrá con este estado de las tablas:

Desde Jujuy	
Destino	**Distancia**
Salta	115
Formosa	953
Resistencia	961

Desde Formosa	
Destino	**Distancia**
Jujuy	953
Resistencia	168
Salta	976

Desde Resistencia	
Destino	**Distancia**
Salta	846
Formosa	168
Jujuy	961

Desde Salta	
Destino	**Distancia**
Jujuy	115
Formosa	976
Resistencia	846

Tabla 5

Como se mencionó anteriormente, con esto se logra reducir enormemente la cantidad de información a manejar de manera manual cuando el número de nodos se incrementa. Consideremos para ejemplificar el grafo de la figura 2:

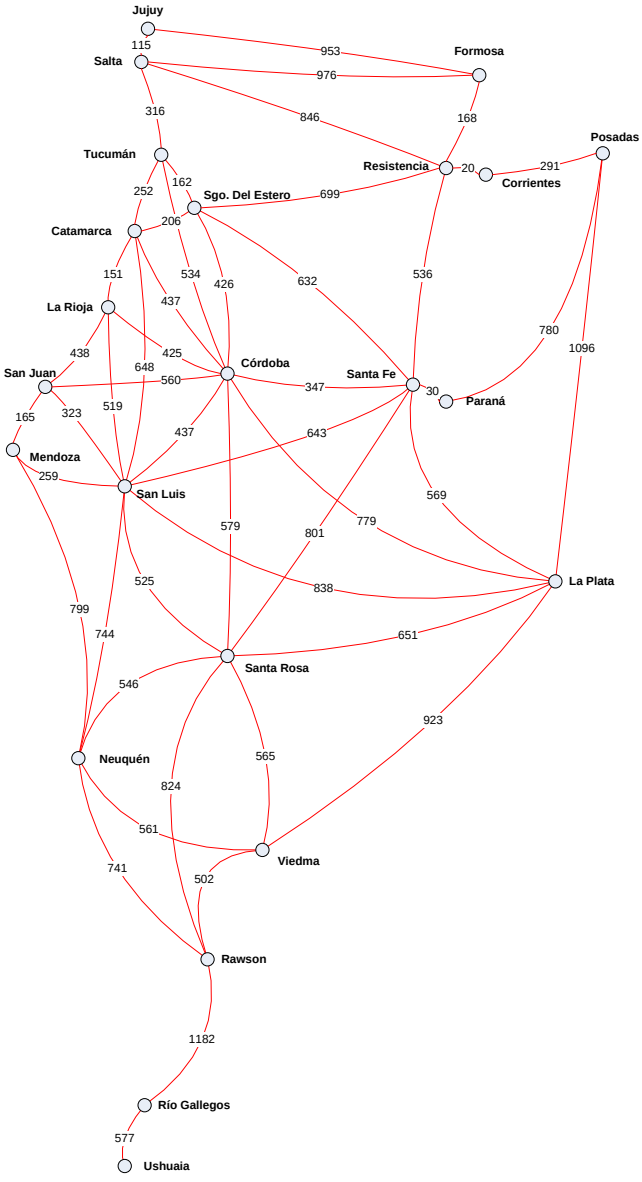

Fig. 2 – Grafo de capitales provinciales

Este es el grafo de distancias de rutas terrestres entre las capitales provinciales argentinas[6].

Los arcos en rojo representan la información ingresada, es decir que tuvimos que obtener y cargar 104 distancias (4,5 por nodo, número que configura la constante *k* mencionada al principio del presente documento), por otro lado, de haber tenido que cargar las distancias entre todos los nodos del grafo, tendríamos que haber manejado 264 distancias. Vemos entonces que ya en una escala reducida hay ventajas evidentes, pero estas se hacen verdaderamente notables en una escala de cien nodos, como la que intentaremos utilizar, en donde la relación entre ambos paradigmas se hace de 230/5.000.

Más aún, si incluyésemos aunque más no sea una sola ruta asimétrica, esto implicaría, en el caso de que estemos usando una matriz de *n x n*, que ya deberíamos calcular nuevamente cada una de las rutas y que deberíamos considerar la matriz completa de 10.000 entradas ya que no habría forma de establecer *a priori* cuáles rutas serán simétricas y cuáles no. Con el ruteo dinámico, por otro lado, solo deberíamos agregar una nueva distancia de un nodo a otro y, que será distinta a la distancia que ya tiene en su tabla de rutas hacia x[7]. Como el algoritmo no trata de actualizar las rutas hacia ciudades adyacentes no ocurrirá que al recibir una distancia a su vecino, menor a la que conoce actualmente, actualice la tabla rompiendo la asimetría y generando información errónea.

---

[6] Nótese que, en primer lugar, el grafo no es un mapa de rutas y las líneas rojas representan distancias sin pretender representar el recorrido de los caminos reales, y en segundo lugar, si bien se trató de ubicar cada nodo de acuerdo a la posición relativa entre las ciudades, las escalas se han alterado deliberadamente por razones prácticas. La representación del grafo con el recorrido hallado, que proporciona la aplicación usada, por otro lado, representa las distancias en una escala real.

[7] Como se verá más Adelante, para implementar esto deberemos indicar además el sentido de circulación de cada ruta, que por defecto en el sistema es bidireccional.

5.1.5 Resolución por métodos heurísticos

Teniendo en cuenta que, como se mencionó anteriormente, la resolución de problemas de este tipo, intentando hallar una solución óptima mediante la fuerza bruta se vuelve rápidamente impracticable a medida que se incrementa el número de nodos a considerar, los métodos de resolución se abocan a encontrar una solución lo suficientemente buena en un tiempo razonable, empleando distintas heurísticas y meta-heurísticas.

Un método heurístico muy empleado, dada su sencillez, rapidez de aplicación y calidad de su solución, especialmente para una cantidad pequeña de nodos, es el algoritmo voraz (*greedy algorithm*). Los algoritmos voraces se basan en una toma de decisiones basada en información disponible sin importar las consecuencias futuras de esta decisión, en otras palabras se busca siempre la *elección local* óptima sin preocuparse por la *solución global* que derivará de esta. En el caso del TSP, lo que se hace es, desde cada nodo, se elegirá el camino más corto disponible hacia cualquier otro nodo, y se continuará sucesivamente hasta completar el circuito. Es evidente que este método dará soluciones distintas para cada grafo según sea el nodo que se elija como punto de partida. Esto será así porque los nodos por los que se pasa al principio tienen más rutas disponibles de las cuales elegir. De esta forma, para obtener la mejor solución posible con este algoritmo, se parte de cada una de los nodos alternativamente, y finalmente se escoge la mejor solución encontrada.

Con esta heurística, salvo para espacios de soluciones muy acotados o casos particulares poco probables, la solución encontrada es sub-óptima y corresponde a un óptimo local (o cercano a este), que tenderá a alejarse más del óptimo global a medida que el número de nodos a considerar se incremente.

5.2 Metaheurística de optimización por colonia de hormigas

Las metaheurísticas se valen de algoritmos heurísticos para la resolución de problemas complejos, agregando una serie de variables estocásticas, que pueden ser ajustadas de manera empírica, buscando la mejor combinación de ellas que se ajuste a un problema específico. En el transcurso de este trabajo se buscarán las mejores combinaciones para trabajar con los métodos propuestos, que son distintas variantes de la Optimización por Colonia de Hormigas, incluyendo una variante del método *Ant System* que incluye una utilización particular de la heurística Voraz.

5.2.1 Descripción de la Optimización por Colonia de Hormigas

Las hormigas reales son capaces de encontrar el camino más corto entre una fuente de alimento al hormiguero sin el uso excesivo de señales visuales (Hölldobler y Wilson, 1990). Además, son capaces de adaptarse a los cambios en el medio ambiente, por ejemplo, encontrar un nuevo camino más corto una vez que el viejo ya no es factible debido a un nuevo obstáculo (Beckers, Deneubourg y Goss, 1992; Goss, Aron, y Deneubourg Pasteels, 1989). En la figura 3a se grafica cómo las hormigas se mueven en una línea recta que conecta su alimento al hormiguero, si no existe entre estos dos puntos ningún obstáculo o accidente del terreno que impida esta situación

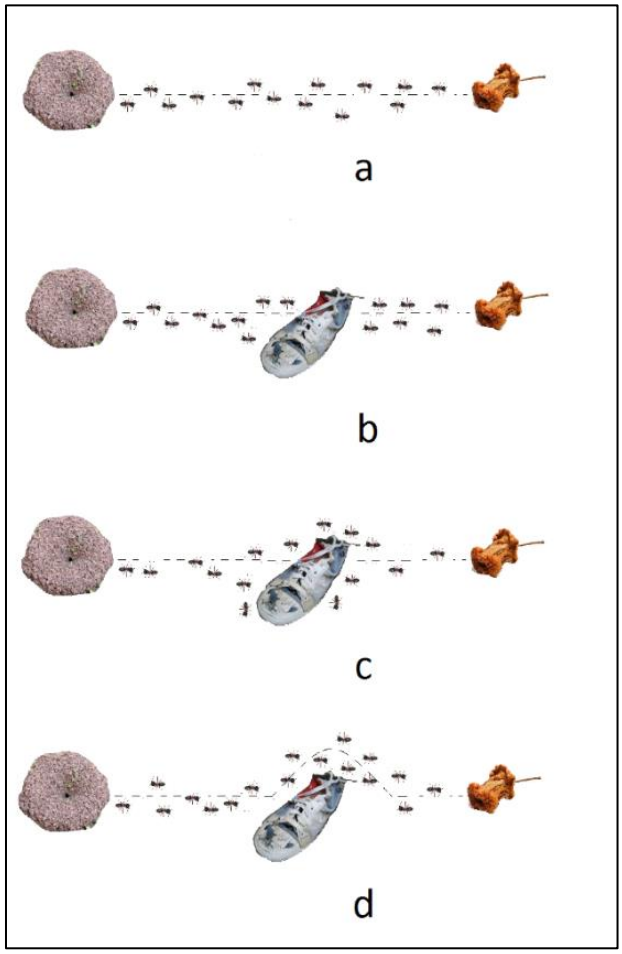

Fig. 3 – Comportamiento de una colonia de hormigas

Es sabido que el medio primario para que las hormigas puedan mantener esta línea es un rastro de feromonas. Las hormigas depositan una cierta cantidad de feromona al caminar, y cada hormiga probabilísticamente preferirá seguir una dirección rica en feromonas. Este comportamiento elemental de las hormigas reales se pueden utilizar para explicar cómo pueden encontrar el camino más corto que reconecta línea interrumpida después de la

aparición repentina de un obstáculo inesperado en su trayectoria inicial (Figura 3b). De hecho, una vez que el obstáculo ha aparecido, esas hormigas que están justo en frente al obstáculo no pueden continuar siguiendo el rastro de feromona y no les queda más opción que elegir entre un giro a la derecha o a la izquierda. En esta situación podemos esperar que la mitad de las hormigas elijan un giro a la derecha y la otra mitad gire a la izquierda. Una situación similar se puede encontrar en el otro lado del obstáculo (figura 3c). Es interesante notar que esas hormigas que eligen, por casualidad, la ruta más corta alrededor del obstáculo serán las que más rápidamente reconstruyan el rastro de feromona interrumpido, en comparación con las que elijan el camino más largo. Por lo tanto, el camino más corto recibirá una mayor cantidad de feromona por unidad de tiempo y, a su vez un mayor número de hormigas comenzará a elegir el camino más corto. Debido a este proceso de retroalimentación positiva (autocatalítico), todas las hormigas elegirán rápidamente el camino más corto (Fig.3d). El aspecto más interesante de este proceso autocatalítico es que encontrar el camino más corto alrededor del obstáculo parece ser una propiedad emergente de la interacción entre el comportamiento distribuido de la colonia de hormigas y la forma de los obstáculos: aunque todas las hormigas se mueven aproximadamente a la misma velocidad y depositan un rastro de feromonas aproximadamente al mismo ritmo, es un hecho que se tarda más en eludir los obstáculos por su lado más largo que sobre su lado más corto lo cual hace que el rastro de feromona se acumule más rápido en el lado más corto. Además, a causa de la preferencia de las hormigas por los niveles más altos del rastro de feromonas es posible que esta acumulación se potencie.

Se puede implementar un proceso similar en un ambiente simulado habitado por hormigas artificiales en la resolución del TSP ya descripto (*Dorigo et al, 1997*)[8]. En un mecanismo

[8] Ant colonies for the traveling salesman problem (Dorigo et al. - Université Libre de Bruxelles, Bélgica).

básico de optimización de colonias de hormigas (que no es precisamente el empleado en este trabajo) una hormiga artificial es un agente que se mueve de ciudad en ciudad en un grafo de un TSP (para nuestra implementación, este grafo representa el territorio argentino). Cada hormiga escoge la ciudad de destino usando una función probabilística que depende tanto de la feromona acumulada en cada arco del grafo como de un valor heurístico, que en este caso es función de la distancia entre las ciudades. Las hormigas artificiales preferirán probabilísticamente las ciudades que están conectados por los arcos con una gran cantidad de feromona y están más cerca. Inicialmente, las hormigas artificiales son colocadas en ciudades seleccionadas al azar. Para cada intervalo de tiempo dado, en simultáneo, las hormigas se trasladan a las nuevas ciudades y modifican el rastro de feromona en los arcos, lo que se conoce como actualización local de la feromona. Cuando todas las hormigas han completado un recorrido, la hormiga que hizo el recorrido más corto modifica los arcos pertenecientes a su recorrido (lo que se llama actualización global de la feromona) mediante una cantidad de feromona que es inversamente proporcional a la longitud de su recorrido.

En nuestra colonia de hormigas artificiales se han replicado tres de los comportamientos de las hormigas reales: (i) la preferencia por los caminos con un alto nivel de feromona, (ii) la mayor tasa de crecimiento de la cantidad de feromona en los caminos más cortos, y (iii) la comunicación entre hormigas mediada por rastro. Las hormigas artificiales fueron también provistas con unas pocas capacidades que no tienen en su contraparte natural, pero que se ha observado que se adaptan bien a la aplicación de un TSP: las hormigas artificiales pueden determinar a qué distancia están las ciudades y están dotadas de una memoria de trabajo que les permite recordar qué ciudades han ya sido visitadas (esta memoria de trabajo se vacía al comienzo de cada nuevo ciclo, y se actualiza después de cada movimiento añadiendo la nueva ciudad visitada).

El modelo descripto, utilizado por *Dorigo* en sus aplicaciones al TSP, se basa en investigaciones de *Deneouburg et al*. sobre el comportamiento de colonias de hormigas

reales, quienes sugirieron el primer modelo estocástico simple de colonia de hormigas basado en los comportamientos observados:

Sea $P_{i,a}$: la probabilidad de que una hormiga llegue a la intersección *i* y seleccione la rama *a* en un instante *t*.

$$P_{i,a} = \frac{\left[k + \tau_{i,a}\right]^{\alpha}}{\left[k + \tau_{i,a}\right]^{\alpha} + \left[k + \tau_{i,a'}\right]^{\alpha}}$$

donde $\tau_{i,a}$ es la concentración de feromona en la rama *a* de la intersección *i*, $\tau_{i,a'}$ es la concentración de feromona en la otra rama de la intersección *i*, α es un parámetro estocástico que representa el peso de la concentración de feromonas ebn la elección y *k* es el tiempo que se tarda en atravesar la rama *a*.

Basada en este modelo, la regla probabilística más empleada en los esquemas de hormigas artificiales es la siguiente:

$$P_k(r,s) = \begin{cases} [\tau_{rs}]^{\alpha}[\eta_{rs}]^{\beta}\left(\sum_{u \in J_k(r} [\tau_{ru}]^{\alpha}[\eta_{ru}]^{\beta}\right)^{-1}, & \text{si } s \in J_k(r) \\ 0, & \text{en otro caso} \end{cases}$$

donde τ_{rs} es la feromona del arco a_{rs}, η_{rs} es la información heurística del arco a_{rs}, α y β son pesos que establecen la importancia relativa entre la información heurística y los niveles

de feromona y J_r representa el conjunto de nodos alcanzables desde *r* y no visitados aún por la hormiga *k*.

Los parámetros son determinados de manera experimental. Para su comprensión debemos tener en mente los siguientes criterios. Si α tiende a 0, se elegirán los nodos de mayor valor heurístico, dando lugar a un algoritmo voraz típico. Si por otro lado β tiende a 0, los recorridos de las hormigas se aglomerarán rápidamente (situación de estancamiento) dando lugar a soluciones subóptimas, en general muy pobres.

Para modelar la forma en que las hormigas dejan su rastro de feromonas, hay que emular la forma en como estas van impregnando los caminos por los que pasan y cómo esta se evapora con el correr del tiempo. La forma más sencilla y que copia de manera más realista el comportamiento natural, consiste en hacer avanzar a las hormigas simultáneamente y a medida que lo hacen van depositando una cantidad similar de feromona.

A diferencia de las hormigas reales, las artificiales tienen la posibilidad de impregnar más fuertemente los mejores caminos encontrados, lo cual se aprovecha de diversas formas según el tipo de optimización que se elija. Esto puede lograrse si todas las hormigas actualizan los niveles de feromonas de los arcos por los cuales pasaron cuando ya han completado su recorrido, de manera que pueden tener una idea de la calidad de su solución. Una solución básica, empleada en la optimización *Ant System* (AS) que se ha usado en este trabajo, consiste en ir calculando los recorridos de las hormigas de a uno, de manera secuencial, y al final del mismo depositar una cantidad de feromona que es inversamente proporcional al recorrido total de esa hormiga.

La evaporación de feromona es usada para impedir una acumulación descontrolada de la misma y por otro lado esto permite olvidarse de las malas decisiones tomadas. Este porcentaje es igual en cada arco y se controla mediante el parámetro $0 \leq \rho \leq 1$, que representa el coeficiente de persistencia de la feromona, siendo la velocidad de evaporación igual 1 - ρ.

El modelo de actualización de feromona en AS viene dado por:

$$\tau_{rs}(t) = \rho.\tau_{rs}(t-1) + \sum_{k=1}^{m} \Delta t_{rs}^{k}$$

donde $\Delta \boldsymbol{t}_{\boldsymbol{rs}}^{\boldsymbol{k}}$ es la cantidad de feromona depositada por la hormiga k en cada arco visitado

$$\Delta t_{rs}^{k} = \begin{cases} \dfrac{1}{C(S_k)}, & \text{si la hormiga vistó el arco } a_{rs} \\ 0, & \text{en otro caso} \end{cases}$$

$C(S_k)$ es el costo de la solución generada por la hormiga *k*, lo cual en nuestro caso significa la longitud del recorrido seguido por la misma, *m* es el número de hormigas y *ρ* es el coeficiente de persistencia de la feromona. Nótese entonces que los arcos prometedores, que esperamos sean visitados por muchas hormigas recibirán aportes extra de feromonas, mientras que en los no visitados el nivel de feromona se irá desvaneciendo hasta eventualmente tornarse imperceptible. Además, las hormigas que realicen circuitos más cortos, depositarán más feromona en sus propios arcos, reforzando la realimentación positiva sobre los mejores recorridos encontrados.

Del el modelo AS, que el más sencillo usado en este trabajo y uno de los primeros propuestos por Dorigo en 1997, se propusieron tres variantes: *ant-density, ant quantity* y *ant cycle*. En las dos primeras, más primitivas, la actualización de feromona se realiza luego de que cada hormiga pase por cada arco (movimientos simultáneos). La última, que es una de las que se han empleado en este trabajo, se realiza después de haber completado todos los recorridos, lo cual nos permite reforzar las mejores soluciones halladas y no solamente marcar los arcos más visitados. Se ha elegido este sistema en particular, en esta variante, porque habiendo sido la base de sistemas posteriores mejorados, es aun suficientemente buena para ser tomada en consideración como referencia, es tal vez la más sencilla de implementar y se ha probado abundantemente en casi dos décadas.

El algoritmo de la AS es el siguiente:

T : número de iteraciones que ejecutará el algoritmo,

m : número de hormigas,

n : número de nodos,

L[h] : lista de los nodos visitados (camino) por la hormiga *h*,

$C(S_h)$: costo de la solución generada por la hormiga *h*,

distancia_arcos(L[h]): es la suma de las distancias de los arcos en *L[h]*

```
/* Inicialización de parámetros */
Para i=1 hasta n hacer
     Para j=1 hasta n hacer
          τij = τ0 (inicialización de feromonas en los arcos)
Desde t=1 hasta T hacer
     Desde h=1 hasta m hacer
          L[h][1] = nodo_inicial (dónde empiezan las hormigas)
```

```
/* Construcción de soluciones por las hormigas */
Desde i=2 hasta n hacer
      Desde h=1 hasta m hacer (para cada hormiga)
            L[h][i] ← regla_transición (J(L[h][i-1]), τ, η)

/* Actualización de feromona */
Desde h=1 hasta m hacer (para cada hormiga)
      C(Sh) = distancia_arcos(L[h])
      mejorsol = arg min_h{C(Sh)}
Desde i=1 hasta n hacer
      Desde j=1 hasta n hacer
            τij(t)= ρ.τij(t-1) + ∑_{i=1}^{m} Δt_ij^k

/* Actualización de mejor solución visitada*/
Si (C(mejorsol) < C(global)) C(global) = C(mejorsol)
Devolver global
```

En cuanto a la inicialización de los valores de feromona, si estos se eligen muy bajos, las primeras búsquedas estarán muy influenciadas por los primeros recorridos que estarán muy guiados por la heurística. Esto concentraría la búsqueda alrededor de un posible mínimo local y se estancará pronto en esa región del conjunto de soluciones. Esto, si bien debería generar soluciones subóptimas podría servir como método para encontrar, de manera muy rápida y sencilla, ya que se requerirían pocas iteraciones, una mejora de la solución obtenida con una heurística voraz. Para probar esto, en este trabajo he desarrollado un método que llamaremos *Combinado*, en el cual en lugar de disminuir el valor inicial de feromona para dar más peso a la heurística, se probará marcando más fuertemente los arcos de una solución del algoritmo voraz, forzando la búsqueda local pero sin restringirla del todo al patrón original obtenido por este método heurístico. El objetivo es probar si se puede obtener una mejora local y qué tan rápido se puede lograr. [9]

Por otro lado, si las feromonas se inicializan en valores muy altos, deberán pasar muchas iteraciones para que la evaporación y los nuevos depósitos empiecen a revelar patrones diferenciados que puedan influir de manera significativa en las búsquedas. Una solución posible, y que es la más extendida en los modelos de AS ya que ha sido probada con éxito en muchos estudios, es la de impregnar cada arco del grafo con un valor de feromona igual o apenas superior al depositado por las hormigas en una iteración. Como este valor, según se mencionó, está determinado por la calidad de la solución del recorrido (más corto el recorrido, más feromona se deposita), se toma como referencia el recorrido obtenido empleando la heurística voraz, de manera que, entonces, la feromona inicial en cada arco viene determinada por:

[9] Nótese que reforzar el patrón producido por la heurística voraz no es lo mismo que reforzar el peso de la heurística intrínseca del AS.

