ALGORITMOS DE OPTIMIZACIÓN BASADOS EN COLONIAS DE HORMIGAS

Rafael García Fernández
75158784M
rafag@correo.ugr.es
BioInformática 2012-2013
Profesor prácticas: José García Moreno-Torres

1 ÍNDICE

2 DESCRIPCIÓN DEL PROBLEMA DEL VIAJANTE DE COMERCIO (TSP)

El problema del viajante de comercio (TSP) es uno de los problemas más conocidos de optimización combinatoria computacional.

En su descripción más básica se tiene una serie de ciudades y una ciudad en la que se comienza, habitualmente elegida al azar. Se quiere encontrar una ruta con la que se pase por todas las ciudades una sola vez y se vuelva a la inicial, siendo la distancia recorrida por el viajante la menor posible.

La solución más directa para este tipo de problema es la fuerza bruta. Se buscan todas las posibles combinaciones de ciudades, eligiendo la de menor trazado. El problema radica en que el número de posibles combinaciones viene dado por el factorial del número de ciudades (n!), por lo que incluso con un número de ciudades reducido se hace impracticable esta solución con los medios computacionales actuales.

Debido a que los algoritmos clásicos no pueden resolver este problema, técnicas computacionales como heurísticas evolutivas, redes de Hopfield, etc. han sido utilizadas para encontrar soluciones.

El TSP está así entre los problemas denominados NP-completos. Estos son los problemas no resolubles en tiempo polinomial en función del tamaño de la entrada (en el caso del TSP el número de ciudades que el viajero tiene que recorrer). Sin embargo, algunos casos concretos del problema han sido resueltos hasta su optimización, lo que convierte el problema en un excelente laboratorio para algoritmos de optimización que pertenezcan a la misma familia. En este trabajo se va a utilizar el TSP para analizar y comparar varios algoritmos de optimización basados en sistemas de colonias de hormigas.

3 DESCRIPCIÓN DE LOS ALGORITMOS OCH

3.1 EL SISTEMA DE HORMIGAS (SH)

El sistema de hormigas (SH) es el más básico y el algoritmo base del que parten las demás variantes. Así, sus principales características son comunes al resto de los algoritmos, siendo algunas de estas potenciadas para obtener mejores resultados.

Cada hormiga artificial es un mecanismo probabilístico de construcción de soluciones que usa para ello:

- Rastros de feromona artificiales, que representan la experiencia adquirida por las hormigas con el paso del tiempo.
- Información heurística, la distancia entre ciudades en el caso del TSP.

Cada una de las hormigas de la colonia ha de encontrar una ruta solución para el problema. Estando en una ciudad, las hormigas hacen uso de una regla probabilística de transición, la cual define la probabilidad con la que una hormiga situada en una ciudad ha de moverse a otra determinada ciudad, estando está en su mismo vecindario y no habiendo sido visitada ya. Es por esto por lo que hace falta que cada hormiga lleve una lista con las ciudades que ya ha visitado. La regla de transición viene definida por:

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

- τ_{rs} es la feromona del arco a_{rs}
- η_{rs} es la inf. heurística del arco a_{rs}
- $J_k(r)$ es el conjunto de nodos alcanzables desde r no visitados aún por la hormiga k
- α y β son pesos que establecen un equilibrio entre la importancia de la información memorística y heurística

Esta forma de conducir la exploración es también “exploración dirigida”.

Una vez todas las hormigas han encontrado rutas, las mejores rutas son reforzadas aumentándoles su nivel de feromona para encontrar así mejores soluciones en las siguientes iteraciones. De manera análoga, la feromona de todas las rutas se va evaporando con el paso del tiempo. Esto evita el estancamiento en óptimos locales. La regla de actualización de feromona en el sistema de hormigas viene dada por:

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

- $\Delta\tau_{rs}^{k} = \begin{cases} \frac{1}{C(S_k)}, & \text{si la hormiga k ha visitado el arco } a_{rs} \\ 0, & \text{en otro caso} \end{cases}$
- $C(S_k)$ es el coste de la solución generada por la hormiga k, es decir, la longitud del circuito S_k
- m es el número de hormigas

Explicados los puntos básicos del algoritmo, vamos a ver la secuencia de pasos que sigue el algoritmo y algunos detalles del código de este.

1. Inicialización de parámetros, como puede ser la matriz del rastro feromona (pheromone[][]) a un valor inicial de feromona predeterminado (initial_trail). También se genera la matriz total (combinación heurística y matriz de rastro de feromona) como una copia de la matriz de feromona. Esto se hace a través de la función init_pheromone_trails llamada en la función init_try.
   Initial_trail en SH vale: trail_0 = 1. / ((rho) * nn_tour()).
2. Mientras que no se hayan completado todas las iteraciones marcadas o el tiempo de ejecución máximo no haya sido superado, se repiten los siguientes pasos:
   1. Al comienzo de cada iteración cada hormiga es asignada a una ciudad distinta, aunque esto es solo en teoría ya que en la implementación se asignan a una misma ciudad.
   2. Se utiliza la regla de transición vista más arriba (exploración dirigida) para ver a qué ciudad se mueve la hormiga artificial. La función utilizada es neighbour_choose_and_move_to_next, llamada desde la función construct_solution del main. Se decide a que ciudad se mueve de forma probabilística (las a priori mejores rutas son elegidas con mayor probabilidad), dejando fuera las que ya han sido visitadas.
   3. Se procede a la evaporación de la feromona. Para esto se utilizan la función de evaporation llamada dentro de la función pheromone_trail_update del main, la cual reduce cada uno de los rastros de feromona por el factor rho.
   4. Acto después se actualizan las feromonas, teniendo en cuenta las mejores soluciones obtenidas hasta el momento por las hormigas. Esto se hace mediante la función global_update_pheromone llamada desde el método as_update. Dependiendo de lo buena que sea la solución de cada una de las hormigas, se suma a cada arco que forma la ruta una cantidad determinada de feromona (1.0/tour_length).
   5. Se almacena la mejor ruta obtenida y se comprueba si es mejor que las obtenidas en anteriores iteraciones. Si se da esto, se actualiza la mejor ruta.
3. Se devuelve la mejor solución.