$\tau_{ij} = \tau_0 = m/C^{nn}$

donde m es el número de hormigas y C^{nn} es la longitud del recorrido obtenido con heurística voraz. En el caso del método combinado, los arcos pertenecientes al recorrido global heurístico (voraz) se impregnaran con un valor inicial de $k\ \tau_0$, siendo k un nuevo parámetro estocástico cuyo valor ideal, si existe, intentaremos determinar experimentalmente.

Para las condiciones de parada del programa se deberían tomar en cuenta cuestiones tales como un número máximo preestablecido de iteraciones, tiempo total, calidad de la solución o estancamiento de la misma. Estas dos últimas condiciones requieren de ciertos conocimientos previos que no pueden calcularse sino que deben ser obtenidos mediante la experimentación complementando ésta con un análisis estadístico de los resultados. En la primear etapa de pruebas a realizar, uno de los objetivos principales es el de determinar la calidad de las soluciones y la ocurrencia de eventos de estancamiento de los tres modelos empleados. Durante esta etapa se tomará como condición única de parada una cantidad de iteraciones por prueba de 1000. Esto nos permitiría elegir la mejor combinación de parámetros y nos daría una primera aproximación a lo que podemos esperar en cuanto a calidad de solución y estancamiento, pudiendo más adelante refinar estas conclusiones mediante estudios acotados (con menos combinaciones de parámetros) pero más largos (más iteraciones o más número de pruebas).

5.2.2 Métodos optimizados

Tomando como base el AS han surgido métodos mejorados que permiten una mayor consistencia en la obtención soluciones de buena calidad. Estudios comparativos de distintos métodos han arrojado resultados más pobres en el AS, especialmente en escenarios en los cuales se llega o supera a los 100 nodos, por este motivo se estableció en los objetivos del trabajo la implementación de un grafo con una cantidad de 100

ciudades. La comparación que se planea llevar a cabo es contra el modelo MMAS, uno de los métodos optimizados que se describirá en detalle más adelante. A continuación se dará una breve descripción, a modo de referencia, de los métodos optimizados más conocidos y utilizados:

- Sistema de Hormigas Elitistas (*AS_e*). La idea innovadora básica de este método es el de proveer un refuerzo adicional de feromonas a los arcos comprendidos dentro de la mejor solución encontrada, desde el comienzo de la búsqueda (mejor solución global).

$$\tau_{rs}(t) = \rho.\tau_{rs}(t-1) + \sum_{k=1}^{m} \Delta\tau_{rs}^{k} + e.\Delta\tau_{rs}^{mejor_{global}}$$

$$\text{donde } \Delta\tau_{rs}^{mejor\_globalk} = \begin{cases} {}^{1}/_{C^{mejor\_global}} & \text{si arco}(i,j) \in L^{mejor\_global} \\ 0 & \text{de otra forma} \end{cases}$$

  Este sistema agrega un parámetro *e*, que sería el número de hormigas elitistas, que debe ser ajustado convenientemente para obtener mejores resultados en el menor tiempo.
- Sistema de hormigas basado en rankings (*AS_{rank}*), propuesto por Bullhemiern en 1999. En este sistema se introduce la idea de que cada hormiga tiene un ranking y que deposita una cantidad de feromona de acuerdo con el mismo. Además, de manera similar a *AS_e*, el mejor recorrido recibe un refuerzo adicional. Las hormigas se clasifican en orden de acuerdo a la calidad de su solución. En un recorrido dado solo las *w*-1 hormigas mejor clasificadas y la que produjo el mejor camino pueden depositar feromona. La cantidad que deposita cada una también se ve afectada por el ranking multiplicándose esta cantidad por un peso (*w*) al que se le restará el valor *r* correspondiente al ranking de la hormiga. De esta forma, la hormiga mejor posicionada obtiene el peso máximo *w*.

$$\tau_{rs}(t) = \tau_{rs}(t-1) + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{rs}^{r} + w.\Delta\tau_{rs}^{mejor\_global}$$

$$\text{donde } \Delta\tau_{rs}^{r} = {}^{1}/_{Cr} \text{ y } \Delta\tau_{rs}^{mejor\_global} = {}^{1}/_{C^{mejor\_global}}$$

AS_{rank} mejora ligeramente a AS_e y sustancialmente a AS.

- Sistema de Colonia de Hormigas (ACS). Extiende al AS en tres aspectos: 1) explota más la experiencia acumulada por las hormigas. 2) para la actualización de feromona al final del recorrido se utiliza solo la hormiga que produjo la mejor solución global hasta el momento. 3) Se añade una nueva actualización local de feromona, lo que significa que cada hormiga modifica automáticamente el nivel de cada arco que visita, lo cual permite diversificar la búsqueda y disminuir las posibilidades de estancamiento. La regla de transición del ACS es:

$$S = \begin{cases} \arg max_{u \in J_k(r)} \{[\tau_{ru}]^{\alpha} [\eta_{ru}]^{\beta}\}, & \text{si } q \leq q_0 \\ S, & \text{en otro caso} \end{cases}$$

donde s es la siguiente ciudad a la que se desplaza la hormiga *k*, *q* es un valor aleatorio obtenido en una distribución uniforme [0,1], $q_0 \in [0,1]$ es el parámetro que determina la probabilidad con la que se elige el arco más prometedor, con el que se puede controlar la posibilidad de concentrar la búsqueda alrededor de la mejor solución encontrada hasta el momento o derivar la misma hacia otros caminos no explorados aún y *S* es la ciudad seleccionada con la regla de transición del AS con un valor de α = 1. La actualización de feromona se realiza mediante la operación:

$$\tau_{rs}(t) = \rho . \tau_{rs}(t-1) + (1-\rho) \frac{1}{C(S_{mejor\_global})}$$

que sólo se aplica sobre los arcos $\tau_{rs} \in S_{mejor\_global}$

Al aplicar feromona sobre los arcos de un solo recorrido la complejidad de los cómputos se reduce de $O(n^2)$ a $O(n)$.

- El sistema Mejor-Peor (BWAS). Fue desarrollado por Cordón et al. en el año 2.000 e incluye componentes de Computación Evolutiva para mejorar el equilibrio intensificación-diversificación. Mantienen la regla de transición del AS y cambia: 1) el mecanismo de actualización de feromona es más explorativo al evaporar todos los rastros, reforzar positivamente solo los de la solución global y negativamente los de la peor solución actual; 2) aplica una mutación de los rastros de feromona para diversificar la búsqueda de soluciones; 3) reinicializa la búsqueda cuando se produce estancamiento. La actualización de feromona en este modelo tiene dos etapas:

1. Se evaporan todos los rastros de feromona y se aporta en los de la mejor solución global:

   $$\tau_{rs}(t) = \rho.\tau_{rs}(t-1) + \Delta\tau_{rs}^{mejor\_global}$$

2. Se realiza una evaporación adicional de los rastros de feromona de la peor solución de la iteración actual que no estén contenidos en la mejor solución global:

   $$\tau_{rs}(t) \leftarrow \rho.\tau_{rs}(t), \forall a_{rs} \in S_{peor\_actual}\ y\ a_{rs} \notin S_{mejor\_global}$$

El BWAS considera búsqueda estancada si durante un número consecutivo de iteraciones (un porcentaje del total) no se consigue mejorar la solución global obtenida. En este caso se vuelve a inicializar la feromona volviendo todos los niveles a τ_0. Para conseguir la diversidad en el proceso de búsqueda se mutan los valores de los rastros de feromona, añadiendo una cierta cantidad mayor a los rastros de soluciones mejores.

5.2.3 El método Min-Max AntSystem (MMAS)

Entre los métodos mejorados hemos elegido el MMAS porque reúne la mayoría de las mejoras incorporadas en otros sistemas y, según experiencias comparativas previas (Stützle & Hoos, 2000 – Stützle & Dorigo, 2001) MMAS, junto al EAS son los sistemas que producen resultados de mejor calidad, más allá de esto, existen mayor cantidad de estudios comparativos disponibles entre MMAS y AS que cualquier otra combinación. En todos estos estudios se pone de manifiesto la superioridad de MMAS. En la tabla siguiente[10] se

[10] Min-Max AntSystem, Thomas Stützle y Holger H. Hoos, 2001,
Université Libre de Bruxelles, Bélgica, University of British Columbia, Canada.

muestran resultados comparativos de problemas TSP standard, de los cuáles se conoce su solución óptima y en los que se observa el desempeño superior de MMAS.

Problema	opt	MMAS+pts	MMAS	ACS	Asr	ASr+pts	ASe	ASe+pts	AS
eil51	426	427,1	427,6	428,1	434,5	428,8	428,3	427,4	437,3
kroA100	21282	21291,6	21320,3	21420	21746	21394,9	21522,8	21431,9	22471,4
d198	15780	15956,8	15972,5	16054	16199,1	16025,2	16205	16140,8	16702,1
ry48p	14422	14523,4	14553,2	14565,4	14511,4	14644,6	14685,2	14657,9	15296,4
ft70	38673	38922,7	39040,2	39099	39410,1	39199,2	39261,8	39161	39596,3
kro124p	36230	36573,6	36773,5	36857	36973,5	37218	37510,2	37417,7	38733,1
ftv170	2755	2817,7	2828,8	2826,5	2854,2	2915,6	2952,4	2908,1	3154,5

Tabla 6 – Estudios comparativos de ACOs (Stüzle y Holger 2001)

El algoritmo MMAS que analizamos en el presente documento explota muy bien el historial de búsqueda al permitir que sólo las mejores soluciones puedan agregar feromona durante la etapa de actualización. Además, utiliza un mecanismo bastante simple para limitar la influencia de la feromona que evita efectivamente la convergencia prematura de la búsqueda. Además, MMAS puede ser fácilmente extendido añadiendo algoritmos de búsqueda local.

El MMAS difiere del AS en tres aspectos clave:

(i) Para aprovechar las mejores soluciones encontradas durante una iteración o durante la ejecución del algoritmo, después de cada Iteración solo una sola hormiga añade feromona. Esta hormiga puede ser la que encontró la mejor solución en la iteración actual (mejor hormiga de la iteración) o la que encontró la mejor solución desde el comienzo del ensayo (mejor hormiga global).

(ii) Para evitar el estancamiento de la búsqueda, el rango de los posibles rastros de feromona en cada componente de la solución se limita a un intervalo $[\tau_{min}, \tau_{max}]$.

(iii) Además, deliberadamente inicializamos los rastros de feromona a τ_{max}, logrando de esta manera una mayor exploración de soluciones al comienzo del algoritmo.

En MMAS la regla modificada de actualizaciones de rutas de feromonas viene dada por

$\tau_{ij}(t+1) = \rho.\tau_{ij}(t) + \Delta\tau_{ij}^{mejor}$ donde $\Delta\tau_{ij}^{mejor} = {}^{1}/_{f(S^{mejor})}$ denota el costo de la mejor solución de la iteración (S^{ib}) o bien de la mejor solución global (S^{gb}).

El uso de una sola hormiga para la actualización de la trayectoria de feromonas también se propuso en ACS. Mientras que en ACS típicamente sólo se utiliza S^{gb} (aunque algunos experimentos limitados también se han realizado usando S^{ib}), MMAS se centra en el uso de las mejores soluciones de la iteración. El uso de una sola solución, ya sea S^{ib} o S^{gb}, para la actualización de feromonas es la herramienta más importante de explotación de búsqueda en MMAS. Debido a esto, los elementos de la solución que son utilizados con mayor frecuencia son fuertemente reforzados. Sin embargo, una elección cuidadosa entre la mejor hormiga de la iteración y la mejor hormiga global para la actualización de los senderos de feromonas controla la forma en que se explota la historia de la búsqueda. Cuando se utiliza sólo S^{gb}, la búsqueda puede concentrarse demasiado rápido alrededor de ésta solución y la exploración de posibles mejoras está limitada, con el consiguiente peligro de quedarnos atrapados en soluciones de mala calidad. Este peligro se reduce cuando se elige S^{ib} para la actualización de la trayectoria de feromonas ya que las mejores soluciones locales pueden diferir considerablemente de iteración en iteración y un mayor número de componentes de solución estarán en grado de recibir refuerzos ocasionales. Por supuesto, también se pueden utilizar estrategias combinadas con S^{ib} como valor predeterminado para actualizar las feromonas intercalando con S^{gb} sólo cada un número fijo de iteraciones que es lo que hemos implementado aquí.

Independiente de la elección entre la S^{ib} y la S^{gb} para la actualización de los rastros de feromonas, el estancamiento de la búsqueda puede ocurrir de todos modos. Esto puede suceder si en un punto de elección dado, el rastro de feromonas es significativamente mayor para una opción que para todas las demás. En el TSP, esto significa que para cada ciudad, uno de los arcos de salida tiene un nivel mucho más alto de feromonas que los otros. En esta situación, debido a la elección probabilística, una hormiga preferirá este componente

de la solución sobre todas las alternativas y le dará un refuerzo adicional a éste. En tal situación, las hormigas construirán la misma solución una y otra vez y la exploración del espacio de búsqueda se detiene. Obviamente, esta situación de estancamiento debe ser evitada. Una forma de lograr esto es influir en las probabilidades de elección del siguiente componente de solución, que dependen directamente de los rastros de feromonas y la información heurística, al limitar la influencia de los niveles de feromona. Para lograr este objetivo, MMAS impone límites explícitos τ_{min} y τ_{max} a los niveles mínimos y máximos de feromonas alcanzables para cada rastro de feromona $\tau_{ij}(t), \tau_{min} \leq \tau_{ij}(t) \leq \tau_{max}$. Después de cada iteración hay que asegurarse que el rastro de feromonas respete estos límites. Si tenemos $\tau_{ij}(t) > \tau_{max}$, se establece $\tau_{ij}(t) = \tau_{max}$Análogamente $\tau_{ij}(t) < \tau_{min}$, se establece $\tau_{ij}(t) = \tau_{min}$. También tengamos en cuenta que aplicando $\tau_{min} > 0$ y $\eta_{ij} < \infty$ para toda componente de la solución, la probabilidad de elegir un componente dado nunca es 0.

Ahora bien, para esto debemos elegir valores apropiados para estos límites. La forma que hemos elegido para esto se basa en la noción de convergencia para MMAS que expondremos a continuación. Decimos que MMAS ha convergido si para cada punto de elección dado, uno de los componentes de la solución tiene τ_{max} como rastro de feromonas asociado, mientras que en todas las alternativas los componentes de la solución tienen un rastro de feromonas τ_{min}. Si MMAS ha convergido, la solución construida al haber estado eligiendo siempre el componente de solución con el máximo rastro de feromonas corresponderá típicamente a la mejor solución que es capaz de encontrar el algoritmo. El concepto de convergencia de MMAS tiene una pequeña pero importante diferencia con el concepto de estancamiento. Mientras que el estancamiento describe la situación en la que todas las hormigas siguen el mismo camino, en situaciones de convergencia MMAS no es el caso debido al uso de límites en los niveles de feromona.

Ahora refinaremos nuestro concepto de convergencia mostrando que el máximo umbral posible de feromonas es un límite asintótico dado por:

$$\lim_{t \to 0} \tau_{ij}(t) = \tau_{ij} \leq \frac{1}{1-\rho} \frac{1}{f(S^{opt})}$$

En MMAS, se establece el rastro máximo de feromonas τ_{max} como una estimación de este valor máximo asintótico. Esto se consigue utilizando $f(S^{gb})$/ en lugar de $f(S^{opt})$t / en la ecuación anterior; cada vez que una nueva mejor solución es encontrada, τ_{max} se actualiza, lo que nos lleva a un valor de τ_{max} que cambia dinámicamente.

Para determinar valores razonables para τ_{min}, nos valdremos de los siguientes supuestos (el primero se basa en observaciones empíricas realizadas en algunos experimentos iniciales para el TSP):

(i) Las mejores soluciones se encuentran poco antes de que la búsqueda se estanque. En tal situación, la probabilidad de reconstruir la mejor solución global en una iteración del algoritmo es significativamente más alta que cero. Se pueden encontrar mejores soluciones cerca de la mejor solución encontrada.

(ii) La influencia principal en la construcción de la solución está determinada por la diferencia relativa entre el nivel superior e inferior de la feromona, más que por las diferencias relativas de la información heurística.

Dadas estas suposiciones, buenos valores de τ_{min} pueden encontrarse relacionando la convergencia del algoritmo al límite mínimo de los rastros. Cuando MMAS ha convergido, la mejor solución encontrada se construye con una probabilidad P_{best} que es significativamente mayor que 0. En esta situación, una hormiga construye la mejor solución si hace en cada punto de elección la decisión "correcta" y elige un componente de solución con el máximo rastro de feromonas τ_{max}. De hecho, la probabilidad P_{dec} de elegir el componente de solución correspondiente en un determinado punto de elección depende directamente de τ_{max}y τ_{min}. En aras de la simplicidad, supongamos que P_{dec} es constante en todos los puntos de decisión. Entonces, una hormiga tiene que hacer n veces la decisión "correcta", y por lo tanto, construirá la mejor solución con una probabilidad de P_{dec}^{n}. Haciendo

$P_{dec}^{n} = P_{best}$

Podemos determinar

$P_{dec} = \sqrt[n]{P_{best}}$

Por lo tanto, dado un valor para P_{best}, podemos determinar ahora los valores para τ_{min}. En promedio, en cada punto de elección una hormiga tiene que elegir entre *avg=n/2* componentes de solución. Entonces, la probabilidad P_{dec} de tomar la decisión correcta se puede calcular como:

$$P_{dec} = \frac{\tau_{max}}{\tau_{max} + (avg - 1)\tau_{min}}$$

Despejando τ_{min}:

$$\tau_{min} = \frac{\tau_{max}(1 - P_{dec})}{(avg - 1)P_{dec}} = \frac{\tau_{max}\left(1 - \sqrt[n]{P_{best}}\right)}{(avg - 1)\sqrt[n]{P_{best}}}$$

Nótese que, si $P_{best} = 1$ entonces $\tau_{min} = 0$. Si hacemos P_{best} demasiado pequeño, puede pasar que $\tau_{min} > \tau_{max}$, lo que nos lleva a que el algoritmo trabajará como si $\tau_{min} = \tau_{max}$, es decir que solo se usará la información heurística para todas las decisiones. En este trabajo se verificaran experimentalmente distintos valores de P_{best} con el objetivo de determinar el mejor para los escenarios propuestos.

Para la inicialización de feromona, debemos establecer el valor de todos los arcos al final de la primera iteración en τ_{max}. Para esto, lo que se hizo fue establecer τ_0 a un valor arbitrariamente grande que nos asegure que luego de la primera actualización todos los valores sean reducidos a τ_{max}. Esta inicialización se hace de esta forma para incentivar una mayor exploración durante las primeras iteraciones, lo cual se potencia si se incrementa ρ a un valor cercano a 1, con lo que la evaporación de feromona será lenta. Si por el

contrario se estableciera el valor inicial de feromona en τ_{min} las diferencias relativas de los valores de feromona en los arcos se haría mayores más rápidamente.

Un mecanismo adicional, llamado rastro de feromonas suavizado (PTS, por sus siglas en inglés), puede ser útil para aumentar el rendimiento del MMAS y por lo tanto lo hemos incorporado en este experimento con el propósito de evaluar su conveniencia. Cuando el MMAS ha convergido o está muy cerca de la convergencia (según lo indica en el factor de ramificación promedio[11]), este mecanismo aumenta los rastros de feromona proporcionalmente a su diferencia al límite máximo del rastro de feromonas:

$$\tau_{ij}^{*}(t) = \tau_{ij}(t) + \delta\left(\tau_{max}(t) - \tau_{ij}(t)\right)$$

Con $0 < \delta < 1$

Donde $\tau_{ij}(t)\ y\ \tau_{ij}^{*}(t)$ corresponden a los rastros de feromonas antes y después del suavizado. La idea básica de PTS es facilitar la exploración aumentando la probabilidad de seleccionar los componentes de la solución con bajo nivel de feromona. El mecanismo propuesto tiene la ventaja de que para $\delta < 1$, la información recopilada durante la ejecución del algoritmo (que se refleja en los rastros de feromona), no está completamente perdida, sino simplemente debilitada. En el caso de $\delta = 1$, este mecanismo corresponde a una reinicialización total de las marcas de feromonas, mientras que para $\delta = 0$ el PTS está completamente desconectado.

11

El factor de ramificación es el número de hijos en cada nodo. Si este valor no es uniforme, como en nuestro caso, se puede calcular un factor de ramificación promedio. En nuestra aplicación, este cálculo se hace cada 100 iteraciones.

El PTS es especialmente interesante si se permiten corridas largas, porque ayuda a lograr una exploración más eficiente del espacio de búsqueda. Al mismo tiempo, el PTS hace al MMAS menos sensible a la elección en particular del límite inferior del rastro de feromonas.

6 Implementación de la aplicación

6.1 Descripción del sistema

El sistema desarrollado consta de una aplicación Java que utiliza una base de datos DB2 conectada con JDBC, en la que se cargan y se consultan los datos del grafo y hojas de cálculo, formato .xls, en en las que se cargan los reportes y una planilla en particular como ayuda a la carga de variaciones de parámetros de entrada. Para trabajar con estas planillas se utilizó la API JXL, que permitió su uso sin modificaciones tanto al trabajar con MS Excel desde una plataforma Windows como al usar LibreOffice Calc desde una máquina con Ubuntu. La base de datos es única y se instaló en la máquina con Windows a la que se accedía remotamente cuando se usaba la estación Linux.

6.1.1 Estructura de la aplicación

La fig. 4 muestra un esquema de la estructura de clases de la aplicación viajante. En el paquete **viajante** se encuentra la operatoria básica de la aplicación. Se encuentra la interfaz principal de usuario y se inician todas las corridas (la resolución por heurística voraz se lleva a cabo en este paquete sin interactuar con el paquete hormigas). El paquete **BD** agrupa las clases que gestionan las conexiones a base de datos y genera la lista inicial de ciudades (solo opera al comienzo del programa si se va a realizar una búsqueda, o bien opera íntegramente si se usa la opción de manejo de base de datos). En el paquete de **hormigas** está toda la funcionalidad de las metaheurísticas empleadas. El paquete **graficos** contiene las funcionalidades para armar un grafo del recorrido a partir de una solución dada, y contiene el contorno del mapa de Argentina que se usa para circunscribir al grafo.

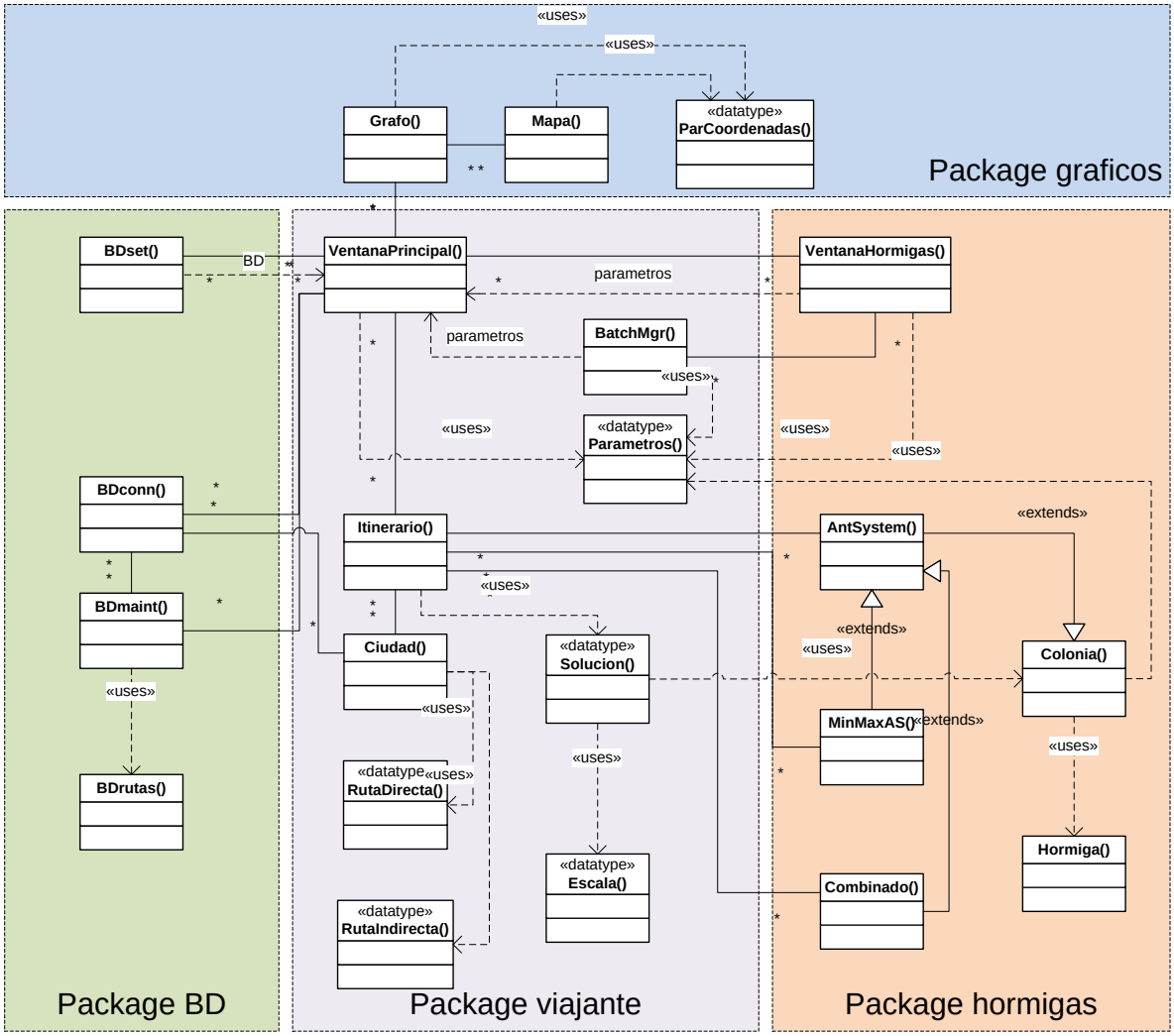

Fig. 4 – Estructura de la aplicación

Se describirá a continuación resumidamente la función y modo de operación de cada una de las clases principales.

- La clase **VentanaPrincipal**: Es la clase principal de la aplicación. Abre la ventana principal una vez que se selecciona la base de datos a conectar. Se organiza desde la misma toda la operatoria de las búsquedas y su correspondiente reporte. Se llega desde aquí al mantenimiento de base de datos y se puede acceder al grafico de una solución si se desea. Contiene a su vez una clase **AntSystemTask** que extiende **SwingWorker** para el manejo en un *thread* en *background* de las búsquedas mediante metaheurísticas.
- La clase **Itinerario**: Gestiona las búsquedas. Lleva a cabo la búsqueda por heurística voraz autónomamente y se comunica con las clases del paquete hormigas para pedir búsquedas por metaheurísticas. Devuelve una solución (tipo de datos **solucion**) que se verá parte en pantalla y mayormente servirá a cargar una planilla de datos.
- La clase **BatchMgr**: a través de esta se organizan las corridas de series de pruebas con distintas combinaciones de parámetros. Se encarga de separar los distintos conjuntos de parámetros y tamaño y cantidad de pruebas por cada uno, y lanza secuencialmente las mismas. Administra las soluciones obtenidas creando reportes en Excel. Si una corrida se interrumpe puede retomarse a partir del último resultado obtenido.
- La clase **Ciudad**: Contiene los datos propios de cada ciudad utilizados tanto por la aplicación para determinar las soluciones del problema, como datos de identificación y de coordenadas geográficas que son usadas por la representación gráfica.
- La clase **BDconn**: gestiona la conexión a base de datos. Extrae todos los datos y genera una array de ciudades que será usada para generar búsquedas y también es usado por las clases de gestión de la base de datos.
- La clase **BDmaint**: provee una herramienta de gestión de la BD que permite agregar o quitar tanto ciudades como rutas directas entre ciudades (para esto último se vale de la clase BDrutas(), que proporciona una ventana de dialogo adicional y funcionalidades relativas al menajo de las rutas.
- La clase **Colonia**: Es el fundamento de las metaheurísticas. Contiene las características comunes a cada una de ellas, en lo que se refiere a sus propiedades (parámetros), si aportar la lógica propia de cada metaheurística, la cual reside en cada una de las clases restantes del paquete hormigas.
- La clase **AntSystem**: contiene la base de la lógica de las metaheurísticas, especialmente la forma en que se elige un camino a partir de las probabilidades obtenidas de los niveles de feromona y la información heurística. Lo que cambia en los otros sistemas es la forma en que se cran y actualizan los niveles de feromonas.

- La clase **MinMaxAs**: Establece límites a los niveles de feromona y mecanismos para reinicializar total o parcialmente los mismos ante situaciones de convergencia. Toma de AntSystem la lógica por la cual las hormigas eligen los arcos de su recorrido.
- La clase **Combinado**: Extiende AntSystem agregando solo una propiedad y un método por el cual inicializa los niveles de feromona reforzando los arcos de una solución obtenida con una heurística voraz.

6.1.2 Estructura de la base de datos

Para la carga de los datos de las ciudades y sus rutas se implementaron tres bases de datos en DB2, a las cuales el programa se conecta mediante JDBC. Las tres bases tienen la misma estructura, diferenciándose solo en la cantidad de nodos cargados en cada una. Hay una base de test de 4 nodos que se utilizó para pruebas de la lógica de la aplicación en sí. La información cargada corresponde al grafo de la fig. 1. La segunda base es la de capitales, que contiene las 23 capitales provinciales del país (representada por la fig. 2). Se utiliza para pruebas cortas, como referencia. Se verá que, con la metaheurística afinada correctamente (incluso con AS) se obtiene siempre, y rápidamente, una solución que entendemos óptima (si bien no se demostró formalmente, luego de cientos de pruebas se llega a la conclusión de que siempre se converge a un valor mínimo, idéntico en todos los casos). La base principal contiene información de 100 ciudades y es la utilizada en las pruebas más significativas. Se eligió este valor porque según experiencias previas publicadas se sabe que AS no tiene un buen desempeño, por lo que se comparará el mismo con MMAS. El grafo con la información cargada en la base de datos se representa en la fig. 5.

Además de cargar nombre de la ciudad y las distancias de cada arco se incluyó en la base, como ya se indicó, los datos de coordenadas geográficas de las ciudades que resultan útiles para dibujar el grafo del recorrido.

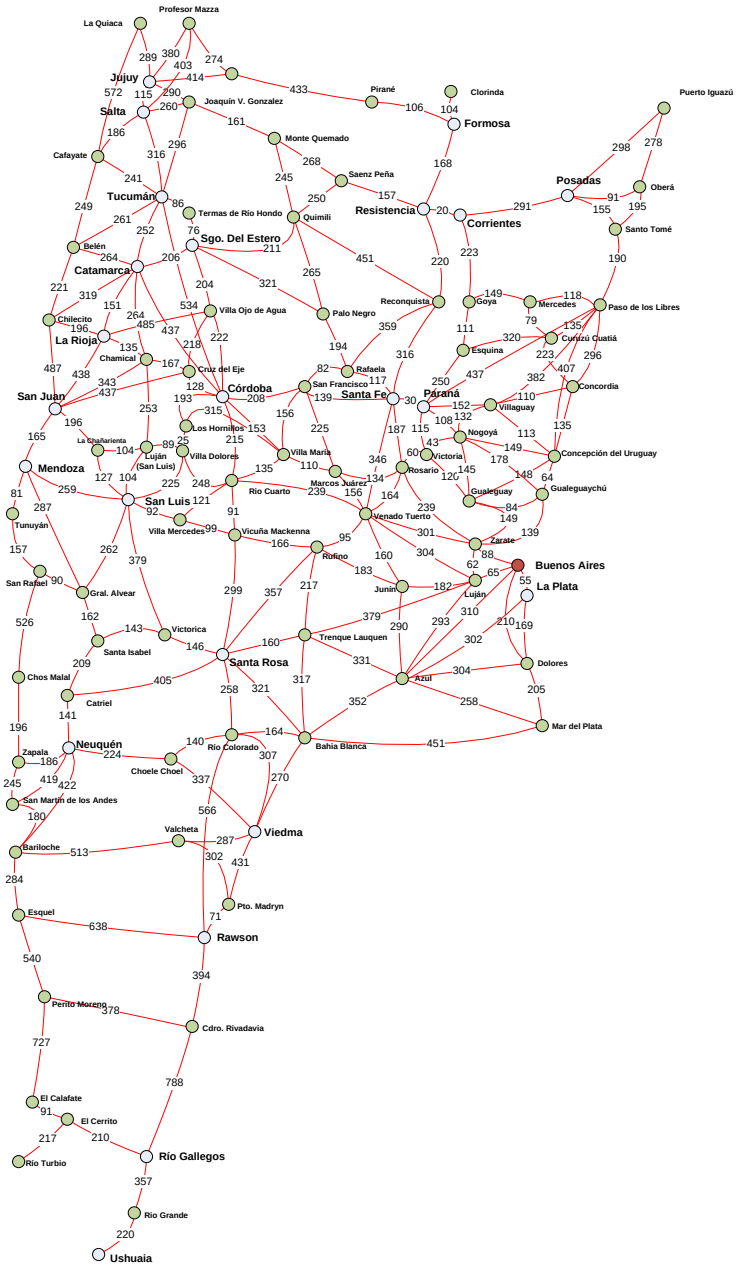

Fig. 5- Grafo 100 ciudades

La estructura de las bases es la siguiente:

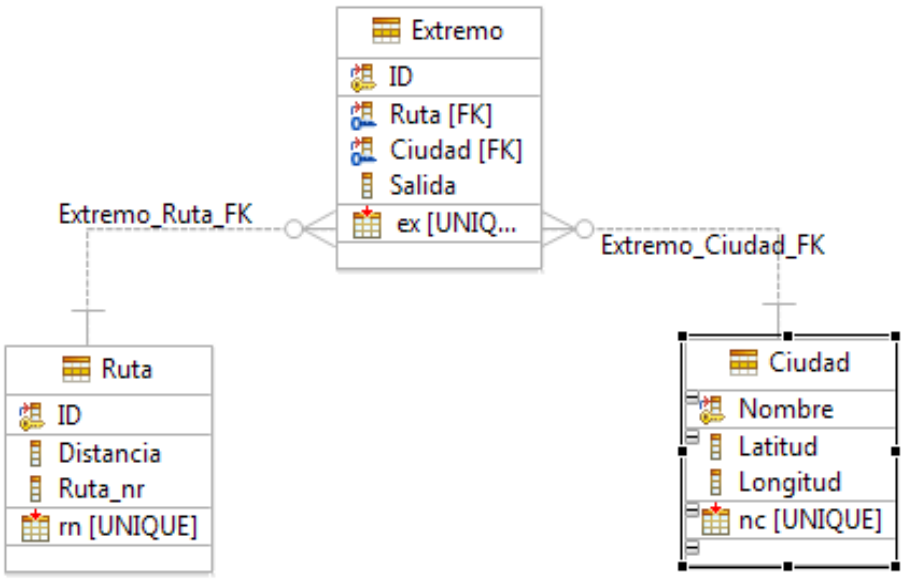

Fig. 6 – Estructura de las bases de datos

Las bases tienen una estructura sencilla de tres tablas. La tabla ciudad contiene el nombre y las coordenadas geográficas de cada nodo. Ruta tiene un **ID** numérico automático, la longitud de ese arco. Extremo relaciona a cada ciudad con los arcos que las unen a ciudades adyacentes (rutas directas). Tiene un campo **Salida** (CHAR de longitud uno con valores posibles “S” o “N”) que indica el sentido de circulación en ese arco y se usa para

esquemas de TSP asimétrico. El valor de **Salida** es S, si se permite que la hormiga salga de la ciudad por ese arco, y N si solo se permite que llegue pero que no salga.

6.1.3 Interfaz gráfica de usuario

La *fig 7.* Muestra una pequeña ventana de dialogo que aparece apenas se arranca la aplicación y sirve simplemente para la elección de la base de datos a la cual la aplicación se va a conectar. Las operaciones de base de datos, salvo que se escoja la opción de mantenimiento de la BD, se realizan por única vez al comienzo del proceso y luego esta conexión se cierra cuando comienza el momento de las pruebas propiamente dichas.

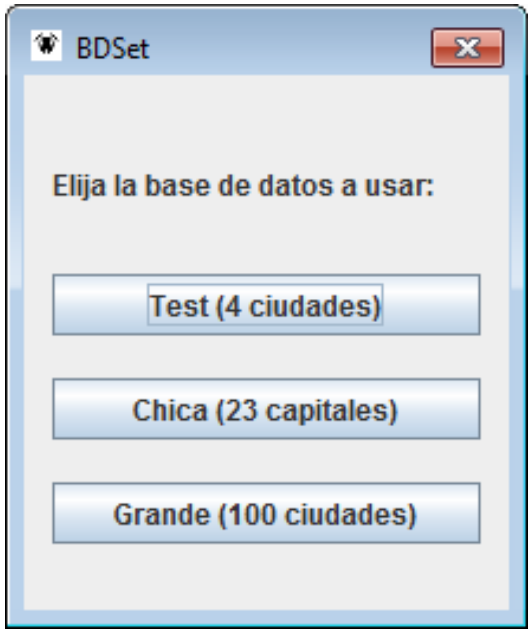

Fig. 7 – Elección de la BD

Una vez elegida una BD, se lleva a cabo la conexión y se pasa a la ventana principal (*fig.8*). Desde aquí se controla, mediante *radio buttons*, si se utilizará heurística Voraz o alguna metaheurística de hormigas. La heurística voraz incluye las opciones de búsqueda del mejor recorrido (alterna los puntos de inicio probando desde cada una de las ciudades) o la simple exploración desde un solo punto de inicio a elección. Si se usa ACO en cambio, se tiene solo la opción de la búsqueda mejor, y si vamos por esta, se abre la ventana de dialogo de elección de los parámetros y tipo de metaheurística a emplear.

El botón de mantenimiento de BD nos abre las ventanas correspondientes (se verán más adelante). La Ventana principal contiene un área de texto de múltiples usos. Durante las corridas largas, ya sea tipo *batch* o corridas simples de muchas iteraciones, se despliega durante la misma el tiempo transcurrido y el tiempo estimado restante, así como información adicional correspondiente a los parámetros empleados y el archivo de salida al que se están direccionando los datos a reportar.

Al finalizar se muestran las escalas del recorrido, su distancia entre las mismas, y la distancia total recorrida. La ventana cuanta también con tres barras de estado que van mostrando el porcentaje de iteraciones de un ensayo, el porcentaje de ensayos de una prueba dada y, si se hacen varias pruebas en una sala corrida, con distintas combinaciones de parámetros, se marca en la última barra el porcentaje de pruebas de la corrida.

El botón Mostrar Grafo nos permite acceder a un mapa con el grafo del recorrido enmarcado en un mapa del territorio argentino. Cada nodo está indicado con el mismo número de orden que le corresponde en la tabla del recorrido desplegada en al área de texto.

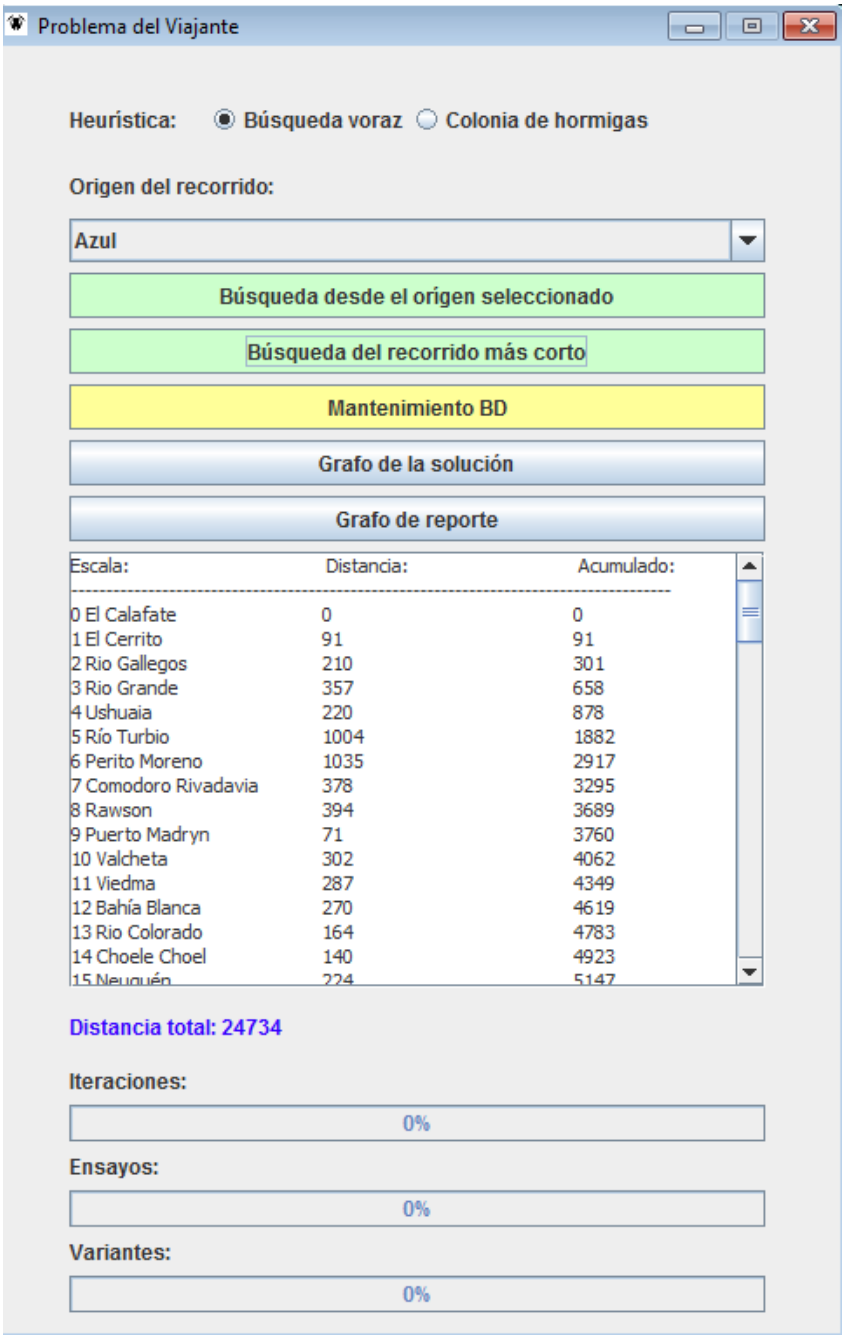

Fig. 8 – Ventana Principal

La fig.9 es la ventana que nos permite seleccionar el tipo de metaheurística y sus parámetros (o conjunto de combinaciones de los mismos). Los tres primeros botones son claramente el tipo de metaheurística a emplear, los dos segundos indican si vamos a indicar una combinación de parámetros para una única prueba o si se hará una prueba múltiple, en dónde se tomarán distintas combinaciones de un formulario de entrada (planilla de cálculo) que se verá más adelante en detalle. Los parámetros son los descriptos anteriormente. El archivo de reporte se guarda en una planilla por defecto o se puede modificar antes de comenzar el archivo de salida.