3.2 EL SISTEMA DE COLONIAS DE HORMIGAS (SCH)

El sistema de colonia de hormigas (SCH) extiende a su predecesor SH, haciendo hincapié en la mejora de la actualización de la feromona para mejorar su rendimiento. También cambia la manera de seleccionar la siguiente ciudad a la que la hormiga viaja.

La regla de transición nueva establece un equilibrio entre la exploración de nuevos arcos (exploración dirigida, vista en SH) y la explotación de la información acumulada (selección de manera determinística del arco más prometedor). Esta regla es también llamada "regla pseudoaleatoria".

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

- s es la ciudad escogida por la hormiga k en su siguiente movimiento
- q es un uniforme en [0,1]
- $q_0 \in [0,1]$ es la probabilidad con la que se escoge determinísticamente el arco más prometedor (explotación)
- S es una ciudad aleatoria seleccionada según la regla de transición del SH (exploración dirigida)

Respecto a la actualización global de la feromona, esta solo se ve modificada por la hormiga que encuentra la mejor solución en cada una de las iteraciones. La operación que representa ahora la actualización global de la feromona queda así:

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

La mayor novedad del SCH viene dada por la aparición de una actualización local de feromona. Cada hormiga modifica automáticamente la feromona de cada arco por el que pasa. Esto diversifica la búsqueda, evitando óptimos locales, ya que cada vez que un arco es visitado se convierte en menos prometedor, por lo que se exploran otros arcos no visitados. La actualización local de la feromona viene dada por la expresión siguiente:

$$\tau_{rs}(t) = (1-\varphi)\cdot\tau_{rs}(t-1) + \varphi \cdot \tau_0$$

Así, los pasos para el algoritmo SH visto en la sección anterior no difiere mucho de los pasos a hacer para el SCH.

La primera diferencia se encuentra en la inicialización de parámetros, siendo el valor inicial del rastro de feromona (initial_trail) trail_0 = 1. / ((double) n * (double) nn_tour( )).

La regla de transición funciona de la siguiente manera: con una probabilidad de q0 se selecciona el mejor arco de acuerdo a la información heurística y a los rastros de feromona existente. Con probabilidad 1-q0 se hace uso de la exploración dirigida (utilizando probabilidades se elige una ciudad vecina que no haya sido visitada con anterioridad). De esto se ocupa la función neighbour_choose_and_move_to_next.

Al finalizar esto se realiza la actualización local mediante la función local_acs_pheromone_update, la cual se encarga de diversificar la búsqueda eliminando algo de feromona al paso de cada hormiga por la ruta.

Una vez finalizada la iteración, se realiza la actualización global. Esta se consigue a través del método global_acs_pheromone_update llamada desde el método acs_global_update. Se le pasa como parámetro “best_so_far_ant”, para indicar que solo se usa la mejor hormiga hasta el momento para modificar los rastros de feromona

3.3 EL SISTEMA DE HORMIGAS MAX-MIN (SHMM)

El sistema de hormigas Max-Min (SHMM) es una extensión del SH con un mejor balance de exploración-explotación y un mecanismo especial que reduce la posibilidad de estancamiento en la búsqueda.

Este sistema solo introduce cambios en lo referente a la feromona, usando la misma regla de transición que en el SH.

En la actualización global de la feromona solo se considera la mejor solución (sin importar si es de la iteración actual o global). Además, se evapora el resto de rastros existentes.

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

En otros algoritmos OCH no existen límites para los valores de feromona, sin embargo el SHMM introduce un límite mínimo y un límite máximo (estos son calculados de manera heurística). Al contrario de lo que podría pensarse, todos los rastros se inicializan al valor máximo de feromona en lugar de al valor pequeño, consiguiendo así que al aplicar la regla de actualización los arcos buenos mantienen sus valores mientras que los malos los pierden. Con esto también se consigue una mayor exploración de rutas al comienzo de la ejecución del algoritmo.

Para terminar, cuando se detecta que la búsqueda se ha estancado, se reinicia la búsqueda poniendo todos los rastros de feromona al valor máximo.

La combinación de estas mejoras convierte al SHMM en el mejor algoritmo OCH a día de hoy.

Se procede ahora a ver como se traducen estos cambios en la implementación. En la inicialización de parámetros, se inicializan los valores máximos y mínimos de feromona usando heurísticas y se usa el máximo como parámetro initial_trail para la función init_pheromone_trails.

```
trail_max = 1. / ( (rho) * nn_tour() );

trail_min = trail_max / ( 2. * n );
```

La construcción de soluciones no cambia con respecto al SH y al SCH (no hay actualización local de los rastros de feromona como hay en este último).

La evaporación de los rastros se hace mediante la función evaporation como en el resto de algoritmos, evaporando todos los rastros existentes por el factor rho.

```
pheromone[i][j] = (1 - rho) * pheromone[i][j];
```

La actualización global de la feromona se hace haciendo uso de la función mmas_update, que simplemente busca la mejor solución hasta el momento y se la pasa como parámetro a la función global_update_pheromone. También se comprueba si la búsqueda está estancada y de ser así se reinicializa la búsqueda.