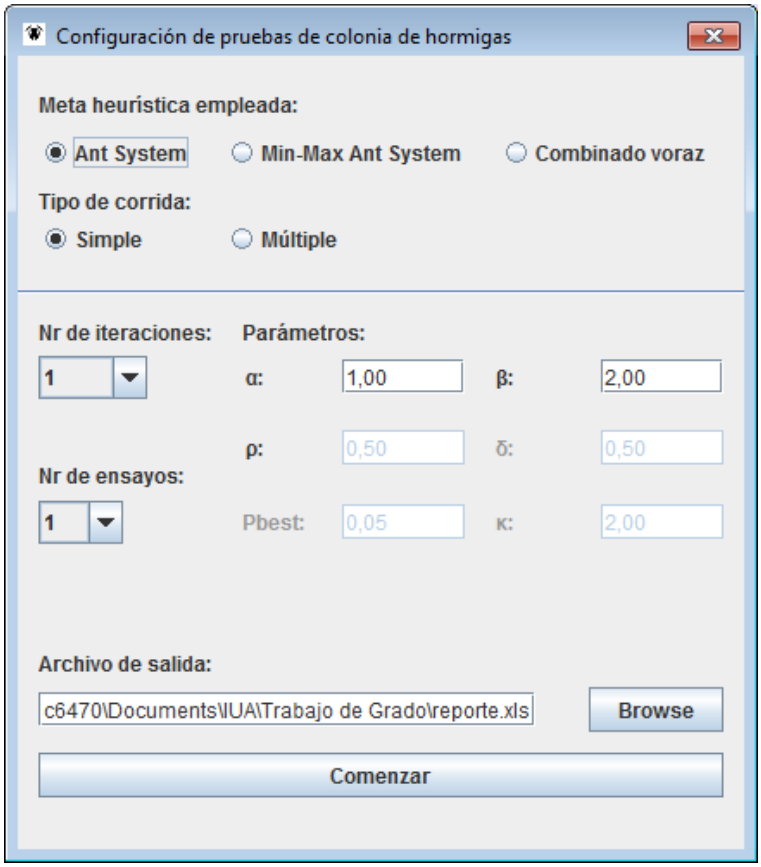

Fig. 9 - Ventana Hormigas

La fig. 10 muestra el grafo de un recorrido, que aparece al finalizar una búsqueda y al que se accede mediante el botón mostrar grafo de la ventana principal (grafo de la solución actual) el botón grafo de reporte, que abre una ventana que nos permite elegir un reporte y un itinerario en particular dentro del mismo.

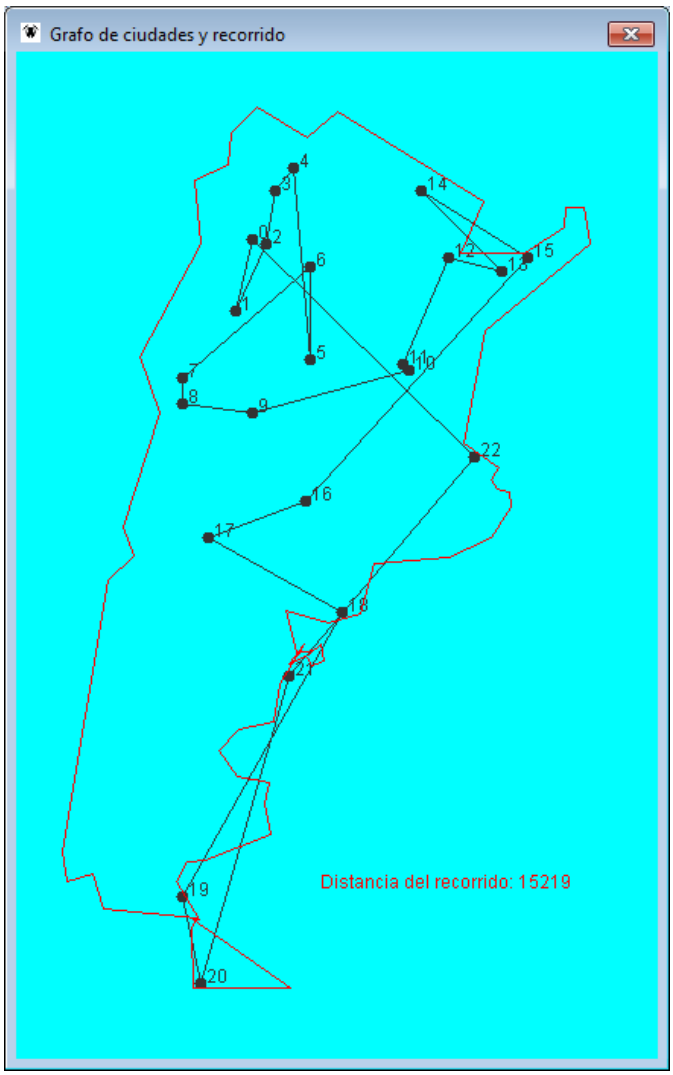

Fig. 10 – Grafo del recorrido calculado

La fig. 11 muestra la primera ventana de mantenimiento de base de datos, en la que se puede agregar o borrar una ciudad de nuestro grafo. En el caso de agregar se pide nombre y coordenadas geográficas. Para el caso de borrar se pide elegir una ciudad de un *combo box*.

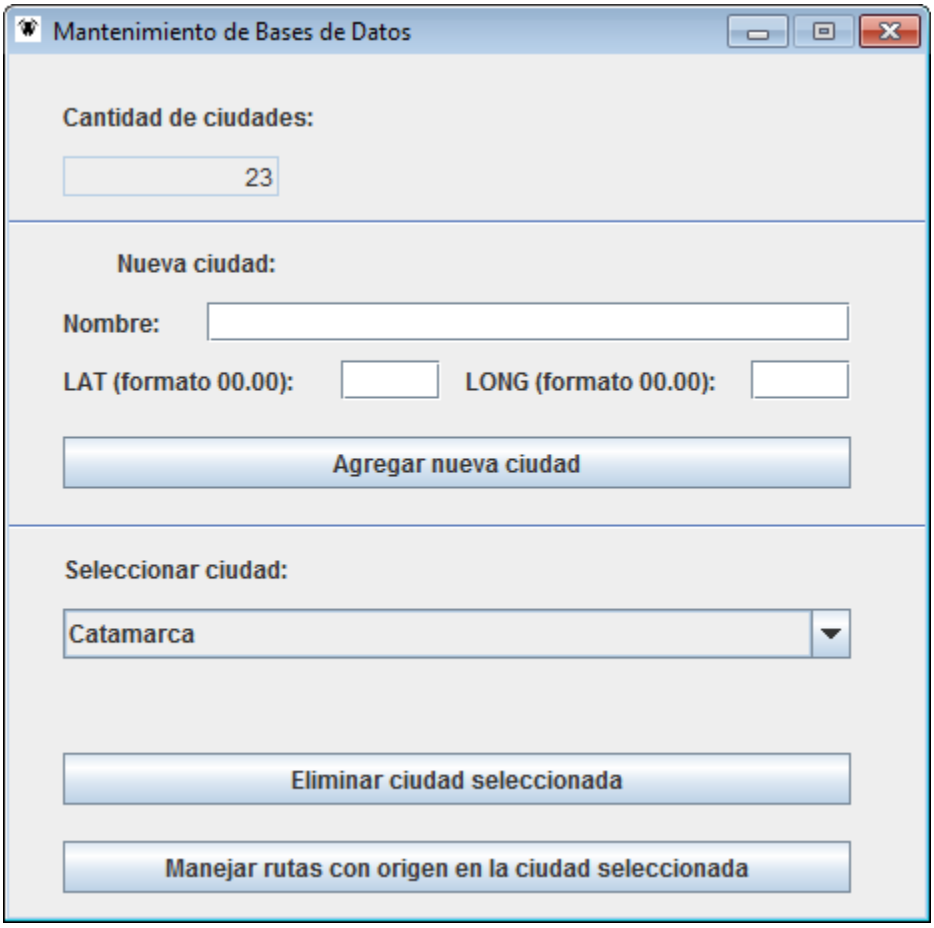

Fig. 11 – Ventana Mantenimiento BD

Si se acciona el botón **Manejar rutas con origen en la ciudad seleccionada,** se abre una ventana para este fin, la cual se muestra en la *fig. 12*. Para agregar seleccionamos un destino de un combo box que muestra todas las ciudades para las cuales aún no hay ruta directa conocida desde el origen seleccionado, al costado tenemos un campo de texto para agregar la distancia de esta ruta entre ciudades. Para el caso de uso de querer remover una ruta, se usa otro combo box que muestra las ciudades a las que sí se puede acceder en ese momento.

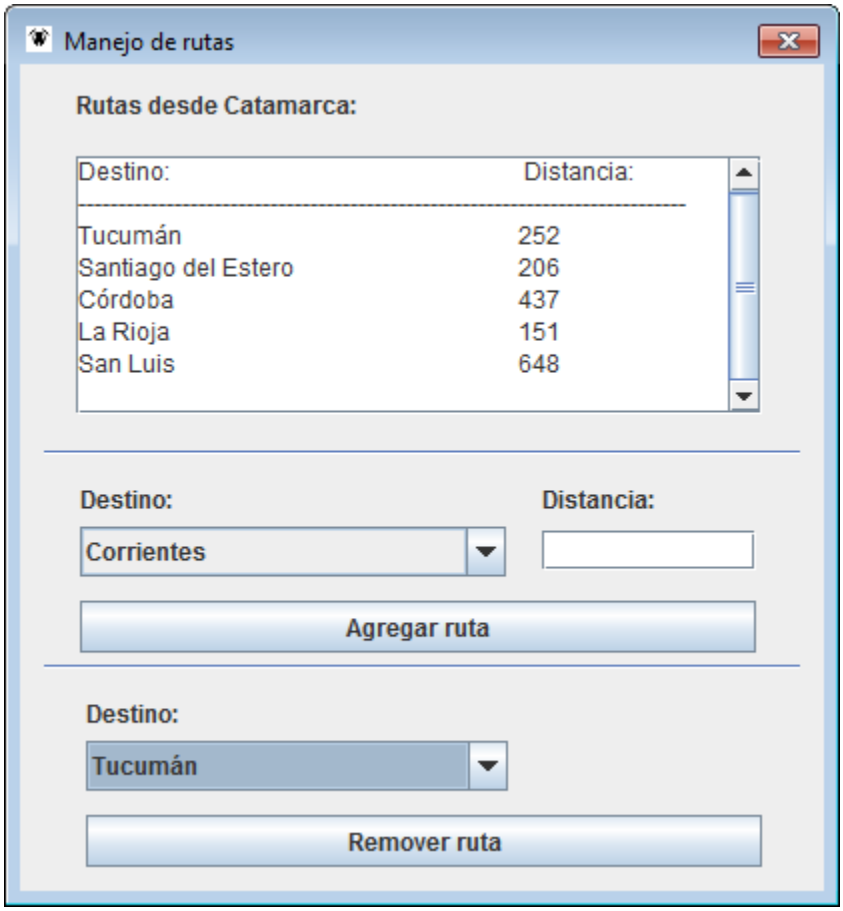

Fig. 12- Ventana gestión de Rutas en BD

6.1.4 Reportes

Más allá de presentar una solución en pantalla, en la ventana principal, y permitir además la visualización del grafo, las pruebas múltiples necesarias para la obtención de una cantidad de datos que pueda ser estadísticamente significativa requiere la creación de distintos reportes. Existen dos tipos de reporte que se descargaran desde la aplicación directamente a planillas Excel (formato .xls).

El primer tipo de reporte es usado para una corrida simple, es decir una corrida con una sola combinación de parámetros, aunque puede tener *n* intentos, cada uno con *m* iteraciones. En este sencillo caso, el reporte está compuesto de dos planillas: en la primera, etiquetada resultados, se muestran tanto los parámetros usados como los resultados finales de cada intento (distancia de la solución, iteración de la mejor solución, y tiempo total del ensayo y tiempo hasta encontrar la solución), en la segunda planilla, etiquetada progreso, se van mostrando soluciones locales a medida que se avanza con las iteraciones, información que luego se utiliza gráficamente para modelar el progreso de los cálculos, lo que permite evaluar los procesos de estancamiento, y las velocidades para alcanzar soluciones y sus calidades. Los datos se presentan con formato condicional de escala de colores, siendo los mejores resultados los del extremo verde y los peores los rojos.

Distancia	Conv.	T-conv	T-final		Menor	Tipo	Alfa	Beta	Rho	Pbest	Delta
20972	810	85,61	105,10		20891	MMAS	1.0	5.0	0.98	0.05	0.5
20932	683	71,33	104,60		Iteraciones						
20932	847	88,01	104,18		1000						
20891	993	105,43	106,16								
20932	696	74,69	112,35								
21024	751	108,08	149,24								
21005	965	147,65	152,16								
20932	614	73,72	115,16								
20932	841	89,13	105,94								
20932	972	117,10	121,08								

Fig. 13 – Reporte de corrida simple

El segundo tipo de reporte se toma cuando se elige la opción de una corrida múltiple, en la cual se especifica un rango para cada parámetro y la clase **BatchMgr** se encarga de programar corridas con cada una de estas combinaciones. Cada combinación puede ser probada una cantidad *e* de ensayos, pero el reporte mostrará para cada combinación solamente los datos correspondientes a la mejor solución. En cada caso mostrará el mejor recorrido de la combinación, el promedio de los recorridos de esa combinación, su desviación estándar y la longitud del peor recorrido. Se agrega la columna con números de referencia (primera columna), que servirán para identificar el recorrido correspondiente en la segunda hoja de la planilla y para restablecer el orden inicial si se pretende trabajar con ordenando los datos según el criterio que se considere más conveniente. Esta segunda hoja, etiquetada Itinerarios, muestra para cada mejor solución de cada combinación el recorrido compuesto por cada escala y la distancia entre las mismas. Con esta información se puede eventualmente reconstruir el grafo del recorrido.

#	Mejor	Promedio	Peor	SD	T.prom.final	T.prom.sol	alfa	beta	rho	delta	pbest	iteraciones	ensayos	T.final.total	T.sol.total	En iter.
1	23803	24381	24973	358,21	138,96	66,05	1	1	0,40	0	0	1000	10	1389,61	660,52	201
2	23622	24136	24575	276,93	117,20	60,42	1	1	0,50	0	0	1000	10	1171,98	604,23	509
3	23618	24125	24840	353,03	140,51	63,46	1	1	0,60	0	0	1000	10	1405,11	634,64	294
4	22324	22720	22994	222,74	90,09	41,46	1	2	0,40	0	0	1000	10	900,86	414,58	753
5	22291	22633	23117	242,37	81,15	47,31	1	2	0,50	0	0	1000	10	811,45	473,06	404
6	22384	22783	23423	311,74	72,11	40,50	1	2	0,60	0	0	1000	10	721,06	404,96	817
7	21897	22238	22621	231,41	166,78	65,92	1	3	0,40	0	0	1000	10	1667,75	659,22	442
8	22034	22351	22840	240,86	190,30	76,23	1	3	0,50	0	0	1000	10	1903,03	762,27	523
9	22096	22332	22520	135,64	123,86	81,72	1	3	0,60	0	0	1000	10	1238,65	817,24	988
10	21983	22262	22538	197,30	133,12	61,14	1	4	0,40	0	0	1000	10	1331,24	611,42	416
11	21939	22212	22568	218,13	115,36	46,72	1	4	0,50	0	0	1000	10	1153,56	467,24	420
12	22041	22276	22514	130,03	117,20	48,99	1	4	0,60	0	0	1000	10	1172,00	489,93	855
13	21995	22386	22699	170,02	117,86	82,32	1	5	0,40	0	0	1000	10	1178,58	823,19	962
14	21728	22285	22632	235,28	112,32	68,29	1	5	0,50	0	0	1000	10	1123,22	682,90	692
15	22082	22343	22550	158,84	119,37	72,63	1	5	0,60	0	0	1000	10	1193,70	726,28	930
mejor	**21728**	**22212**	**22514**	**130,03**												
peor	23803	24381	24973	358,21												
promedio	22389,13	22764,20	23160,27	232,17												
media	22082	22351	22699	231,41												

Fig. 14 – Reporte de corrida múltiple (primera planilla)

6.1.5 Planillas de parámetros de entrada

La carga de los rangos de parámetros a emplear en corridas múltiples se realiza mediante una planilla de cálculos (fig. 15) que la aplicación lee para crear una *array* de combinaciones que se usará para iniciar cada una de las corridas individuales. Se eligió esta alternativa por dos motivos: el primero es que resultaba engorroso el uso de formularios de entrada en la aplicación siendo que además, como segundo motivo, esta modalidad permite estimar, conociendo el tiempo promedio de una iteración del programa, cuánto demorará la corrida aproximadamente antes de cargarla en la aplicación, con lo cual es posible jugar con la cantidad de parámetros para ajustar los tiempos a valores razonables.

peso heurística			cant				Tiempo estimado total	
αMax	αMin	Δ-α		Combinaciones	iteraciones	tiempo it	horas	minutos
1	1	1	1	9	180000	0,13	6	15
peso feromona								
βMax	βMin	Δ-β						
5	5	1	1					
evaporación								
ρMax	ρMin	Δ-ρ						
0,98	0,98	0,1	1					
smoothing								
δMax	δMed	δMin						
1	0,5	0	3					
Min-Max								
P_{best}MAX	P_{best}MED	P_{best}MIN						
0,5	0,05	0,005	3					
Parms								
Iteraciones	Ensayos	Eurística						
20000	1	MMAS	20000					
Última combinación								
0								

Fig. 15- planilla de carga de valores paramétricos

En esta planilla, en los renglones 3, 6 y 9 se cargan los valores de α, β y ρ respectivamente. En la primera columna se cargan los valores máximos, en la segunda los mínimos y la tercera es el intervalo entre un valor y otro. En la cuarta columna se calculan las cantidades de combinaciones que surgen de acuerdo a los valores seleccionados. Los renglones 12 y 15 se usan para cagar los valores de δ y P_{best}, pero en estos casos no se indican intervalos sino tres valores prefijados. En el renglón 18 se cargan la cantidad de iteraciones, el número de ensayos y el tipo de metaheurística empleada. En el renglón 21 se carga un valor que especifica el último valor cargado de un reporte de una corrida interrumpida por cualquier motivo. En este caso el programa retoma desde donde se quedó y continúa cargando los datos en la misma planilla hasta completarla (o hasta que se interrumpa de nuevo).

6.2 Algoritmos empleados

En esta sección se hará una descripción de los algoritmos empleados desde el punto de vista de su implementación en esta aplicación en particular.

6.2.1 Emulación de ruteo dinámico

Como ya se adelantó, la construcción de la matriz de distancias de los arcos del grafo se realiza a partir de contar con la información mínima necesaria para su cálculo, lo cual reduce mucho la cantidad de datos a manejar, especialmente a medida que se agregan nodos al esquema.

Para implementar este mecanismo, la clase ciudad contiene una tabla (array) de rutas directas, es decir, arcos hacia otras ciudades que en realidad corresponden a rutas reales que unen dos ciudades del grafo **sin pasar por ninguna de las otras**, y que además corresponden a la **ruta más corta** entre las mismas. Esta tabla de ruteo se inicializa al

comienzo del programa, cuando se conecta a la base de datos para bajar toda la información inicial relativa al grafo. Una vez bajada toda esta información y cuando se crearon todas las instancias de ciudades con sus correspondientes tablas de rutas directas, se desconecta la BD y comienza el cálculo de las rutas indirectas, que dará lugar a otra tabla que incluye todos los arcos de grafo desde cada instancia, todo lo cual es llevado adelante por la clase **BDconn**. En principio se crea un *array* con todas las instancias ciudad, que en este punto contienen su nombre, posición y tabla de rutas directas. Luego se procede de la siguiente forma:

```
Ciudades: array de instancias ciudad
converge : boolean que indica si la red converge
C : número de ciudades,
R : número de rutas directas de la tabla,
rutasD: tabla de rutas directas,
rutasI: tabla de rutas indirectas,
destino: ciudad adyacente a una ciudad dada
/* Inicializacion de rutasI */
Para c = 1 hasta C hacer
      Para la ciudad c obtener rutasD
      Para r = 1 hasta R hacer
            Ingresar rutasD[1] en rutasI
/* Propagación de tablas */
Mientras no converge hacer
      Para c = 1 hasta C hacer
```

```
Para la ciudad c obtener rutasD
Para r = 1 hasta R hacer
      Para rutasD[r] obtener destino
      Enviar rutasI a destino
      Actualizar tablas en destino
```

La lógica anterior está implementada en el método `propagarRutas()` de la clase **DBconn**. Mientras que en cada ciudad, el método `recibeRutas()`

En cada ciudad, cuando se recibe una tabla de un vecino, se compara la información con la contenida en la tabla existente. Si la nueva información permite mejorar alguna ruta, se actualiza y se informa que se hizo esta modificación. Si al final de una rueda completa de propagación de tablas ningún nodo informa mejoras entonces se declara la convergencia del grafo y el proceso se detiene.