Dado que existen límites en los valores de feromona de SHMM, se hace uso de la función check_pheromone_trail_limits para ver si estos se cumplen. Esto no se haría si se usase búsqueda local.

3.4 EL SISTEMA DE HORMIGAS MEJOR-PEOR (SHMP)

El sistema de hormigas mejor-peor (SHMP) se parece bastante al SHMM. Es una extensión del SH basada en la incorporación de componentes de computación evolutiva para mejorar el equilibrio intensificación-diversificación.

Al igual que el SHMM mantiene la regla de transición del SH, además de evaporar todos los rastros en el mecanismo de actualización, aportando sólo en el de la mejor solución (ya sea de la iteración actual o global). Se definen también unos topes máximos y mínimos de valores de feromona y se reinicializa la búsqueda cuando se produce estancamiento. La expresión de la actualización viene dada por:

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

Una de las principales novedades se debe a que además de aportar feromona en la mejor solución global, penaliza (evaporación extra) los rastros de la peor hormiga de la iteración actual que no estén en contenidos en la mejor global:

$$\tau_{rs}(t) \leftarrow (1 - \rho)\cdot\tau_{rs}(t), \quad \forall\, a_{rs} \in S_{peor-actual} \quad y \quad a_{rs} \notin S_{mejor-global}$$

Este refuerzo negativo del rastro de la peor hormiga hace que la regla de actualización tenga un comportamiento más intensificativo que otros algoritmos de hormigas.

El SHMP da la búsqueda por estancada cuando pasan un número consecutivo de iteraciones (porcentaje del total) sin que se mejore la mejor solución global obtenida. Como consecuencia de esto, se aplica la reinicialización de la búsqueda, poniendo todos los rastros de feromona al valor inicial.

Un aspecto totalmente nuevo del SHMP es la mutación de los valores de los rastros de feromona para conseguir diversidad en el proceso de búsqueda. La mutación se aplica en cada rastro con una probabilidad determinada, añadiéndosele a la ruta un valor normal de media 0 entre un umbral mínimo y uno máximo. El umbral máximo viene dado por la media de los rastros de la feromona de la mejor solución hasta el momento.

$$\tau'_{rs}(t) = \tau_{rs}(t) + N(0,\tau_{umbral}) \quad ; \quad \tau_{umbral} = \frac{\sum_{a_{rs} \in S_{mejor-global}} \tau_{rs}(t)}{n}$$

La función de mutación se caracteriza por el aumento de la fuerza de mutación con el paso de las iteraciones. Al principio, el umbral máximo de la mutación (media de los rastros de la mejor solución) es cercano al valor inicial de feromona, pero con el paso de las iteraciones se va haciendo más grande. Cuando la búsqueda se estanca y hay que reinicializar, vuelve a su rango inicial de fuerza.

Se procede ahora a ver como se traduce esto en el código del algoritmo.

La inicialización se hace como en el resto de los algoritmos, haciendo uso de la función init_pheromone_trails con el valor inicial de rastro de trail_0 = 1. / ((double) n * (double) nn_tour()).

La construcción de las soluciones no cambia con respecto al SH o SHMM, haciéndose de la misma manera mediante el método neighbour_choose_and_move_to_next.

Después de la utilización de evaporation para evaporar los rastros, se hace la actualización de feromona haciendo uso de bwas_update. Lo primero que se hace es llamar invocar la función global_update_pheromone con la mejor hormiga como parámetro, para aportar feromona a las rutas de la mejor solución global hasta el momento. Para implementar una de las novedades de este algoritmo, la penalización de la peor solución, se hace uso de la función bwas_worst_ant_update, pasándole como parámetro la peor y la mejor hormiga hasta el momento.

Una vez hecho esto se comprueba si la búsqueda está estancada (distance_between_ants) y si resulta estarlo, se reinicializa el proceso inicializando el rastro a su valor inicial. Si no, se procede a realizar la mutación de los rastros llamando al método bwas_pheromone_mutation, modificando el valor de feromona de las rutas con una probabilidad determinada.

3.5 BÚSQUEDA LOCAL EN ALGORITMOS DE OCH

Los algoritmos de OCH son habitualmente utilizados junto a técnicas de búsqueda local para mejorar su eficacia. La hibridación consiste en aplicar una búsqueda local sobre las soluciones construidas por todas las hormigas en cada iteración antes de proceder a la actualización de feromona.

Al usar búsqueda local se obtienen buenos resultados con muchas menos hormigas que en los algoritmos de OCH sin búsqueda local. El aumento en la eficacia reduce la eficiencia. Por ello, se utilizan junto a la búsqueda local las llamadas "Listas de candidatos", consistentes en estudiar sólo las ciudades más prometedoras en cada paso de la hormiga.

Las búsquedas locales que se suelen utilizar en TSP son la 2-opt y la 3-opt. Estas intercambian entre 4 y 8 arcos respectivamente. En este trabajo se ha utilizado 3-opt.

Así, el algoritmo básico de OCH con búsqueda local en pseudocódigo es el siguiente:

Mientras (¡condicion de parada){

-Construcción probabilística de las soluciones

-Refinamiento de dichas soluciones mediante la búsqueda local

-Actualización global de feromona

}

Debido a que la búsqueda local se realiza de la misma forma para todos los algoritmos, excepto un par de particularidades que se dan en la versión del SHMM, se procede a explicarse de manera general sin entrar en cada algoritmo de manera individual.

La llamada para realizar la búsqueda local se realiza mediante la función local_search () si el flag ls_flag > 0 y después de haberse realizado la construcción de soluciones (antes de la actualización de estadísticas o actualización de feromona).