Para ejemplificar este proceso, tomemos un caso del escenario de capitales. Elijamos una ciudad sin demasiados vecinos para no complicar el ejemplo: Formosa.

La estructura de las tablas se basa en dos tipos de datos: Ruta_directa y Ruta _indirecta. La primera clase tiene los siguientes atributos:

```
private final String destino;
private final int distancia;
private Ruta_indirecta[] tabla_adyacente;
```

tabla_adyacente hace referencia a la tabla de rutas indirectas de cada nodo adyacente. La clase Ruta_indirecta, por su parte tiene estos atributos:

```
private final String destino;
```

```
private int distancia;

private String ruteador;

private double nivel_feromona;
```

En donde **ruteador** indica al nodo adyacente por el que pasamos para llegar al nodo indicado en destino, y será `local` si el destino mismo es un nodo adyacente, o `null` si es el mismo nodo de origen (las tablas de ruteo tienen todas la misma estructura, las mismas ciudades, por lo tanto la tabla de un nodo en particular tendrá como uno de sus componentes a sí mismo, el cual se identificará por ser el único con distancia 99999999 al final de la propagación de rutas, y como **ruteador** siempre tiene `null`). El nivel de feromona del arco se guarda en esta tabla, pero obviamente es irrelevante en esta parte del proceso, solo se usará para la búsqueda de recorridos por métodos meta heurísticos.

Antes de comenzar el proceso de propagación Formosa tiene su tabla de rutas directas, y a partir de esta inicializa la tabla de rutas indirectas.

Tabla de rutas directas de Formosa

destino	distancia	tabla_adyacente
Jujuy	953	null
Resistencia	168	null
Salta	976	null

Tabla de rutas indirectas de Formosa

destino	distancia	ruteador	feromona
Catamarca	99999999	null	0
Corrientes	99999999	null	0
Córdoba	99999999	null	0
Formosa	99999999	null	0
Jujuy	953	local	0
La Plata	99999999	null	0

La Rioja	99999999	null	0
Mendoza	99999999	null	0
Neuquén	99999999	null	0
Paraná	99999999	null	0
Posadas	99999999	null	0
Rawson	99999999	null	0
Resistencia	168	local	0
Rio Gallegos	99999999	null	0
Salta	976	local	0
San Juán	99999999	null	0
San Luis	99999999	null	0
Santa Fe	99999999	null	0
Santa Rosa	99999999	null	0
Santiago del Estero	99999999	null	0
Tucumán	99999999	null	0
Ushuaia	99999999	null	0
Viedma	99999999	null	0

En esta primera inicialización todos los ruteadores son `null` , menos los correspondientes a las rutas adyacentes, ya que o son desconocidos, o es la entrada que corresponde al mismo nodo. En cuanto a las distancias, solo tenemos la de los nodos adyacentes, las demás son todas 99999999.

En un paso siguiente del proceso, Formosa recibirá las tablas de rutas indirectas de sus vecinos, que han sido recientemente inicializadas y solo contienen información de sus propios adyacentes.

Tabla de rutas directas de Formosa

destino	distancia	tabla_adyacente
Jujuy	953	tabla de Jujuy
Resistencia	168	tabla de Resistencia
Salta	976	tabla de Salta

Las tablas resumidas (se ponen solo las distancias), de los nodos adyacentes son:

Tabla de rutas de Jujuy	
destino	distancia
Catamarca	99999999
Corrientes	99999999
Córdoba	99999999
Formosa	953
Jujuy	99999999
La Plata	99999999
La Rioja	99999999
Mendoza	99999999
Neuquén	99999999
Paraná	99999999
Posadas	99999999
Rawson	99999999
Resistencia	9999999
Rio Gallegos	99999999
Salta	115
San Juán	99999999
San Luis	99999999
Santa Fe	99999999
Santa Rosa	99999999
Sgo. Del Estero	99999999
Tucumán	99999999
Ushuaia	99999999
Viedma	99999999

Tabla de rutas de Resistencia	
destino	distancia
Catamarca	99999999
Corrientes	20
Córdoba	99999999
Formosa	168
Jujuy	99999999
La Plata	99999999
La Rioja	99999999
Mendoza	99999999
Neuquén	99999999
Paraná	99999999
Posadas	99999999
Rawson	99999999
Resistencia	99999999
Rio Gallegos	99999999
Salta	846
San Juán	99999999
San Luis	99999999
Santa Fe	563
Santa Rosa	99999999
Sgo. del Estero	699
Tucumán	99999999
Ushuaia	99999999
Viedma	99999999

Tabla de rutas de Salta	
destino	distancia
Catamarca	99999999
Corrientes	99999999
Córdoba	99999999
Formosa	976
Jujuy	115
La Plata	99999999
La Rioja	99999999
Mendoza	99999999
Neuquén	99999999
Paraná	99999999
Posadas	99999999
Rawson	99999999
Resistencia	846
Rio Gallegos	99999999
Salta	99999999
San Juán	99999999
San Luis	99999999
Santa Fe	99999999
Santa Rosa	99999999
Sgo. del Estero	99999999
Tucumán	316
Ushuaia	99999999
Viedma	99999999

Una vez intercambiadas las tablas cada nodo las procesa para actualizar la tabla propia. Para esto revisa la información nueva que no corresponda a nodos adyacentes. Ve la distancia que marca la tabla importada, le suma la distancia que hay hacia el router por el

cual se llega al destino estudiado y si es menor que la que figura en la tabla actual, actualiza la distancia y, si es necesario, el *ruteador*. Después de este primer intercambio la tabla de Formosa queda de la siguiente manera:

Tabla de rutas indirectas de Formosa

destino	distancia	ruteador	feromona
Catamarca	99999999	null	0
Corrientes	188	Resistencia	0
Córdoba	99999999	null	0
Formosa	99999999	null	0
Jujuy	953	null	0
La Plata	99999999	null	0
La Rioja	99999999	null	0
Mendoza	99999999	null	0
Neuquén	99999999	null	0
Paraná	99999999	null	0
Posadas	99999999	null	0
Rawson	99999999	null	0
Resistencia	168	null	0
Rio Gallegos	99999999	null	0
Salta	976	null	0
San Juán	99999999	null	0
San Luis	99999999	null	0
Santa Fe	705	Resistencia	0
Santa Rosa	99999999	null	0
Santiago del Estero	867	Resistencia	0
Tucumán	1292	Salta	0
Ushuaia	99999999	null	0
Viedma	99999999	null	0

Este proceso continúa iterativamente hasta que todas las tablas contienen la información de todos los arcos entre nodos, con la distancia mínima entre ellos.

6.2.2 Heurística Voraz

La implementación de las búsquedas por heurística voraz es bastante sencilla y se lleva a cabo en la clase Itinerario. En esta clase están los métodos que permiten realizar una búsqueda desde un origen determinado, y también buscar alternativamente desde todos los orígenes posibles para luego elegir la mejor opción. La heurística voraz empleada en el TSP, como se mencionó, es la siguiente: *desde una ciudad dada, ir a la siguiente más próxima no visitada*.

Esto se hace por medio del método `getNext()` de la clase itinerario.

Se incluye la heurística voraz en este trabajo por tres motivos:

1. El primero es que sirva de referencia para las soluciones halladas con metaheurística. Como no estamos empleando un escenario de TSP conocido, del cual se conoce su solución óptima, ni estamos en condiciones de calcularla, tomaremos como referencia de la calidad de una solución cuánto esta mejora la obtenida heurísticamente, en lugar de cuánto se acerca a la óptima.
2. El segundo es que se necesita un recorrido de referencia para la inicialización de feromonas.
3. Por último, vamos a investigar si se pueden obtener resultados distintos mediante un refuerzo inicial del recorrido voraz.

6.2.3 Metaheurísticas de Colonia de Hormigas

Para la implementación de las metaheurísticas se parte de una clase Colonia, que contiene las propiedades comunes de los sistemas empleados, a los que se accede por medio de *getters* protegidos, que solo accede las clases que la heredan.

La lógica de cada metaheurística se implementa en una clase aparte. Primero está la clase AS, que extiende a Colonia, y luego MMAS y Combinado, ambas extendiendo AS ya que esta tiene mucha lógica en común con las otras, por ser el algoritmo de ACO original. El algoritmo de AS ya se describió anteriormente, y se implementa en el método

`antSystem()` de la clase **AntSystem** (no confundir con el constructor de la clase, `AntSystem()`).

En esta clase, el nivel inicial de feromona en todos los arcos se fija en:

$\tau_{ij} = \tau_0 = {}^{m}/_{C^{nm}}$

En donde *m* es el número de hormigas y C^{nm} es la longitud del recorrido más corto obtenido con heurística voraz.

El número de hormigas en cada escenario se fija igual al número de ciudades, lo cual, por experiencias anteriores, se sabe que es un número adecuado, y por otro lado, con esta cantidad podemos hacer que cada hormiga comience su recorrido en una ciudad distinta, no dejando ciudades sin cubrir, por lo que no tiene sentido la aleatorización de los puntos de origen.

Para el MMAS se utilizará *smoothing*, que se disparará cuando el factor de ramificación promedio sea $FRP < 1{,}1 * \tau_{min}$ y hayan pasado 50 iteraciones sin modificaciones. Esto se implementa primero verificando el intervalo de iteraciones sin mejoras de manera de no calcular el *FRP* en cada iteración. Este intervalo se mide con un contador que se restablece luego de hallar una nueva solución global o cada vez que se calcula el FRP. Esto significa que, si pasaron 50 iteraciones sin mejoras se calculará el FRP, pero si este valor no dispara el PTS no volverá a calcularse en las próximas 50 iteraciones aunque el valor de la solución siga siendo el mismo.

Para el cálculo del *FRP* hemos establecido en la clase **MinMaxAs** una variable global tauSum, en donde los métodos que elevan los niveles de feromona del mejor camino guardan la sumatoria de los niveles de todos los arcos del mismo. Luego simplemente se procede, solo en las iteraciones que es requerido, a sumar los niveles de todos los arcos

del grafo, se les resta tauSum y se los divide por el número de arcos que no están en la solución global.

El número de arcos fuera del grafo de la mejor solución (o de cualquier solución en particular) se calcula de la manera siguiente. Cada ciudad tiene una tabla de rutas en la que hay una cantidad de entradas igual al número de ciudades, por lo que el número total de entradas para los grafos es igual a n^2, donde n es el número de ciudades[12].Como en cada tabla debe haber dos entradas que corresponden a arcos de un determinado recorrido, y una entrada que corresponde a la ciudad de origen (es decir que no indica ningún arco), el número buscado será:

```
totalArcos = n_ciudades*(n_ciudades-3)
```

Para probar el efecto del *smoothing* en la solución se llevarán adelante pruebas con tres valores de δ. Con un δ de 0 el *PTS* está inactivo, ya que el incremento de feromona de los arcos fuera del mejor recorrido es nulo. Con un δ de 0,5 el *PTS* incrementará el nivel de feromona de cada arco llevándolo a la mitad de la diferencia entre $\tau_{max}(t)\ y\ \tau_{ij}(t)$. Por último, con un δ de 1, los valores de feromona en los arcos se restablecen a τ_{max}.

[12] Nótese que cada ciudad tiene la información completa de todos los arcos que salen de ella, por lo que en realidad la información de cada arco aparece dos veces, una en cada uno de los extremos (ciudades) que la limitan. Entonces si bien el número de arcos del grafo es $n^2/2$, el número de entradas que nosotros consideramos es n^2. De la misma manera, el valor de tauSum es en realidad el doble de la cantidad de feromona del recorrido de la mejor solución. Haciendo esta salvedad, el número calculado del FRP representa el nivel promedio feromona de los arcos no incluidos en la solución.

Para el cálculo del umbral τ_{max} se parte $\tau_{max} = \frac{1}{(1-\rho)f(S^{gb})}$. En el inicio, tomamos S^{gb} de la solución obtenida por la heurística voraz.[13]

Para el cálculo de τ_{min} nos valemos del valor de P_{best} pasado como parámetro. En este trabajo experimentaremos con valores de P_{best} de 0,5, 0,05 y 0,005.

El cálculo inicial y el ajuste de estos valores se llevan a cabo con este método:

```
private void setTaus(int recorrido){
    tauMax=1/((1-rho)*recorrido);
    double raiz = (double) 1/m;
    tauMin=tauMax*(double)(1-Math.pow(pbest, (raiz)));
    tauMin=tauMin/((m/2)-1)*Math.pow(pbest, (raiz));
}
```

Este método se dispara al inicio del cálculo y cada vez que se actualiza la solución global. Por último, el método que llamamos combinado, ya que básicamente es un AS en el que de la solución heurística que se usa para inicializar el recorrido de feromonas no solo se usa su distancia recorrida para calcular el valor idéntico de todos los arcos, sino que se tiene en cuenta el recorrido obtenido y se refuerza especialmente el mismo al inicio, para incentivar las búsquedas tempranas alrededor de esa solución local. No se espera con esto

[13] Como se vio anteriormente, el valor de τ_{max} se irá refinando sucesivamente a medida que se encuentren mejores soluciones. Nótese que como el valor de τ_{min} también depende de τ_{max}, también su valor se ira refinando al ir encontrando mejores soluciones.

una solución global de mejor calidad que con los métodos anteriores, pero esperamos ver una mejora más rápida de la solución local heurística.

Consecuentemente, esté método solo incorpora un parámetro *k* como propiedad, que determina el grado de este refuerzo local, y se cambia el método que inicializa la feromona, que se hace un poco más complejo ya que no todos los arcos empiezan con el mismo nivel:

```
    private void initFeromona(Escala[] itinerario){
        float tau0 = (float) m/L0;//nivel inicial de feromona
        for(int i = 0;i < n_ciudades;i++){
            for(int j = 0;j < n_ciudades;j++){
                if(i!=j){
                    super.setFeromonaEnArco(i, j, tau0);
                }
            }
        }
        String ciudad_i; String ciudad_j;//incremento de feromona en
        // los arcos pertenecientes a la solución global
        for(int e=0;e<itinerario.length-1;e++){
            ciudad_i = itinerario[e].getDestino();
            ciudad_j = itinerario[e+1].getDestino();
            int i = super.getPosCiudad(ciudad_i);
            int j = super.getPosCiudad(ciudad_j);
            super.setFeromonaEnArco(i, j, tau0*kappa);
```

```
            }

        }
```

7 Pruebas realizadas

7.1 Descripción

Las pruebas realizadas son de dos tipos bien diferenciables. Primero se hará una prueba a la aplicación en sí. Se llevará a cabo el proceso del agregado completo de toda una ciudad con sus correspondientes rutas para verificar las posibilidades de escalamiento del algoritmo de ruteo y la sencillez de tal modificación comparada con lo que requeriría la actualización manual de toda una grilla de 100 x 100. Para probar el funcionamiento en un esquema asimétrico, esta ciudad se creará *ad hoc* con al menos un camino unidireccional, verificándose su funcionamiento.

Las otras pruebas no serán sobre la aplicación sino que se harán usando a la misma como medio para su obtención. Se trata de pruebas sobre los algoritmos que incluirán, primeramente, determinación experimental de los mejores parámetros en cada caso, comparación de la calidad de las soluciones, determinación de longitudes óptimas de pruebas (número de iteraciones) en cada caso, verificación del comportamiento de la metaheurística combinada.

7.1.1 Pruebas de escalamiento del grafo con ruteo dinámico

Como se mencionó durante la descripción de la estructura de la base de datos, la misma está preparada para soportar un esquema de TSP asimétrico[14]. Los algoritmos de la clase **DBconn**, por su parte, también dan soporte de TSP asimétrico mediante el reconocimiento del campo Salida, por el cual una Ciudad incorpora un arco a su tabla de rutas directas para cada entrada de la tabla extremo que tenga a esta ciudad como origen Y que el campo salida correspondiente sea igual a “S”. Al haber trabajado primariamente con un esquema simétrico debido a que se ha copiado el mismo de la realidad, en la que las rutas consideradas son todas bidireccionales, para simplificar la operatoria, la ventana de mantenimiento de base de datos (rutas) al agregar una ruta de una ciudad a otra automáticamente agrega los dos extremos conectados a un arco de la misma longitud, y no toma en cuenta el campo Salida, que al crearse el extremo tomará el valor por defecto “S”. En esta prueba entonces cargaremos los datos de las rutas asimétricas por medio de comando SQL en el administrador de DB2.

Como no hemos encontrado ciudades con recorridos asimétricos reales la ciudad de esta prueba será inventada, con nombre ficticio **Prueba101**. Los datos a ingresar en la tabla ciudad serán:

Nombre	Prueba101
Latitud	30
Longitud	63

Tabla 7 – Valores del nodo Prueba101

[14] Un esquema de TSP asimétrico es aquel en el que al menos un arco es unidireccional, de manera que la distancia entre dos nodos es diferente según sea el sentido de circulación.

Esta ciudad tendrá una conexión bidireccional hacia Córdoba de 190 Km y dos conexiones unidireccionales hacia otro nodo: hacia Palo Negro de 95km y desde Palo Negro 140 Km. En la *fig. 16* se muestra el sector del grafo donde se incluyó **Prueba 101**.

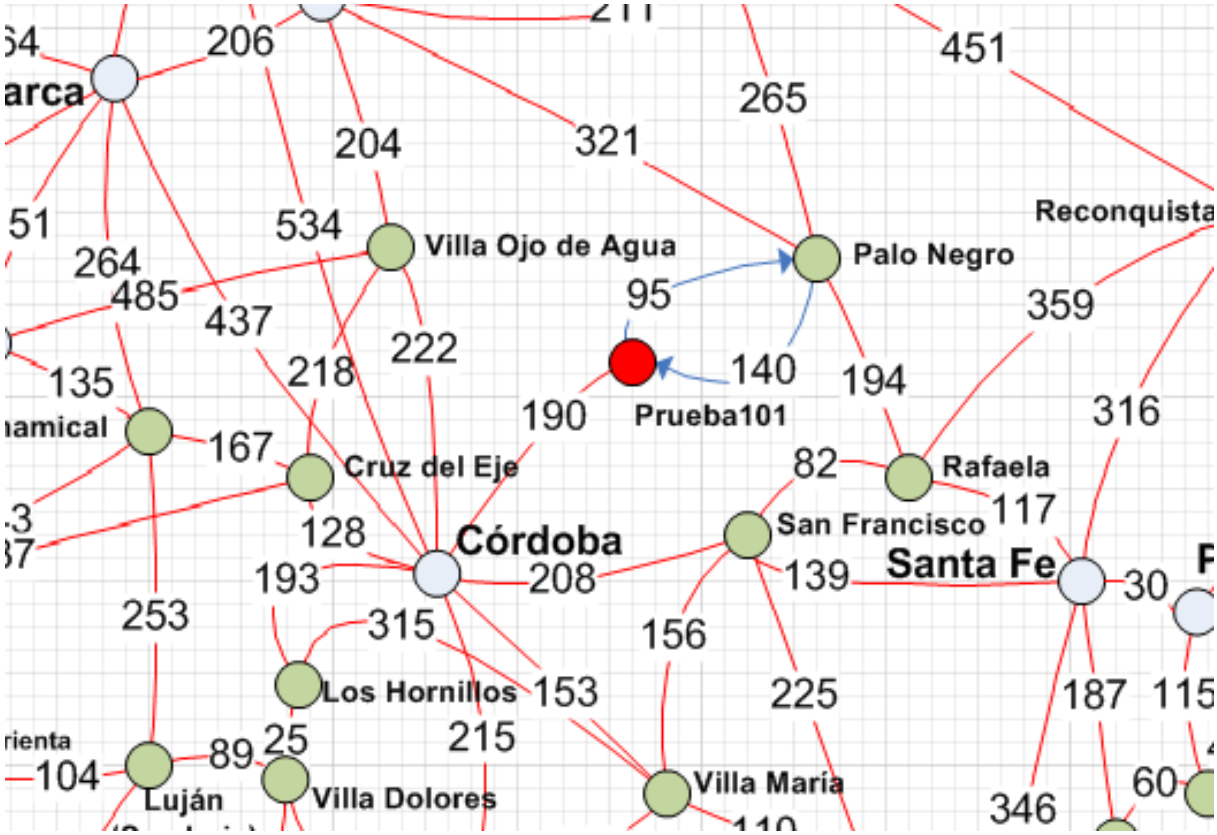

Fig. 16 – Grafo con la inclusión de Prueba101

Para agregar el nodo se procede a cargar los datos correspondientes por medio de la primera ventana de mantenimiento de base de datos, como se ve en la figura 17.