La función local_search tiene un for que hace que cada una de las hormigas realice búsqueda local. Se utiliza un switch y varios cases para ver qué tipo de búsqueda local se realiza (2, 2.5 o 3). En este trabajo solo tendremos en cuenta el 3-opt, del que se ocupa la llamada three_opt_first (ant[k].tour). Una vez esta función termina se actualiza el tour_length de cada hormiga con la función compute_tour_lenght(ant[k].tour).

Los algoritmos de búsqueda local están implementados en los archivos ls.h y ls.c. En la documentación de la función three_opt_first el creador indica que es una implementación mala del three_opt_first, por lo que no considero que sea necesario entrar en detalles sobre cómo minimiza la longitud de la ruta.

Un cambio importante en la implementación de los algoritmos al utilizar búsqueda local se produce en la actualización de feromona. A diferencia de los algoritmos básicos, en los que se evaporaban todos los rastros, solo se evaporan las feromonas de los arcos pertenecientes a la lista de candidatos con la ciudad actual para hacer la evaporación de la feromona más rápida, de manera que es capaz de abordar instancias de TSP más grandes. Esto se hace mediante la función evaporation_nn_list, excepto en el caso del SHMM que se hace con mmas_evaporation_nn_list(). Esta diferencia se debe a que esta función comprueba los límites inferiores de feromona, por lo que no es necesario hacer uso de la función

check_pheromone_trail_limits que se encarga de esta comprobación en la versión del algoritmo sin búsqueda local.

Además, en el SHMM con búsqueda local se da otra particularidad a la hora de actualizar la feromona (mmas_update). Se realiza la programación para u_gb definido en el artículo “Future Generation Computer Systems Article” de Stuetzle [Stuetzle2000].

4 EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

4.1 INSTANCIAS DEL PROBLEMA ESTUDIADAS

Los algoritmos anteriormente descritos han sido probados haciendo uso de 6 instancias del problema TSP. Éstas han sido obtenidas desde la biblioteca TSPLIB y corresponden a TSP simétricos. Son las siguientes:

- Eil76: 76 ciudades. Coste de la solución óptima: 538
- KroA100: 100 ciudades. Coste de la solución óptima: 21282
- D198: 198 ciudades. Coste de la solución óptima: 15780
- Lin318: 318 ciudades. Coste de la solución óptima: 42029
- Att532: 532 ciudades. Coste de la solución óptima: 27686
- Rat783: 783 ciudades. Coste de la solución óptima: 8806

4.2 VALORES DE LOS PARÁMETROS EN LOS ALGORITMOS

Una correcta elección de los valores de los parámetros seleccionados a la hora de ejecutar los algoritmos es crucial para la obtención de buenos resultados. Se procede a ver los distintos parámetros y sus correspondientes valores con los que se ejecutaran los algoritmos. A no ser que se diga lo contrario el parámetro se ejecutará con todos los algoritmos.

- Número de ejecuciones a realizar: 5. Dado que los algoritmos son de carácter probabilístico, se hace necesaria la ejecución de éste varias veces para evitar que el resultado final no pueda estar sesgado (ya sea positivamente o negativamente) por un caso particular, mostrando así un comportamiento distinto al que normalmente se obtendrá.
- Número de hormigas: 15
- Tiempo de ejecución/Criterio de parada: este depende de la instancia del TSP sobre la que se ejecuta el problema. Para las instancias eil76 y kroA100 es de 300 segundos, d198 y lin318 600 segundos y att532 y rat783 900 segundos. Otro criterio de parada que se utiliza es el parar si alguna hormiga encuentra un camino de la misma longitud que el coste de la solución óptima de la instancia.
- Valor de los parámetros de la regla de transición: alfa = 1 (heurística) y beta = 2 (feromona).
- Valor del parámetro de evaporación: 0.2. Utilizado para la actualización de la feromona después de cada iteración.
- Valor óptimo: depende de la instancia del TSP. Usado para realizar estadísticas y como criterio de parada si se obtiene una solución inmejorable.
- Número de vecinos del vecindario: 40. Utilizado sólo con la búsqueda local activada.
- Valor de q0 en la regla de transición: 0.8. Sólo para SCH y SCH con BL.
- Valor parámetro evaporación en la actualización local de feromona: 0.2. Solo para SCH y SCH con BL.
- Probabilidad de mutación: 0.3. Sólo para SHMP.
- Iteraciones sin mejora: 0.3. Sólo para SHMP.

4.3 RESULTADOS OBTENIDOS

Los algoritmos han sido probados sobre un portátil Intel Core i5-2410 de 64 bits, con una velocidad de procesador de 2.30 GHz y 4 GB de RAM. Sistema operativo Ubuntu 9.04 lanzado desde una máquina virtual (VMWare) corriendo en Windows 8.

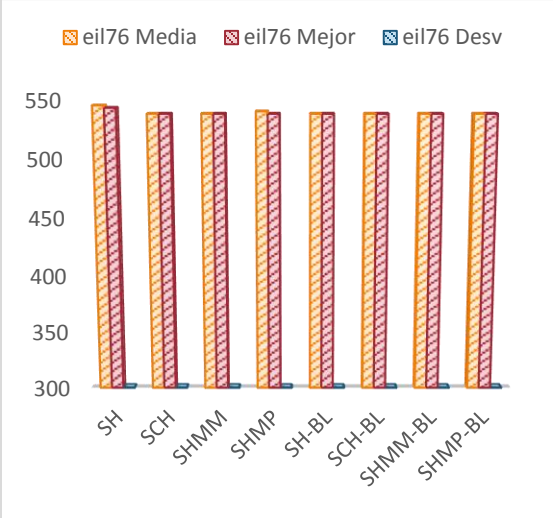
Las tablas con los resultados de cada una de las ejecuciones de los algoritmos pueden verse en el apéndice A.

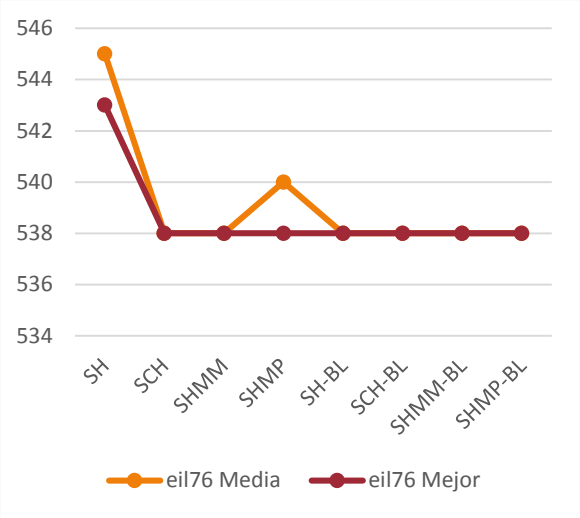
Modelo	eil76			kroA100			D198		
	538			***21282***			***15780***		
	Media	**Mejor**	**Desv**	**Media**	**Mejor**	**Desv**	**Media**	**Mejor**	**Desv**
SH	545	543	2,49	22174	22045	97,94	16949	16780	97,9
SCH	538	538	0	21309	21282	44,2	16122	15914	141,41
SHMM	538	538	0	21282	21282	0	15931	15924	5,35
SHMP	540	538	2,28	21409	21282	146,82	16144	15968	88,95
SH-BL	538	538	0	21282	21282	0	15780	15780	0
SCH-BL	538	538	0	21282	21282	0	15780	15780	0
SHMM-BL	538	538	0	21282	21282	0	15780	15780	0
SHMP-BL	538	538	0	21282	21282	0	15780	15780	0

Modelo	lin318			att532			rat783		
	42029			***27686***			***8806***		
	Media	**Mejor**	**Desv**	**Media**	**Mejor**	**Desv**	**Media**	**Mejor**	**Desv**
SH	46972	46491	244	33648	33145	284	11278	11031	141,02
SCH	42629	42155	242,11	28312	28137	201,58	9125	9023	63,6
SHMM	42727	42641	80,01	28244	28155	80,22	9039	9005	17,83
SHMP	42776	42500	250,54	28681	28434	161,78	9400	9342	42,01
SH-BL	42087	42082	13,74	33742	33510	124,57	8938	8924	10,34
SCH-BL	42029	42029	0	27689	27686	7.6	8808	8006	3.2
SHMM-BL	42029	42029	0	27686	27686	0	8806	8806	0
SHMP-BL	42029	42029	0	27686	27686	0	8817	8806	11,97

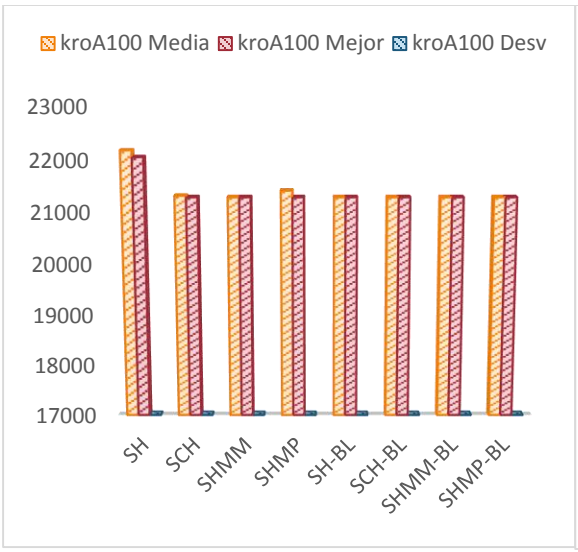
Se procede a analizar los resultados de cada una de las instancias TSP en términos de mejor resultado, mejor resultado medio obtenido y robustez.

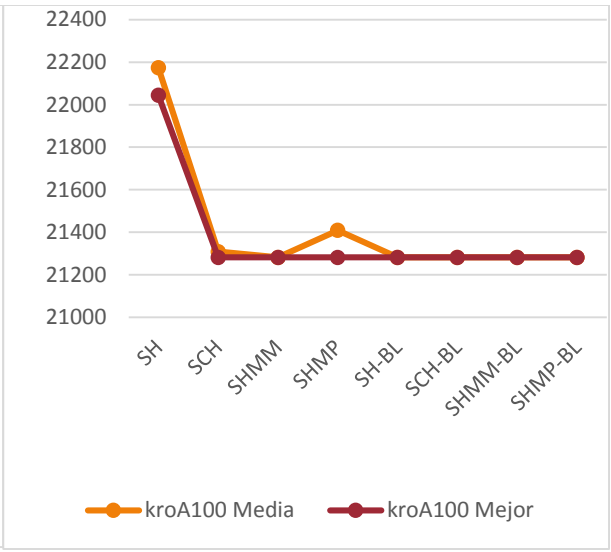
- Eil76: los 8 algoritmos se comportan bastante bien, logrando 6 de ellos la longitud óptima como media de las ejecuciones. Solo el SHMP y el básico SH no lo logran, aunque igualmente consiguen resultados muy cercanos al óptimo. Cuando se aplica la búsqueda local a estas versiones del sistema de hormigas logran los resultados óptimos sin problema. Como se irá viendo a continuación, esta será la tónica general.