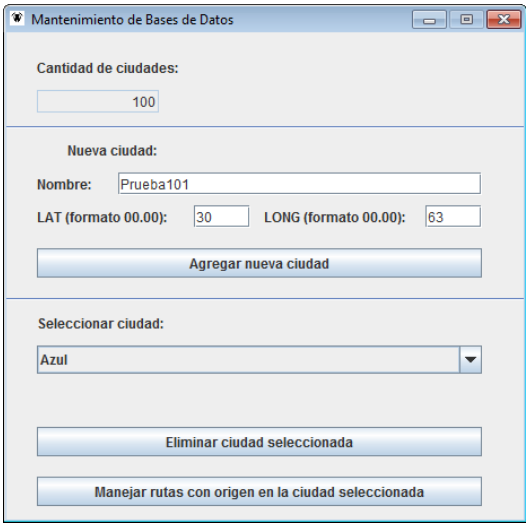

Fig. 17- Agregado de la ciudad 101

Cuando la agregamos se abre automáticamente la ventana de manejo de rutas indicando que la ciudad no está conectada de ninguna manera al grafo por lo que se pide la carga de al menos una ruta (fig 18).

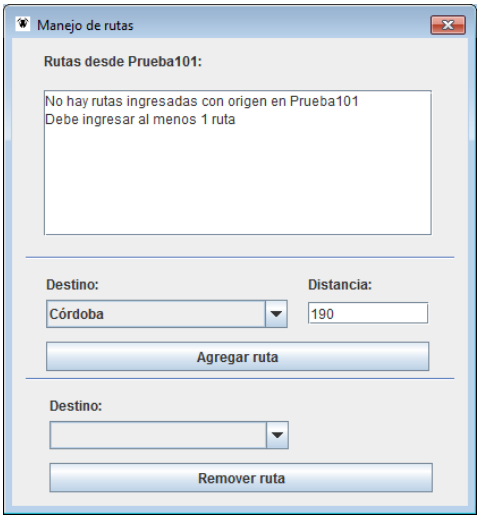

Fig. 18 – Prueba 101, agregado de la primera ruta

Aprovechamos esta oportunidad para cargar la ruta a la ciudad de Córdoba, que como es bidireccional puede ser cargada desde la aplicación.

Para cargar las rutas unidireccionales usamos SQL desde Data Studio. El primer paso es cargar las rutas de 95 y 140 km de longitud:

```
Script3.sql

insert into "Ruta" ("Distancia") values (95)
insert into "Ruta" ("Distancia") values (140)


Total execution time => 313 ms

Script: Script3.sql
Database Name: TF2
Authorization Id (Database): db2admin
System/IP Address : ARSIS-SCHERCHYK/192.168.0.7

User Id (System) : sc6470
```

A continuación verificamos los ID de las últimas dos rutas ingresadas comprobando que corresponden a las distancias correspondientes:

```
ID    Distancia
--------------
761         95

762        140
```

Con estos datos ingresamos los extremos teniendo en cuenta de colocar el valor requerido del campo Salida:

```
Script4.sql

insert into "Extremo" ("Ruta","Ciudad","Salida") values (761,'Prueba101','S')
insert into "Extremo" ("Ruta","Ciudad","Salida") values (761,'Palo Negro','N')
insert into "Extremo" ("Ruta","Ciudad","Salida") values (762,'Prueba101','N')
insert into "Extremo" ("Ruta","Ciudad","Salida") values (762,'Palo Negro','S')


Total execution time => 267 ms

Script: Script4.sql
Database Name: TF2
Authorization Id (Database): db2admin
System/IP Address : ARSIS-SCHERCHYK/192.168.1.4

User Id (System) : sc6470
```

Con lo cual damos por terminado el proceso

7.1.2 Determinación de las mejores combinaciones de parámetros

Para las determinaciones de las mejores combinaciones de parámetros se repasó un amplio abanico de parámetros, probándose cada combinación varias veces y conservando el mejor resultado obtenido de cada una. Estos resultados se utilizarán también más adelante como punto de partida para las pruebas del métodos combinado, en los que se variará en principio solo k y se harán algunas pruebas con algunas variaciones de parámetros para verificar si se apartan o no de lo verificado aquí.

Para los parámetros de AS se probó el siguiente rango de parámetros (tabla 8).

peso heurística			cant	
αMax	αMin	Δ-α		Combinaciones
1	1	1	1	30
peso feromona				
βMax	βMin	Δ-β		
6	1	1	6	
evaporación				
ρMax	ρMin	Δ-ρ		
0,7	0,3	0,1	5	
Parms				
Iteracione	Ensayos	Eurística		
1000	10	AS	10000	
Última combinación				
0				

Tabla 8 – combinaciones de parámetros probados para AS

Para MMAS la cantidad de pruebas se amplía considerablemente dada la mayor cantidad de parámetros. Por otro lado, por sus características, no se esperan estancamientos de MMAS en el corto plazo, por lo que se prevén mejores resultados en pruebas más largas y menor dispersión de resultados (menor desviación estándar). Sin recurrir en este momento a pruebas de larga duración (≥ 1000), que se dejarán para probar más adelante valores elevados de ρ, se duplicará el número de iteraciones con respecto a AS, disminuyendo la cantidad de ensayos a la mitad.

El rango de parámetros empleados se muestra en la siguiente tabla:

peso heurística			cant	
αMax	αMin	Δ-α		Combinaciones
1	1	1	1	180
peso feromona				
βMax	βMin	Δ-β		
5	2	1	4	
evaporación				
ρMax	ρMin	Δ-ρ		
0,98	0,58	0,1	5	
smoothing				
δMax	δMed	δMin		
1	0,5	0	3	
Min-Max				
P_{best}MAX	P_{best}MED	P_{best}MIN		
0,5	0,05	0,005	3	
Parms				
Iteraciones	Ensayos	Eurística		
2000	5	MMAS	10000	
Última combinación				
0				

Tabla 9 Combinaciones de parámetros para MMAS

Como se ve en la tabla 8 se trata de 180 combinaciones, cada una probada 5 veces durante 2000 iteraciones en un grafo de 100 nodos (con 100 hormigas construyendo cada una su recorrido en cada iteración). Esto nos da un tiempo de resolución estimada de 63 horas para esta prueba en particular en una notebook DELL Latitude E6410 con Windows 7, y un tiempo 15% menor con una máquina idéntica pero con SO Ubuntu.

Se realizaran prueba adicionales, similares a las descriptas, pero con la base de 23 ciudades y teniendo en cuenta que prácticamente todas las soluciones tenderán a un valor

mínimo similar, que suponemos óptimo en un número de iteraciones mucho menor que el usado hasta aquí, por lo que el tamaño de estas pruebas deberá ser más acotado.

7.1.3 Búsqueda de las mejores soluciones y establecimiento del tamaño de las muestras

Una vez determinadas las combinaciones base de parámetros a usar, ser realizarán ensayos de AS para verificar hasta qué punto es esperable conseguir mejoras aumentando el número de iteraciones sin llegar al inevitable estancamiento que debería surgir en un esquema que no restablece sus valores de feromonas.

Se irán realizando pruebas gradualmente más largas comparando curvas de progreso para estimar un valor máximo de prueba para AS, luego de lo cual se realizarán 20 pruebas con estos parámetros para evaluar la calidad del sistema. Básicamente se evaluarán distancia recorrida, desviación estándar entre las soluciones y tiempo promedio de solución (hasta la convergencia) y tiempo promedio total (tiempo de proceso aún después de haber hallado la solución).

Para MMAS se realizarán pruebas similares, pero además se variará una vez más el parámetro ρ ya que no se espera un buen desempeño con un valor alto de este parámetro en 2000 iteraciones. Se espera que la calidad de los resultados, con un valor alto de ρ se muy buena y tal vez superior a valores más bajos de la misma en pruebas suficientemente largas. Está por verse si, la dificultad de este problema en particular permite soluciones suficientemente buenas con valores bajos de ρ. Si esto fuera así, para probar mejor este parámetro en particular habría que escalar la magnitud de nuestro TSP aumentando el número de nodos, lo cual quedaría fuera del alcance y las posibilidades de este trabajo, pero podría desarrollarse en estudios sucesivos.

7.1.4 Prueba del algoritmo combinado

Los estudios del método Combinado estarán destinados a verificar si se pueden alcanzar en el corto plazo, resultados de similar calidad a los obtenidos con AS (siempre buscando mejorar la solución base aportada por la heurística voraz) en un menor tiempo. Se probará incrementando el valor de *k* mientras se obtengan mejores performances. Con un valor *k* experimental óptimo alcanzado, se variarán los parámetros *α, β* y *ρ* para observar si existe alguna diferencia de comportamiento del esquema con respecto al AS si el refuerzo combinado.

7.2 Resultados Obtenidos

Nota: Como primer dato obtenido experimentalmente durante los experimentos de ACO, tanto con BD de 32 capitales (cualquiera sea el método usado), como con la BD de 100 ciudades, (usando MMAX), cuando se logra una sintonía correcta de parámetros, luego de un cierta cantidad de iteraciones los resultados tienden a converger a un valor mínimo idéntico que sospechamos óptimo, aunque no se ha comprobado esto de manera definitiva. En todo caso, haciendo esta salvedad y teniendo en cuenta de que, bajo las condiciones adecuadas, estos valores se obtienen de manera consistente, supondremos estos valores como óptimos para el caso de este experimento y las comparaciones de la calidad de los resultados se harán con referencia a estos, en lugar a los valores obtenidos con heurística voraz, como se planteaba durante la planificación de las pruebas. La calidad de la solución quedará definida como su separación relativa respecto de esta solución óptima, especificada entonces como:

$$Q_{sol} = \frac{Sol - Sol_{opt}}{Sol_{opt}}$$

Los valores que entonces tomamos como referencia son:

Base	Óptima (mejor obtenida)	Heurística voraz	Mejora porcentual
23 capitales	11869 Km	11935 Km	0,55%
100 ciudades	20840 Km	24734 Km	18,68%

Los grafos de solución para 23 capitales son:

Fig. 19 – Mejores grafos con 23 capitales. ACO vs. Heurística voraz

Tabla del recorrido de la fig. 19:

Escala:	Distancia:	Acumulado:
0 Neuquén	0	0
1 Santa Rosa	546	546
2 Córdoba	579	1125
3 San Luis	437	1562
4 Mendoza	259	1821
5 San Juán	165	1986
6 La Rioja	438	2424
7 Catamarca	151	2575
8 Santiago del Estero	206	2781
9 Tucumán	162	2943
10 Salta	316	3259
11 Jujuy	115	3374
12 Formosa	953	4327
13 Resistencia	168	4495
14 Corrientes	20	4515
15 Posadas	291	4806
16 Paraná	780	5586
17 Santa Fe	30	5616
18 La Plata	569	6185
19 Viedma	923	7108
20 Rawson	502	7610
21 Rio Gallegos	1182	8792
22 Ushuaia	577	9369
23 Neuquén	2500	11869

Los grafos comparados de mejores recorridos para 100 ciudades se muestran en la *figura 20*. El mejor recorrido para ACO se obtuvo mediante MMAS (en múltiples oportunidades, como se mencionó y se puede ver en los reportes), mientras que con AS no se pudo lograr nunca un valor ni siquiera cercano.

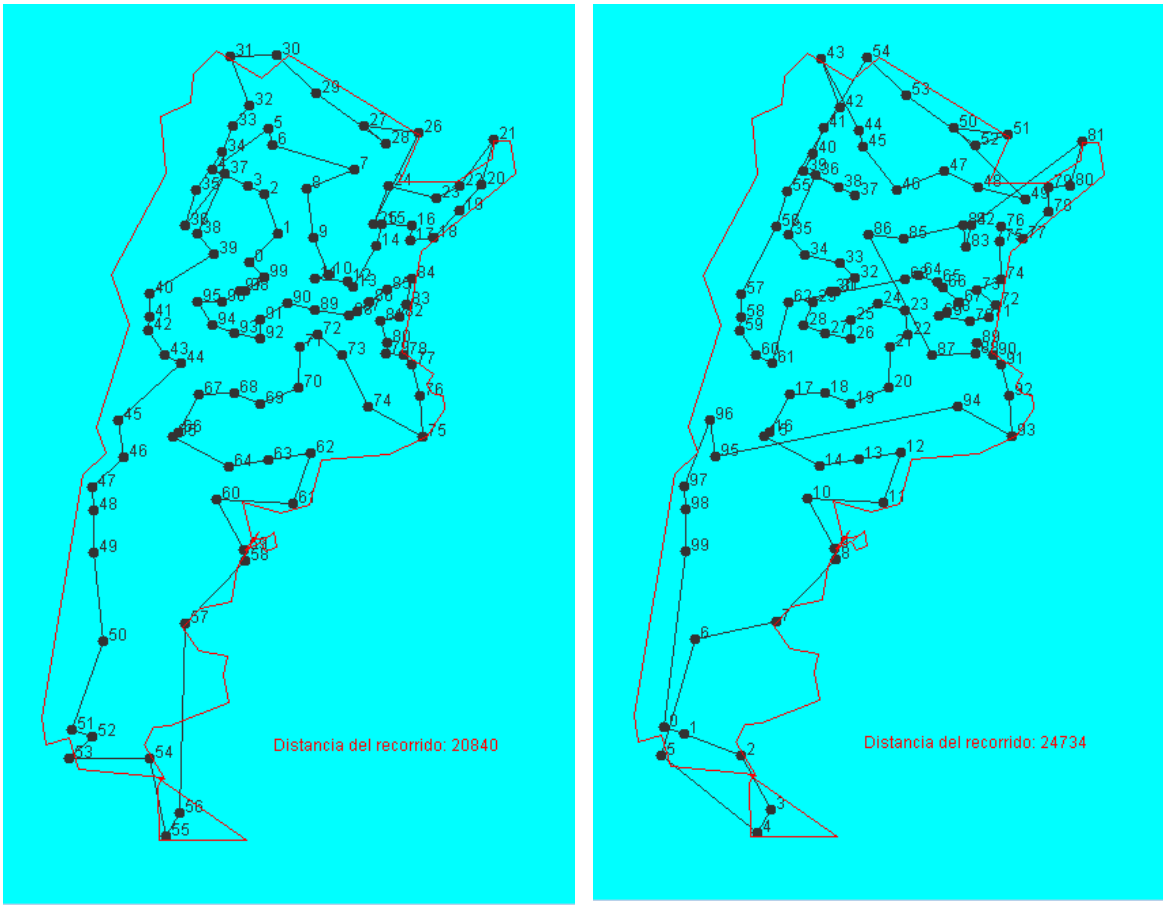

Fig. 20- Mejores grafos con 100 capitales. ACO vs. Heurística voraz

7.2.1 Agregado del nodo 101

En principio verificamos en mantenimiento de BD que estén correctamente cargados los datos[15] (fig. 19).

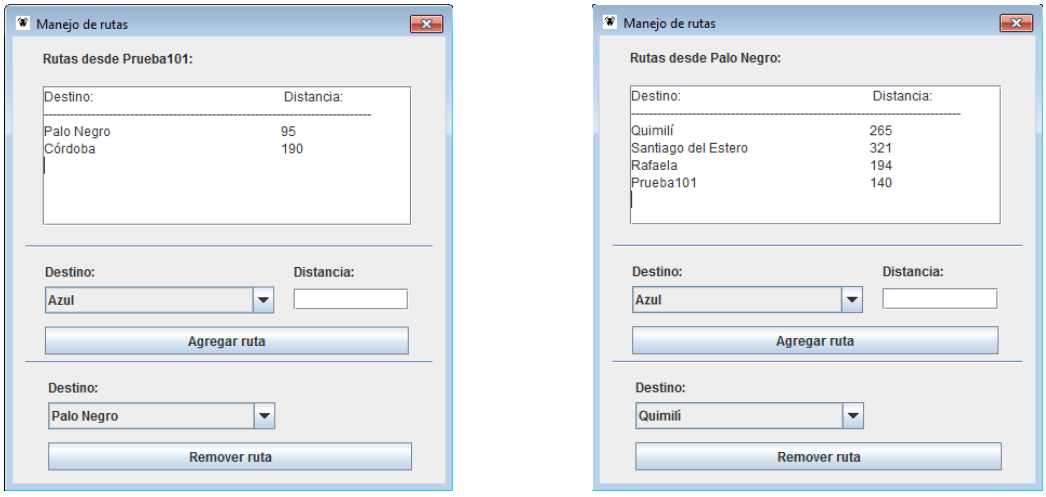

Fig. 21 – Rutas entre Prueba101 y Palo Negro

Como se puede apreciar, la ruta de **Prueba101** a Palo Negro quedó cargada con 95 Km y en el sentido inverso 140 Km.

Para probar que no haya errores en la creación de la matriz de arcos de distancias hacemos una prueba de búsqueda heurística, que arrojó el siguiente resultado:

---

[15] Nota: las datos desplegados en las ventanas de Mantenimiento de BD (combo boxes y áreas de texto) no se extraen de los *queries* a la BD directamente sino que salen de datos ya elaborados y cargados en el *array* de ciudades, como ser las tablas de rutas.

Escala:	Distancia:	Acumulado:
0 El Calafate	0	0
1 El Cerrito	91	91
2 Rio Gallegos	210	301
3 Rio Grande	357	658
4 Ushuaia	220	878
5 Río Turbio	1004	1882
6 Perito Moreno	1035	2917
7 Comodoro Rivadavia	378	3295
8 Rawson	394	3689
9 Puerto Madryn	71	3760
10 Valcheta	302	4062
11 Viedma	287	4349
12 Bahía Blanca	270	4619
13 Rio Colorado	164	4783
14 Choele Choel	140	4923
15 Neuquén 2	24	5147
16 Catriel	141	5288
17 Santa Isabel	209	5497
18 Victorica	143	5640
19 Santa Rosa	146	5786
20 Trenque Lauquen	160	5946
21 Rufino	217	6163
22 Venado Tuerto	95	6258
23 Marcos Juárez	156	6414
24 Villa María	110	6524
25 Río Cuarto	135	6659
26 Vicuña Mackenna	91	6750
27 Villa Mercedes	99	6849
28 San Luis	92	6941
29 Luján (San Luis)	104	7045
30 Villa Dolores	89	7134
31 Los Hornillos	25	7159
32 Córdoba	193	7352
33 Cruz del Eje	128	7480
34 Chamical	167	7647
35 La Rioja	135	7782
36 Catamarca	151	7933
37 Santiago del Estero	206	8139
38 Termas de Rio Hondo	56	8195
39 Tucumán	86	8281
40 Cafayate	241	8522
41 Salta	186	8708
42 Jujuy	115	8823
43 La Quiaca	289	9112
44 Joaquín V. Gonzalez	579	9691

45	Monte Quemado	161	9852
46	Quimilí	245	10097
47	Sáenz Peña	250	10347
48	Resistencia	157	10504
49	Corrientes	20	10524
50	Formosa	188	10712
51	Clorinda	104	10816
52	Pirané	210	11026
53	Los Blancos	433	11459
54	Profesor Mazza	274	11733
55	Belén	838	12571
56	Chilecito	221	12792
57	San Juán	487	13279
58	Mendoza	165	13444
59	Tunuyán	81	13525
60	San Rafael	157	13682
61	General Alvear	90	13772
62	La Chañarienta	389	14161
63	**Prueba101**	**601**	**14762**
64	Palo Negro	**95**	14857
65	Rafaela	194	15051
66	San Francisco	82	15133
67	Santa Fe	139	15272
68	Paraná	30	15302
69	Nogoyá	108	15410
70	Victoria	43	15453
71	Rosario	60	15513
72	Gualeguay	180	15693
73	Gualeguaychú	84	15777
74	Concepción del Uruguay	64	15841
75	Villaguay	113	15954
76	Concordia	110	16064
77	Curuzú Cuatiá	223	16287
78	Mercedes	79	16366
79	Paso de los Libres	118	16484
80	Santo Tomé	190	16674
81	Posadas	155	16829
82	Oberá	91	16920
83	Puerto Iguazú	278	17198
84	Goya	812	18010
85	Esquina	111	18121
86	Reconquista	574	18695
87	Zarate	730	19425
88	Luján	62	19487
89	Buenos Aires	65	19552
90	La Plata	55	19607
91	Dolores	169	19776

```
92 Mar del Plata              205             19981
93 Azul                       258             20239
94 Junín                      290             20529
95 Villa Ojo de Agua          801             21330
96 Chos Malal                 1511            22841
97 Zapala                     196             23037
98 San Martín de los Andes    245             23282
99 Bariloche                  180             23462
100 Esquel                    284             23746
101 El Calafate               1267            25013
```

El grafo resultante es el siguiente (la posición de Prueba101 ha sido resaltada con rojo):

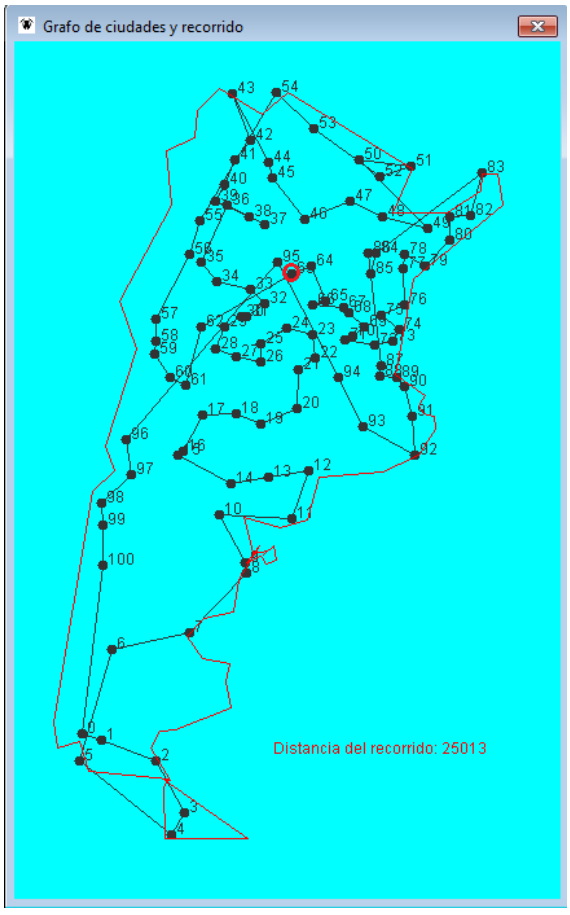

Fig. 22 – Grafo con 101 ciudades y TSP asimétrico (heurística voraz)

7.2.2 Mejor combinación de parámetros

En lo que respecta a AS, comenzaremos por determinar el valor apropiado para *β* (peso de la feromona en la elección probabilística). Como se indicó anteriormente, se realizaron pruebas para valores de *β* entre 1 y 6. Los resultados de las pruebas obtenidas[16] nos permiten en principio evaluar los promedios de los resultados de todas las pruebas agrupados por este parámetro.