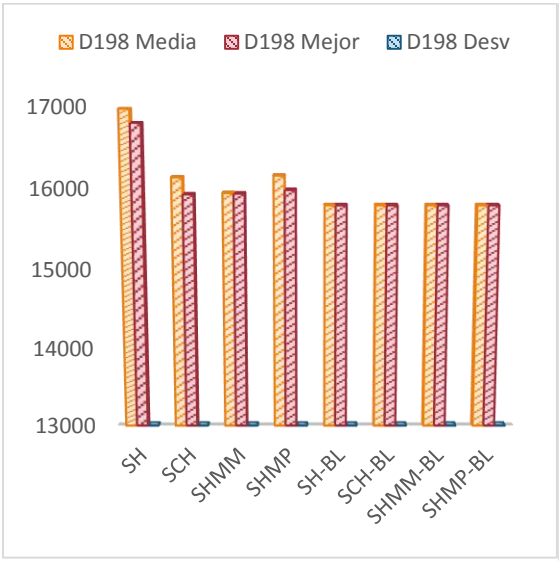


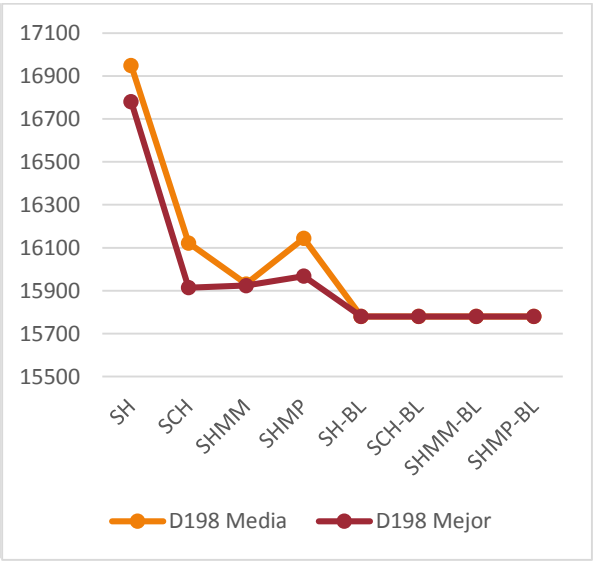
- kroA100: el algoritmo con peor comportamiento de largo es el SH, obteniendo resultados muy lejos del óptimo tanto en media como en mejor resultado. Su desviación típica es de 97,94 lo que nos dice que tampoco consigue resultados especialmente robustos. El resto de algoritmos consiguen en alguna iteración la longitud óptima y todos excepto el SCH y el SCMP lo consiguen en media. Entre estos 2 es preferible la utilización del SCH, ya que además de tener una media bastante más cercana a la óptima, su desviación típica representa una gran diferencia en lo que a robustez se refiere (44,2 por 146,82 del SCMP). De nuevo las versiones con búsqueda local no tienen problema alguno para obtener la longitud óptima en media.



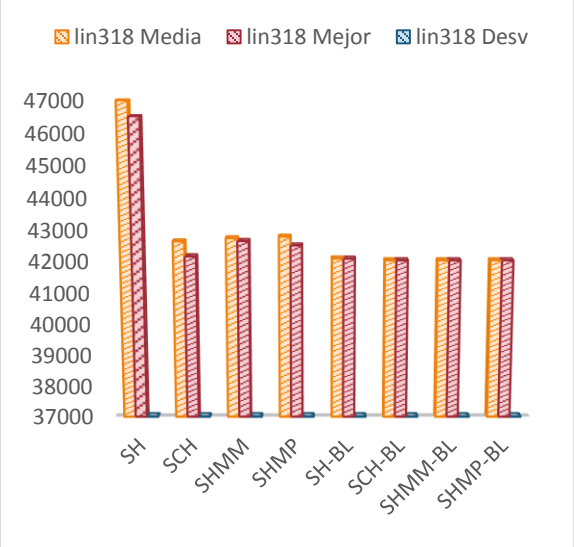


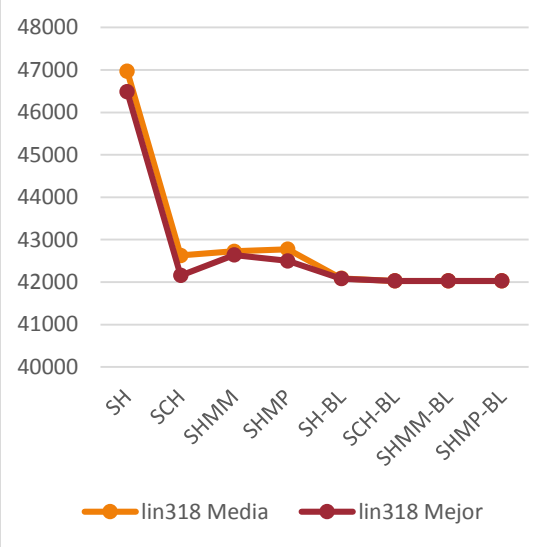
- D198: en esta instancia ninguno de los algoritmos sin búsqueda local se queda cerca de los óptimos tanto en mejor ejecución como en media. El SH empieza a mostrar sus carencias de manera clara, quedándose a una distancia 1000 en ambos aspectos y con una robustez muy mejorable (97 de desviación típica). Entre los otros 3 algoritmos sin búsqueda local se obtienen resultados muy parecidos en mejor y resultado medio (algo mejores en SHMM), aunque en robustez si se aprecian aspectos interesantes. El SHMM presenta una robustez enorme (desviación 5,35) mientras que SHMP (88,95) y sobretodo SCH (141,41) carecen de esta.
  Las variantes de los algoritmos con la búsqueda local activada vuelven a resolver el problema con coste óptimo de media.