β					
1	2	3	4	5	6
23225	22088	21910	21765	21988	22107
23106	22265	21640	21924	21995	21961
22937	21832	21786	21804	21728	21904
23185	21952	21565	21725	21680	21830
23583	22005	21984	21928	21957	21910

Tabla 10 – AS, resultados agrupados por el valor de β.

En la tabla 10 se indican con una escala de colores que va desde verde (mejor), pasando por amarillo (neutro) a rojo (peor) las distancias recorridas en cada prueba. Cada valor indicado representa el mejor valor obtenido entre 10 pruebas de 1000 iteraciones (cada conjunto de pruebas se hizo con valores distintos del parámetro *ρ*). Una mirada a esta tabla nos muestra que los mejores valores se agrupan entre los valores de *β* entre 3 y 5. Puede verse que definitivamente, los resultados para *β = 1* son definitivamente los peores.

Para hacer una comparación más fina de estos valores compararemos el promedio de los valores agrupados para mostrarlos en un diagrama de barras:

[16] Ver datos obtenidos en el reporte "**AS, 30 combinaciones 1000x10.xls"**

Fig. 23 – Promedio de mejores soluciones agrupadas por β en pruebas de 1000 iteraciones en AS

En los resultados para $\beta = 1$, podríamos indagar un poco más sobre el motivo de tan mal desempeño, ya que en algunos trabajos anteriores algunos autores consideran apropiados valores de β entre 1 y 5, mientras que claramente en este caso el valor $\beta = 1$ quedaría definitivamente descartado. Si tenemos en cuenta que los valores de β dan mayor peso al rastro de feromona a medida que disminuyen, cabe preguntarse si su bajo rendimiento se debe a que no se le da suficiente peso relativo a la heurística y por lo tanto la búsqueda de soluciones es demasiado aleatoria, o bien puede ser que los rastros acumulen demasiada feromona dando lugar a estancamiento. Si esto último no ocurre, tal vez obtengamos mejores soluciones con este nivel de β a largo plazo, en comparación con niveles mayores.

Para verificar esto tomamos 10 pruebas de 10.000 iteraciones, con $\rho = 0,5$, y observamos el progreso de las soluciones: ([17])

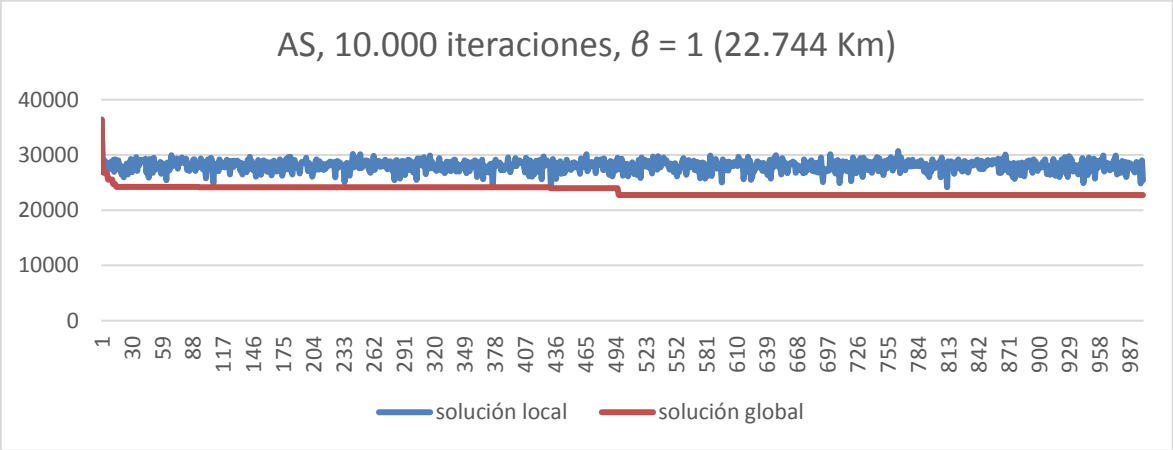

Fig. 24 – AS 10.000 Iteraciones β = 1

En primer lugar queda claro que la búsqueda está viva (no hay estancamiento) después de 10.000 iteraciones. Se ve también que hay alguna mejora ocasional, pero la búsqueda sigue manteniéndose dentro de un rango cuyos valores medios no van disminuyendo. Esto nos indica que si bien es posible esperar alguna mejora en el largo plazo, la performance con esta combinación de parámetros en AS es muy pobre y queda descartada. Más aún, las

[17] Datos del reporte "***AS, 10000 iter x 10 beta 1.xls***"

figuras 25 ([18]) y 26 ([19]) nos muestran que para valores de $\beta = 2$ y $\beta = 3$ también tenemos mejoras a largo plazo, sumado a que ya teníamos mejores valores desde el comienzo.

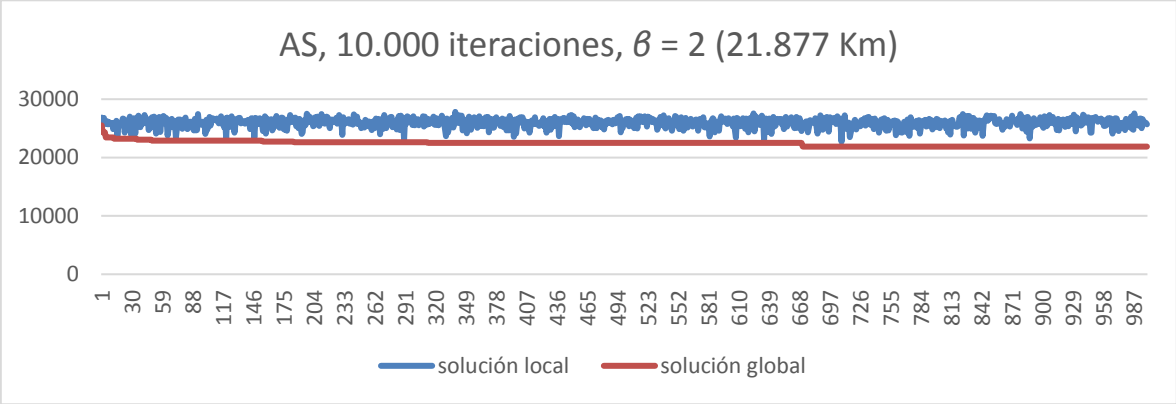

Fig. 25 AS 10.000 iteraciones β = 2.

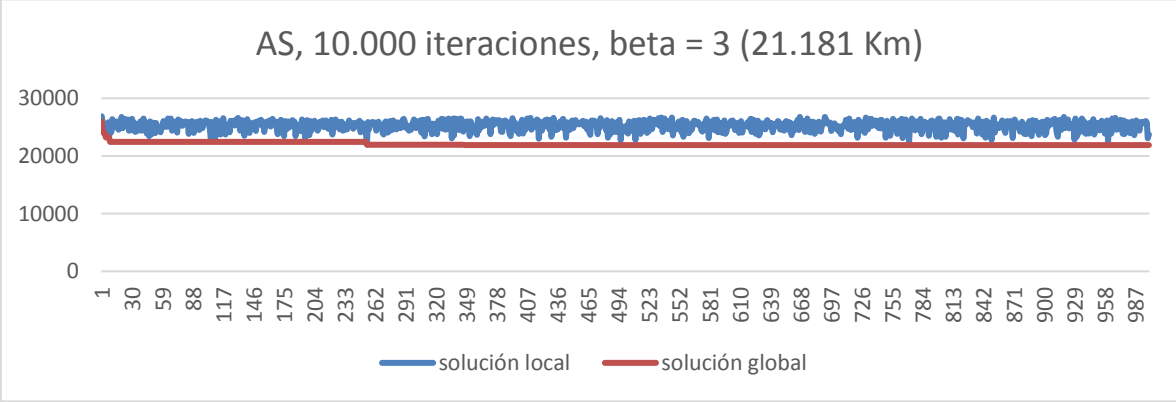

Fig. 26 – AS 10.000 iteraciones β = 3

En nuestro escenario, usando AS el valor de $\beta = 3$ aparece como indicado tanto para pruebas de corto como largo plazo si tenemos en cuenta la calidad de las soluciones. Antes

[18] Datos del reporte "***AS, 10000 iter x 10 beta 2.xls***"

[19] Datos del reporte "***AS, 10000 iter x 10 beta 3.xls***"

de hacer una evaluación final de *β,* estudiaremos el comportamiento de las desviaciones estándar de las muestras, lo que nos permite evaluar si hay una consistencia en la calidad de las soluciones posibles. Esto nos permitirá evaluar qué tan importante será la cantidad de ensayos llevados a cabo.

β					
1	2	3	4	5	6
245,45	267,64	176,75	191,11	80,71	81,76
282,50	175,72	208,57	81,95	93,73	74,86
342,77	257,04	182,48	135,22	149,51	109,06
329,76	260,43	203,53	186,28	226,59	215,81
183,74	151,86	54,90	52,40	72,36	179,55

Tabla 11 - AS, desviaciones estándar agrupadas por β

Cada uno de los valores de la tabla 11 muestra la desviación estándar de los resultados de cada una de las combinaciones bajo estudio (un tamaño de muestra de 10). Un primer vistazo nos muestra que los mejores resultados se agrupa en los valores más altos, siendo *β = 3* no tan bueno en cuanto a la desviación estándar. La siguiente gráfica de barras nos muestra los promedios de estos valores.

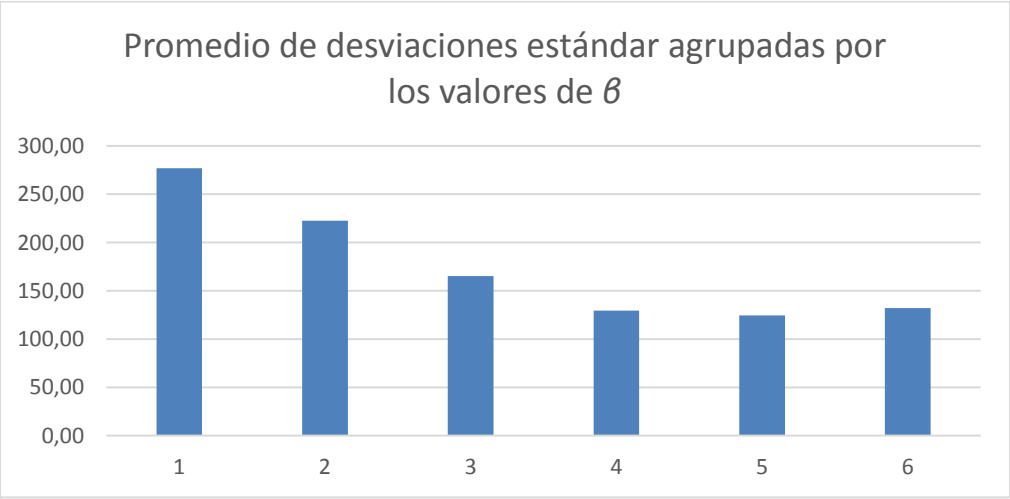

Fig. 27 - Promedio de desviaciones estándar agrupadas por β *en pruebas de 1000 iteraciones en AS*

Los valores de SD para $\beta = 4$ y $\beta = 5$ son aproximadamente un 25% mejores. Debería tenerse en cuenta al momento de diseñar una prueba determinada si debemos usar AS, si no tenemos la posibilidad de hacer varios ensayos para comparar soluciones tal vez un valor de 4 sea más apropiado que 3, de lo contrario $\beta = 3$ sería la mejor elección. Por el lado del valor de ρ, los valores agrupados obtenidos se muestran en la siguiente tabla:

ρ				
0,3	0,4	0,5	0,6	0,7
23225	23106	22937	23185	23583
22088	22265	21832	21952	22005
21910	21640	21786	21565	21984
21765	21924	21804	21725	21928
21988	21995	21728	21680	21957
22107	21961	21904	21830	21910

Fig. 28- AS, resultados agrupados por el valor de ρ

Vemos en la tabla buenos resultados alrededor de valores de $\rho = 0,5$ y $\rho = 0,6$. Esto se confirma viendo la gráfica de los promedios de estos valores, donde vemos que estos valores son claramente mejores que cualquiera otro valor, con 0,6 siendo marginalmente menor.

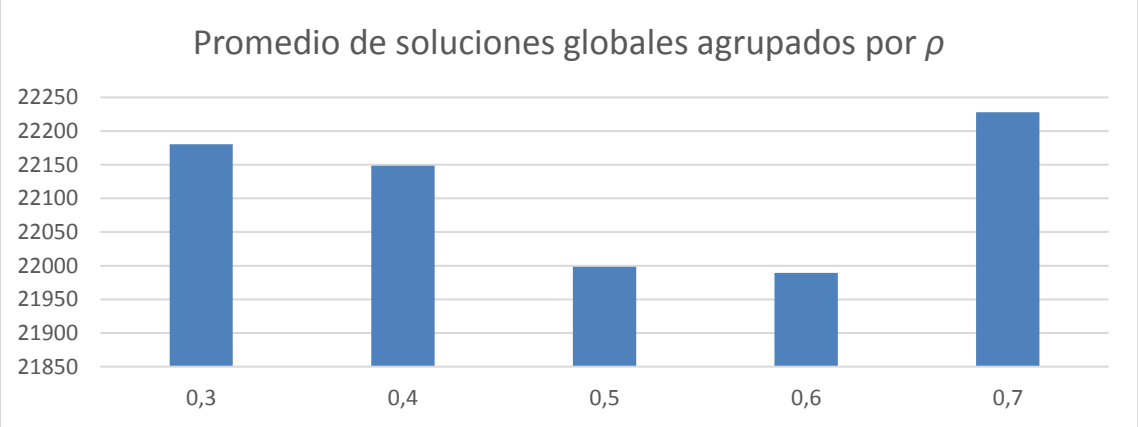

Fig. 29 Promedio de soluciones globales agrupadas por ρ en pruebas de 1000 iteraciones en AS

Pasemos entonces a la evaluación de MMAS.[20] Como hicimos con AS, estudiamos la respuesta a los cambios de β. Partiendo de nuestra experiencia con AS y esperando un comportamiento similar hasta cierto punto de los parámetros α y β no consideraremos valores de $\beta = 1$ y $\beta = 6$, ya que debemos acotar la prueba en este caso, ya que la cantidad de combinaciones, dado que en este caso manejamos un mayor número de parámetros, haría demasiado largas las pruebas e pos de resultados que no son prometedores. Para empezar veremos directamente la gráfica de promedios (las tablas de datos no se muestran aquí por una razón de espacio, se adjuntan los mismo en la planilla correspondiente[21].

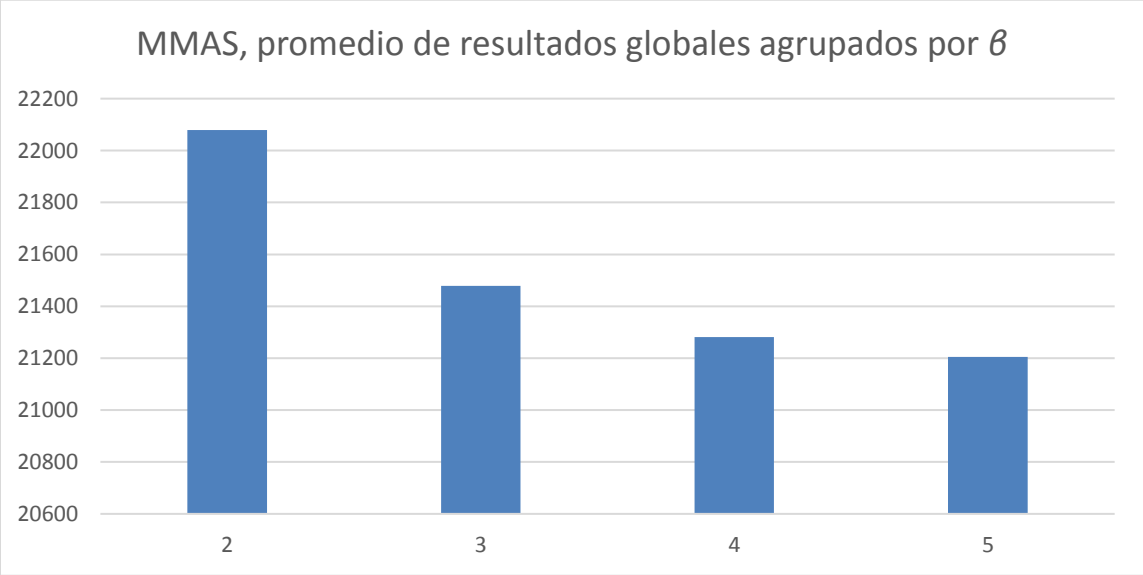

Fig. 30 - Promedio de soluciones globales agrupadas por β en pruebas de 2000 iteraciones en MMAS

[20] Los valores para la determinación de parámetros de MMAS se encuentran en el reporte "***MMAS 180 comb - beta 2-5 - 2000 x 5.xls***"

[21] Ver **"MMAS 180 comb - beta 2-5 - 2000 x 5.xls"**

Los valores de ρ, como ya se mencionó y de acuerdo a la teoría, son muy importantes en MMAS, pero a su vez son muy sensibles al número de iteraciones, ya que si bien se espera un buen desempeño para valores altos de este parámetro, esto se debería ver en corridas largas.

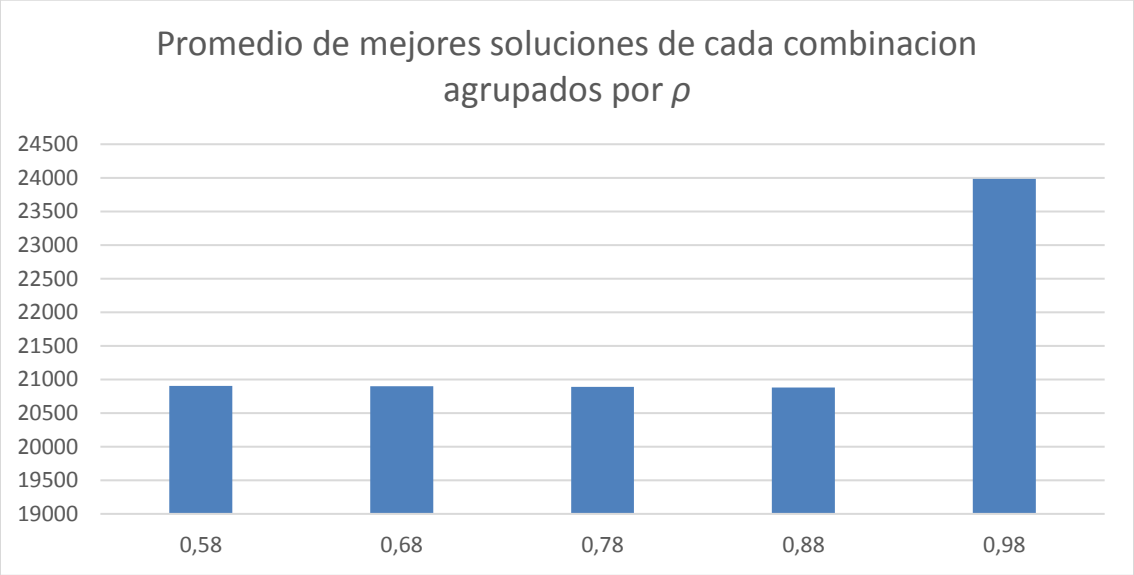

Fig. 31 - Promedio de soluciones globales agrupadas por ρ en pruebas de 2000 iteraciones en MMAS

Los resultados desplegados no permiten definir de manera determinante cuál es el mejor valor de ρ para esta cantidad de iteraciones. Observando el reporte, vemos que el valor más bajo (20.840 Km) se obtiene un mayor número de veces para ρ = 0,88 (sin ser aún una diferencia notable).

Por otro lado, el estudio de las desviaciones estándar muestra que con ρ = 0,88 el rendimiento en realidad no es del todo coherente siendo el comportamiento de los tres valores menores mejor que este, alcanzando un menor valor para ρ = 0,78 (fig. 32), que será el valor a tomar como punto de partida.

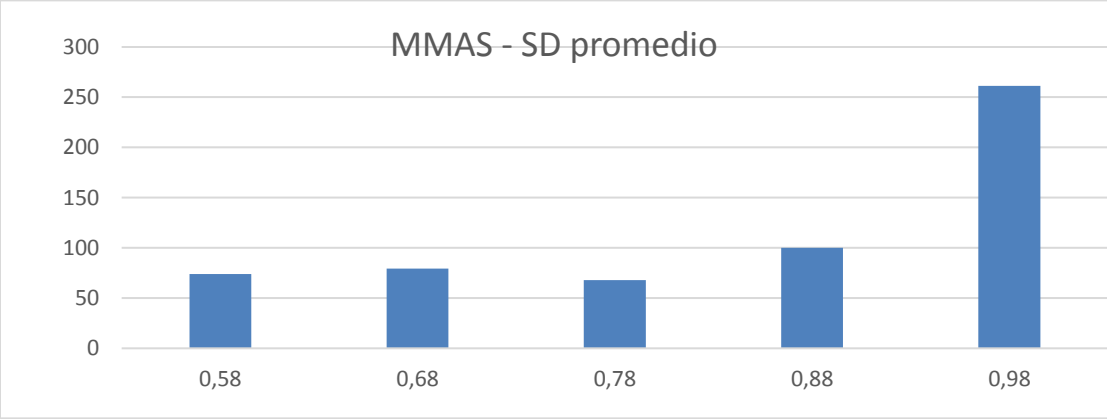

Fig. 32 - Promedio de desviaciones estándar agrupadas por ρ en pruebas de 2000 iteraciones en MMAS

Los resultados agrupados según el coeficiente *δ*, usado en el PTS, muestran que este mecanismo tiene mucha influencia en los resultados, aún en corridas cortas como las de estas pruebas. En la fig. 33 se muestran los promedios de los resultados agrupados por *δ.* Las diferencias no son significativas, pero se espera una influencia más marcada en corridas largas.