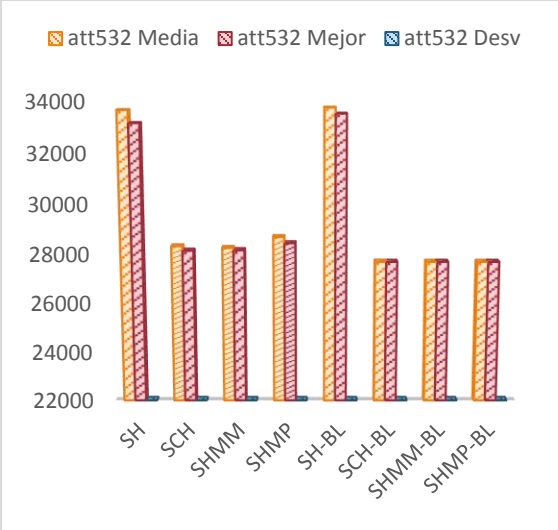


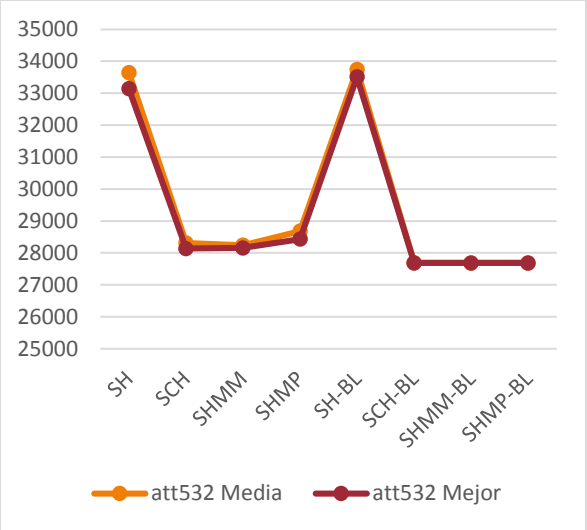
- Lin318: en esta instancia del problema se hace más evidente aún si cabe las carencias de SH ya sea en coste obtenido en la mejor ejecución, media o robustez. Conforme avanzamos en las instancias esta diferencia se va haciendo más grande, lógico por otra parte ya que SH es el algoritmo más básico de los vistos y del que parten los demás.
  Centrándonos en los algoritmos sin búsqueda local, se aprecia de nuevo una gran igualdad entre SCH, SHMM y SHMP, siendo los de SCH ligeramente mejores tanto en media como en mejor a los otros 2 algoritmos. De hecho, SHMP mejora también en mejor ejecución a SHMM (no así en media). Sin embargo, en esta instancia los algoritmos muestran un carácter muy poco robusto, (desviación 242 SCH y 250 SHMP) excepto el SHMM que es el único que consigue unos resultados robustos en cierta medida (desviación 80).
  Las variantes de los algoritmos con búsqueda local mejoran muchísimo de nuevo los resultados, siendo el SH con búsqueda local el único que no consigue la media óptima (nueva muestra de la debilidad de SH), quedándose muy cerca igualmente.



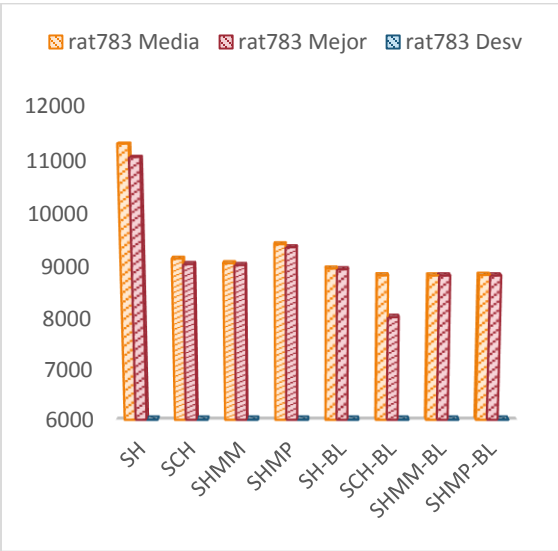


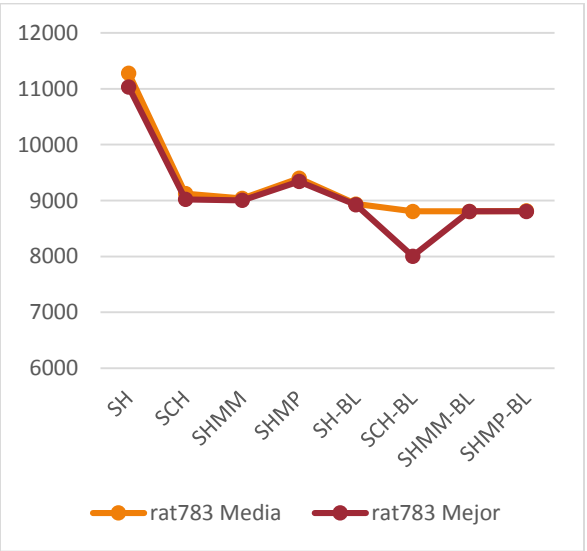
- Att532: muy similar a los resultados obtenidos en la instancia TSP anterior. Máxima igualdad entre los algoritmos SCH, SHMM y SHMP, siendo la robustez del segundo el factor que verdaderamente lo hace mejor (80 contra 201 y 161).
  El algoritmo SH con búsqueda local activada no da buenos resultados, siendo el primer caso en el que vemos que la búsqueda local por sí sola no es suficiente para resolver de manera relativamente óptima el problema. SCH-BL se queda muy cerca de los resultados óptimos, que solo son alcanzados con SHMM-BL y SHMP-BL.





- Rat783: resultados muy parecidos a los de las últimas instancias analizados de TSP. SHMM da los mejores resultados, no sólo en robustez sino en este caso también media y en mejor ejecución. Con las búsquedas locales activadas, el SHMM-BL es el único capaz de resolver el problema de manera óptima, siguiéndole el SCH de cerca.





4.4 CONCLUSIONES

Como se aprecia, con el aumento de complejidad de las instancias del TSP todos los algoritmos se vuelven menos competitivos. El que da mejores resultados en todas las instancias del problema es el SHMM, destacando no solo por sus mejores resultados y por la media de estos, sino también por su gran robustez, algo que el resto de algoritmos no posee.

Probablemente la mayor conclusión que podemos ver es lo beneficioso que resulta activar la búsqueda local para cada uno de los algoritmos, ya que con esta activada se consiguen resultados muy cercanos a los óptimos con la mayoría de los algoritmos. Incluso el SH, que como hemos podido ver tiene bastantes debilidades con problemas un poco complejos, consigue ser competitivo con la búsqueda local activada. Esto se debe a que los algoritmos con búsqueda local activada salen de máximos locales (falsas soluciones óptimas por decirlo de otra forma) de manera mucho más fácil que los originales, que se quedan estancados en soluciones locales con relativa facilidad.

5 APÉNDICE A. TABLAS

Algoritmo SH	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	550	22296	17031	47049	33632	11031
Ejecución 2	544	22104	16780	47139	33694	11429
Ejecución 3	543	22147	17041	47135	34018	11371
Ejecución 4	546	22278	16901	47050	33145	11222
Ejecución 5	544	22045	16993	46491	33753	11340
Media	545	22174	16949,2	46972,8	33648,4	11278,6
Desv. Típica	2,49	97,94	97,9	244	284	141,02

Algoritmo SH con BL	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	15780	42091	33872	8944
Ejecución 2	538	21282	15780	42071	33742	8924
Ejecución 3	538	21282	15780	42112	33820	8930
Ejecución 4	538	21282	15780	42082	33770	8953
Ejecución 5	538	21282	15780	42082	33510	8942
Media	538	21282	15780	42087,6	33742,8	8938,6
Desv. Típica	0	0	0	13,74	124,57	10,34

Algoritmo SCH	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	16001	42791	28188	9184
Ejecución 2	538	21396	15914	42155	28278	9139
Ejecución 3	538	21282	16294	42813	28703	9193
Ejecución 4	538	21305	16204	42694	28256	9023
Ejecución 5	538	21282	16198	42693	28137	9086
Media	538	21309,4	16122,2	42629,2	28312,4	9125
Desv. Típica	0	44,2	141,41	242,11	201,58	63,6

Algoritmo SCH con BL	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	15780	42029	27686	8814
Ejecución 2	538	21282	15780	42029	27686	8806
Ejecución 3	538	21282	15780	42029	27705	8806
Ejecución 4	538	21282	15780	42029	27686	8806
Ejecución 5	538	21282	15780	42029	27686	8806
Media	538	21282	15780	42029	27689,8	8807,6
Desv. Típica	0	0	0	0	7,6	3,2

Algoritmo SHMM	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	15934	42736	28308	9045
Ejecución 2	538	21282	15924	42649	28229	9005
Ejecución 3	538	21282	15930	42641	28155	9046
Ejecución 4	538	21282	15929	42861	28168	9057
Ejecución 5	538	21282	15940	42749	28363	9044
Media	538	21282	15931,4	42727,2	28244,6	9039,4
Desv. Típica	0	0	5,35	80,01	80,22	17,83

Algoritmo SHMM con BL	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	15780	42029	27686	8806
Ejecución 2	538	21282	15780	42029	27686	8806
Ejecución 3	538	21282	15780	42029	27686	8806
Ejecución 4	538	21282	15780	42029	27686	8806
Ejecución 5	538	21282	15780	42029	27686	8806
Media	538	21282	15780	42029	27686	8806
Desv. Típica	0	0	0	0	0	0

Algoritmo SHMP	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21600	16124	42867	28830	9366
Ejecución 2	541	21282	15968	42693	28598	9410
Ejecución 3	544	21577	16160	42606	28662	9461
Ejecución 4	538	21305	16237	43216	28882	9342
Ejecución 5	539	21282	16084	42500	28434	9422
Media	540	21409,2	16114,6	42776,4	28681,2	9400,2
Desv. Típica	2,28	146,82	88,95	250,54	161,78	42,01

Algoritmo SHMP con BL	Eil76	kroA100	D198	Lin318	Att532	Rat783
	Coste	Coste	Coste	Coste	Coste	Coste
Optimo	538	21282	15780	42029	27686	8806
Ejecución 1	538	21282	15780	42029	27686	8806
Ejecución 2	538	21282	15780	42029	27686	8811
Ejecución 3	538	21282	15780	42029	27686	8812
Ejecución 4	538	21282	15780	42029	27686	8840
Ejecución 5	538	21282	15780	42029	27686	8820
Media	538	21282	15780	42029	27686	8817,8
Desv. Típica	0	0	0	0	0	11,97

6 BIBLIOGRAFÍA

[Stuetzle2000] Max-Min Ant System