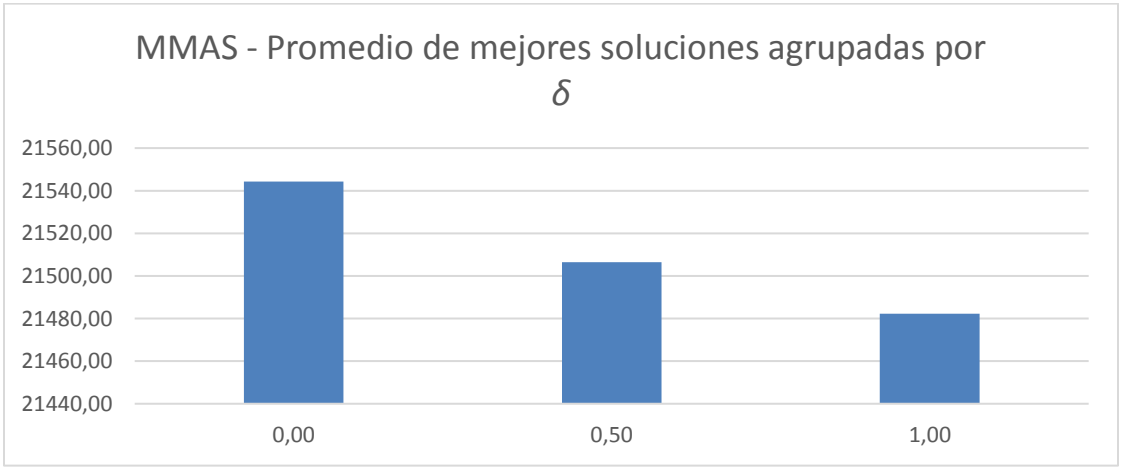

Fig. 33 – Promedio de mejores soluciones agrupadas por δ en MMAS

Los análisis de P_{best} no muestran diferencias significativas:

Pbest		
0,005	0,05	0,5
promedio resultados		
21519,73	21511,15	21502
promedio SD		
111,6146	118,5308	119,3532

Tabla 12 - Análisis de P_{best} para 2.000 iteraciones

A este respecto cabe hacer dos reflexiones. La primera es que se trabajó con tres valores de P_{best} utilizados con éxito en experimentos anteriores[22], de los que se probó un buen desempeño. En segundo lugar, observando los resultados, vemos que, salvo para $\rho = 0,98$ eventualmente todas los demás parámetros de alguna manera nos dan resultados muy buenos en comparación tanto con la heurística voraz como con el AS, incluso llegan fácilmente al valor mínimo de 20.840 Km que sospechamos es el óptimo del escenario, por lo que parecería ser que la poca sensibilidad al cambio de parámetros ocurra debido a que el escenario no es lo suficientemente complejo como para probar acabadamente este sistema.

7.2.3 Pruebas de MMAS con configuraciones probadas

Se intentará seguidamente, y a partir de la información recabada, estudiar el comportamiento de una configuración ideal para este escenario, la calidad de sus soluciones y su capacidad de encontrar el mínimo de la prueba (20.840 Km). Verificar si al menos se puede alcanzar esa meta en corridas largas con $\rho = 0,98$, no esperándose a esta altura un desempeño muy eficiente de este valor de ρ en este escenario.

[22] T. Stützle, H.H. Hoos / Future Generation Computer Systems 16 (2000) 889–914

La tabla siguiente muestra los resultados con su análisis estadístico de cinco pruebas MMAX, *α=1, ϐ=5, ρ=0,78, δ=0,5 y P_{best}, 0,05* para 1000, 2000, 5000 y 10000 iteraciones en cada prueba de 50 ensayos.[23]

Análisis estadístico, MMAS				
Ensayos x Iteraciones	50 x 1000	50 x 2000	50 x 5000	50 X 10000
Promedio	20909,36	20902,12	20882,56	20881,64
Error estándar	4,30	3,10	2,65	2,79
Mediana	20891,00	20891,00	20891,00	20891,00
Moda	20891,00	20891,00	20891,00	20891,00
Desviación estándar	30,41	21,94	18,77	19,71
Varianza de la muestra	924,72	481,41	352,25	388,52
Curtosis	1,22	0,25	1,72	0,99
Sesgo	1,00	0,50	-1,91	-1,72
Rango	165,00	101,00	51,00	51,00
Valor mínimo	20840,00	20840,00	20840,00	20840,00
Valor máximo	21005,00	20941,00	20891,00	20891,00
Sumatoria	1045468,00	1045106,00	1044128,00	1044082,00
Cantidad de mínimos	1,00	1,00	8,00	9,00
Nivel de confianza (99,9%)	15,05	10,86	9,29	9,76

Tabla 13 Análisis estadístico de MMAS variando el largo de las pruebas

En la tabla 20 vemos una continua mejora en la calidad de los resultados en cuanto a su promedio y la cantidad de óptimos obtenidos. Si bien en todos los casos se alcanzó un valor mínimo de 20.840, con mil iteraciones solo hubo una ocurrencia en la totalidad de los 50 ensayos, mientras que para 10.000 iteraciones la cantidad fue de 9, y con 5.000 fue apenas menor, llegando a 8. Con estos resultados, el rendimiento en cuanto a calidad alcanzada por tiempo de procesamiento resulta mejor con 5.000 iteraciones. El tiempo promedio de

[23] Datos extraídos de los reportes "***MMAS 50 x 1000 .xls***", "***MMAS 50 x 2000 .xls***", "***MMAS 50 x 5000 .xls***" y "***MMAS 50 x 10000 .xls***"

una prueba de 5.000 iteraciones con esta configuración y en la estación con Linux fue de 9'35''.

En la última prueba de MMAX $\alpha=1$, $\beta=5$, $\rho=0,98$, $\delta=0,5$ y P_{best}, $0,05$, cinco pruebas de 50.000 iteraciones[24] para estudiar su comportamiento a largo plazo con ρ elevado. Estos fueron los resultados obtenidos:

Distancia	Conv.	T-conv	T-final
20840	8644	856,41	4991,45
20840	4806	480,18	4980,60
20840	30369	2998,88	4939,13
20840	8233	813,33	4943,28
20840	7836	773,99	5302,10

Tabla 14 – MMAS ρ = 0,98, 50.000 iteraciones

Es interesante notar que, en general, $\rho = 0,98$ no requiere de plazos tan largos para converger. En los cinco casos se llegó a la mejor solución, y esto fue antes de las 10.000 iteraciones. Solo en un caso se requirieron 30.369 iteraciones para llegar a esa solución.

7.2.4 Pruebas de metaheurística combinada

El último grupo de pruebas corresponde al algoritmo combinado y se tratará de evaluar su rendimiento en el corto plazo por lo que las pruebas serán todas de 1.000 iteraciones. Como no se cuenta con la opción de pruebas múltiples para esta variante, todas las pruebas se harán mediante reportes separados para cada valor de *k*.

[24] Ver "***MMAS rho 098 beta 4 50000 it.xls***"

Empezaremos por pruebas con una base de 23 capitales[25] en las que se puede alcanzar rápidamente una convergencia hacia un valor óptimo o cercano a este. Haremos pruebas idénticas con AS simple y combinado y se medirá el promedio de los recorridos, el de la cantidad de iteraciones necesarias (I_{conver}) y el del tiempo necesario para la convergencia (T_{conver}) en todos los casos probados.

promedios	**AS**	**combinado**					
		k=2	**k=4**	**k=8**	**k=16**	**k=32**	**k=64**
distancia	11869,92	11869	11869,64	11869,98	11869,64	11869,32	11869,64
I_{conver}	164,18	142,04	106,67	156,66	165,19	178,01	174,41
T_{conver}	0,24	0,20	0,16	0,24	0,23	0,25	0,24

Tabla 15 - Prueba AS/Combinado 23 capitales

La *tabla 15* muestra que efectivamente se verifica una mayor velocidad en la convergencia con la estimulación temprana de feromona en la solución local creada por la heurística voraz en escalas reducidas. Nótese que el promedio de las distancias no varía significativamente ya que en casi todos los casos se llega al resultado de 11.689 Km (en el caso notable de *k=2* los 100 ensayos de 1000 iteraciones arrojaron todos ese mismo valor, en los otros

[25] Datos de la tabla extraídos de "***AS - 23 nodos 100 x 1000.xls***", ***"combinado k2 - 23 nodos 100 x 1000.xls***", "***combinado k4 - 23 nodos 100 x 1000.xls***", "***combinado k8 - 23 nodos 100 x 1000.xls***", "***combinado k16 - 23 nodos 100 x 1000.xls***", "***combinado k2 - 32 nodos 100 x 1000.xls***" y "***combinado k2 - 23 nodos 100 x 1000.xls***".

hubo solo unas muy pocas excepciones). La convergencia, por otro lado, muestra resultados de mejora significativa para *k=4*.

Haremos ahora la misma prueba en escala de 100 ciudades[26].

promedios	AS	combinado					
		k=2	k=4	k=8	k=16	k=32	k=64
distancia	22240,75	22237,02	22291,49	22252,63	22229,57	22216,16	22224,75
I_{conver}	550,9	565,34	536,91	524,26	497,17	509,77	496,31
T_{conver}	76,50	79,73	63,57	62,11	54,50	55,82729	53,83957

Tabla 16 - Prueba AS/Combinado 100 capitales

Vemos dos diferencias evidentes: primero se ve una mayor variación de los valores de distancia y segundo se nota que para *n=100* hay una mayor sensibilidad a valores mayores de *k*. Los mejores desempeños se alcanzaron para *k entre 16 y 64*. No se probaron valores mayores, pero la tendencia a partir de *k=16* se estabiliza.

[26] Datos de la tabla extraídos de "***AS - 100 nodos 100 x 1000.xls***", ***"combinado k2 - 100 nodos 100 x 1000.xls***", "***combinado k4 - 100 nodos 100 x 1000.xls***", "***combinado k8 - 100 nodos 100 x 1000.xls***", "***combinado k16 - 100 nodos 100 x 1000.xls***", "***combinado k2 - 100 nodos 100 x 1000.xls***" y "***combinado k2 - 100 nodos 100 x 1000.xls***".

8 Conclusiones

8.1 Calculo dinámico de la tabla de distancias

El procedimiento llevado a cabo y anteriormente descripto, requirió de menos de tres minutos (sin contar la búsqueda de las distancias en la red, ya que el nodo se creó para la prueba, con lo cual habría que agregar un minuto más de tiempo. Si hubiésemos debido agregar manualmente la información necesaria para completar la matriz, habría sido necesario buscar e ingresar 5.000 distancias, en el caso de TSP simétrico y 10.000 para el asimétrico. Si estimamos un minuto para buscar la distancia requerida e ingresar correctamente, esto nos daría un tiempo estimado de 80 horas para TSP simétrico y 160 para el asimétrico. La diferencia se hace evidentemente más que significativa y justifica ampliamente la adopción de este método.

8.2 Parámetros a utilizar y resultados obtenidos en pruebas de metaheurísticas

8.2.1 AntSystem

La mejor combinación detectada es de $\alpha=1, \beta=4$ y ρ puede ser 5 o 6, y esta parecería no dar lugar a mucho margen de variación.

Aunque este sistema funciona muy bien para n=23 (misma calidad de resultados que MMAS, en tiempos muy cortos), su rendimiento para n=100 es pobre.

8.2.2 MinMaxAntSystem

La mejor combinación detectada es *α=1, ϐ=5, ρ=0,78, δ=0,5 y P_{best}, 0,05.* En el caso de MMAS puede haber más lugar a alguna sintonía fina para corridas más largas, lo cual no pudo ser probado dentro de los límites de nuestras pruebas. Es también de esperar algún comportamiento distinto en un escenario más complejo.

MMAS ha probado un excelente desempeño, aún para n=100, donde con pocas iteraciones es capaz de alcanzar valores que intuimos óptimos. La buena performance dentro de rangos amplios de variables permite presumir que está lejos de su techo y se podría aplicar con éxito en dominios mucho más complejos.

8.2.3 Variante Combinada del AS

Las diferencias en la calidad de la solución, según la *tabla 15* son mínimas porque, recuérdese, los resultados para un escenario de 23 ciudades son consistentemente buenos aún para AS y varían muy poco de un mínimo de 11.869 Km, de modo que no vemos mejoras significativas en este aspecto. Por el lado del tiempo empleado, por otro lado, vemos que el método combinado tienen a converger con muchas menos iteraciones, y por lo tanto más rápido. En el caso de *k=4* vemos que la solución converge en un tiempo 25% menor que para AS. Más allá de *k=4* las mejoras desaparecen.

Para *n=100*, por otro lado, las mejoras de esta variante se extienden incluso para valores más altos de *k*, y si bien las variaciones de la calidad de la solución son pequeñas (alrededor del 0,1 %), son bastante más significativas que para n=32 ya que con tantos nodos AS presenta una mayor dispersión en los valores de su solución.

En cuanto, a los tiempos, las diferencias se hacen, como en el otro caso, mucho más conspicuas. Se pudo medir una disminución del promedio de los tiempos de convergencia con respecto al AS de casi un 30%, es decir un 5% más de mejora que en el caso de *n=23*.

Para *n=100* esto toma mayor importancia ya que los tiempos de computación empiezan a ser grandes, cuando antes la obtención de resultados era prácticamente instantánea.

Se puede concluir entonces que esta variante bien vale la pena de ser implementada, por su sencillez, y su mejora en la rapidez sin afectar (incluso mejorando) la calidad de los resultados, y sin haberse notado situaciones de estancamiento con los parámetros utilizados. Esta idea podría también ser extendida a otros sistemas de ACO en pruebas futuras dado que los resultados recabados nos ofrecen perspectivas alentadoras.

9 Glosario

ACO Ant Colony Optimization (Optimización por Colonia de Hormigas)

ACS Ant Colony System. Una versión mejorada de AS.

algoritmo voraz Método heurístico de múltiples aplicaciones. Se basa en el establecimiento de un criterio determinado para tomar una decisión de manera determinística en cada paso de la resolución de un problema, si tener en cuenta el efecto de esa elección local en la calidad de la solución global final. En nuestro caso se enuncia como sigue: desde cada ciudad, pasar a la siguiente más cercana no visitada.

arco Enlace entre dos nodos de un grafo, en nuestro caso representa un camino o combinación de caminos que permiten ir de una ciudad a otra.

AS Ant System. Desarrollado por Dorigo et al., fue uno de los primeros y más probados sistemas de ACO

autocatalítico Autoacelerador. Efecto que en un procesos dado que permite que un procesos se refuerce a medida que el mismo se lleva a cabo.

BWAS Best-Worst Ant System. Un versión mejorada de AS

ceteris paribus Criterio experimental que indica que se prueba el efecto de la variación de un parámetro en un entorno dado, manteniendo constantes todos los demás.

convergencia En un entorno de redes, cuando se emplea ruteo dinámico, se dice que este alcanzó una condición de convergencia si todos los nodos que la componen tienen información completa de la topología de su dominio.

En el caso de un problema como el TSP resuelto a través de ACO, se dice que el problema converge cuando se llega, a una solución que no puede mejorarse con el esquema de solución propuesto (puede que se haya alcanzado la solución óptima)

EAS Elitist Ant System. Una versión mejorada de AS

ensayos En nuestro problema TSP, denominamos ensayos a una conjunto independiente de iteraciones de un mismo problema con los mismos parámetros, es decir que cada ensayo entrega una solución global particular.

estancamiento En ACO, se produce estancamiento cuando el proceso autocatalítico no se controla por ningún mecanismo (como es el caso de AS) y la acumulación de

	feromonas es tal que todas las hormigas siguen el mismo rastro y la exploración se detiene.
feromona	Las feromonas son sustancias químicas secretadas por los seres vivos, con el fin de provocar comportamientos específicos en otros individuos de la misma especie. Son un medio de transmisión de señales que pueden ser tanto volátiles como no volátiles. En particular son utilizadas por las hormigas para marcar rastros que otras hormigas pueden detectar para seguir un camino dado.
grafo	Representación simbólica de los elementos constituidos de un sistema o conjunto, mediante esquemas gráficos.
greedy algorithm	Ver Algoritmo Voraz
Hamiltoniano	Circuito que recorre un conjunto de nodos en su totalidad, de a uno, terminando (cerrando el circuito), en el punto de origen.
heurística	Método de resolución de problemas que no tienen un algoritmo de resolución determinado. En estos casos se intenta encontrar la mejor solución posible si es que no se puede dar con la solución óptima.
IP	Internet Protocol. Protocolo de ruteo correspondiente a la capa tres del modelo OSI, usado en redes TCP-IP tales como Internet.
iteraciones	Cada uno de los procesos de construcción de una solución local. En nuestro TSP, implica la construcción de un recorrido por cada una de las hormigas del esquema, y su comparación final para determinar el mejor del conjunto.
JDBC	Java Database Connectivity, más conocida por sus siglas JDBC,1 es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.
JXL	La API JXL (también conocida como Java Excel API) es la API más utilizada leer, escribir, crear y modificar hojas en una planilla con formato Excel (.xls) en tiempo de ejecución. No es compatible con el formato .xlsx. Puede utilizarse también con LibreOffice Calc con un SO Linux
metaheuristica	Las metaheurísticas se valen de algoritmos heurísticos para la resolución de problemas complejos, agregando una serie de variables estocásticas, que pueden ser ajustadas de manera empírica, buscando la mejor combinación de ellas que se ajuste a un problema específico.
MMAS	Min-Max Ant System. AS mejorado que incorpora limites a los valores de feromona para reducir la posibilidad de estancamiento. Incorpora opcionalmente un método para restablecer estos valores ante una situación de convergencia.

nodo	Punto de control de un grafo. En nuestro caso son las ciudades a recorrer por el viajante.
NP-Hard	Non Polynomial time Hard. Son todos los problemas de decisión que tienen al menos la dificultad de un problema de tiempo polinomial no determinístico
OCH	Optimización por Colonia de Hormigas
OSPF	Open Shortest Path Fast. Protocolo de ruteo dinámico usado en redes IP
Problema del Viajante de Comercio	Consiste en encontrar el menor recorrido posible, dado un conjunto de nodos (ciudades) a visitar, con la condición de visitarlos a todos y volver al punto de inicio al finalizar.
PTS	Pheromone Trail Smoothing. Suavización de los rastros de feromona. Mecanismo de control de los niveles de feromona usado en MMAS que permite aumenta los niveles de los arcos no contenidos en la solución local cuando se sospechan condiciones de convergencia, con el fin de estimular nuevas búsquedas.
RIP	Routing Internet Protocol. Protocolo de ruteo dinámico usado en redes IP, tomado como modelo para desarrollar el mecanismo de armado de las matrices de los pesos (distancias) de los arcos de nuestros grafos.
RIP2	RIP version 2
ruteador	También llamado enrutador, encaminador o router. Nodo de una red de comunicaciones donde reside la lógica de decisión de encaminamiento de la información.
Smooting	Ver PTS
solución global	Mejor solución encontrada en un momento dado de la resolución de un problema.
solución local	Mejor solución de una iteración.
TSP	Ver Problema del Viajante de Comercio
TSP asimétrico	TSP asimétrico. TSP en el cual existen nodos que se conectan entre si con distintas distancias según sea el sentido de circulación elegido.
TSP simétrico	En un TSP simétrico todo par de nodos se comunican entre por un camino bidireccional y de idéntica distancia en un sentido u otro.

10 Bibliografía y referencias

Ant colonies for the traveling salesman problem (Dorigo et al. - Université Libre de Bruxelles, Bélgica)

ACO Algorithms for the Traveling Salesman Problem (Thomas Stützle and Marco Dorigo, Université Libre de Bruxelles, Belgica).

MAX-MIN Ant System. Future Generation Computer Systems (Stützle y H. H. Hoos. 2000)

Learning Bayesian Networks by ant Colony Optimisation: Searching in Two Different Spaces, Mathware & Soft Computing (L.M. de Campos, J.A. Gámez, J.M. Puerta - Departamento de Ciencias de la Computación e I.A., E.T.S. de Ingenieria Informatica, Universidad de Granada, España).

Ant colony optimization for routing and loadbalancing: Survey and new directions. IEEE Trans Syst Man Cybern (Kwang Mong Sim - University of Kent),

Algoritmos de Optimización basados en Colonias de Hormigas aplicados al Problema de Asignación Cuadrática y otros problemas relacionados (Franco Luis Alejandro Arito - Universidad Nacional de San Luis, Facultad de Ciencias Físico Matemáticas y Naturales, Departamento de Informática).

Optimización basada en Colonia de Hormigas: Generalidades y estudio del algoritmo Sistema Hormiga y aplicación a un Job Shop (Andrés Atehorthúa - Universidad Nacional de Colombia).

Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations (Blum C, Sampels M., 2002 Congress on Evolutionary Computation).