Implementación de tres programas basados en algoritmos ACO

Alberto Cuesta Cañada

ITESM, Campus Monterrey

alberto@cuesta.be

791800

1. Resumen

Recientemente, un número de algoritmos inspirados por el comportamiento colectivo de colonias de hormigas ha sido aplicado con éxito a dificiles problemas discretos de optimización. En el presente documento se presenta el aprendizaje del autor en la materia a través de la resolución de tres problemas con algoritmos ACO.

El presente documento no debe ser considerado como un estudio minucioso, puesto que la mayor parte de las actividades no fueron documentadas, siendo como era un trabajo experimental, dinámico y no llamado a revelar nada que no esté ya presente en la literatura sobre el tema. Un documento más elaborado hubiera requerido mucho más tiempo y esfuerzo, el presente solo debe ser considerado como una explicación general a los programas implementados.

2. Ants.py

Como primera implementación de un algoritmo ACO se escogió simular la busqueda y recuperación de alimento por parte de hormigas. En el programa ants.py un numero predeterminado de hormigas se mueven al azar por un espacio en dos dimensiones. Su único conocimiento acerca del entorno es la dirección y distancia hacia el hormiguero, la cantidad de feromonas en su entorno más inmediato y si se encuentran sobre una fuente de comida. Con esta limitada información las hormigas son capaces de explorar todo el escenario y recoger toda la comida actuando del siguiente modo.

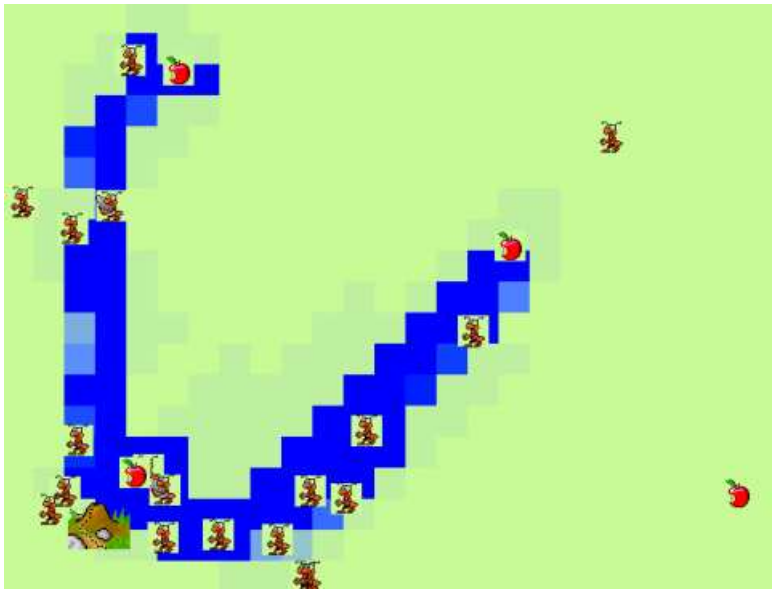
En ausencia de estímulos, la hormiga se mueve al azar.

En caso de detectar feromonas en su alrededor, intentará seguirlas en dirección contraria al nido, si tiene que decidir entre dos direcciones escogerá aquella cuyo contenido de feromonas sea más fuerte.

En caso de encontrar una fuente de comida: cargará una unidad de comida y regresará en linea lo más recta posible al hormiguero, dejando un rastro de feromonas.

En cualquier momento, la hormiga puede decidir realizar un movimiento al azar, aunque ésto provoca que cierto número de ellas pierdan el rastro que estuvieran siguiendo, garantiza que siempre haya un número de hormigas que se dedican a explorar en lugar de explotar las fuentes de comida conocidas.

Las feromonas se debilitan con el tiempo, sin llegar nunca a desaparecer del todo. Que se debiliten garantiza que las rutas más convenientes sean las utilizadas, mientras que el que nunca desaparezcan del todo previene que se pierda la localización de fuentes de comida por falta de uso.



ants.py en funcionamiento

3. TSP.py

El problema del viajante de comercio (TSP) puede ser indefinido informalmente como sigue: Dado un grafo hallar el camino de coste mínimo que pase por todos los nodos sin repetir ninguno.

El programa creado comienza por generar un grafo con forma de malla 2D, el peso de cada arista se determina al azar. A continuación se genera un número de hormigas determinado, que son colocadas en nodos aleatorios del grafo, el comportamiento de las hormigas se define como sigue:

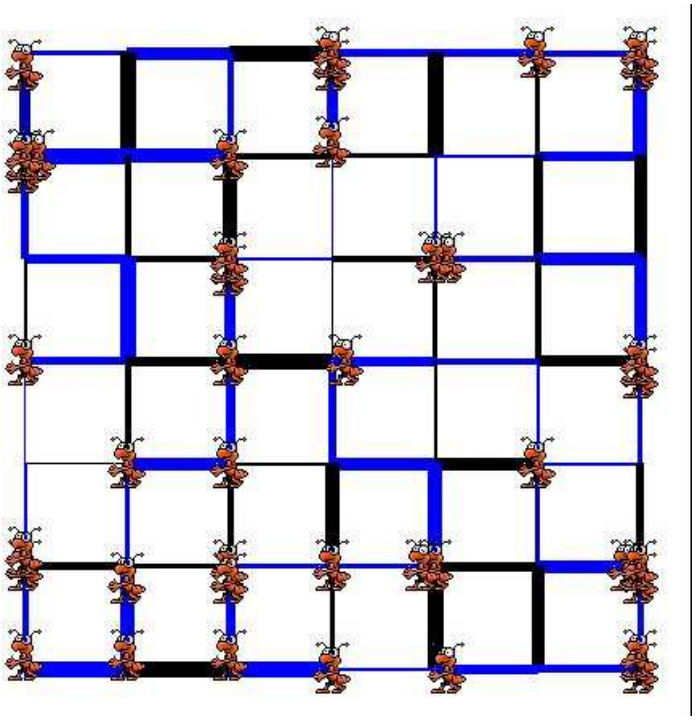
Cada hormiga guarda qué nodos ha visitado en su busqueda actual de una solución.

Cada nodo adyacente a una hormiga tiene una posibilidad $P_{ij} = \frac{Ph_{ij}^{\alpha}+C_{ij}^{\beta}}{\sum Ph_{ij}^{\alpha}+C_{ij}^{\beta}}$ de ser visitado por la misma, donde Ph_{ij} es su contenido de feromonas y C_{ij} es el coste asociado por desplazarse a ese nodo. $\alpha\, y\, \beta$ son variables para modificar el peso de cada variable. La hormiga guarda en memoria que nodos ha visitado, para evitar visitar alguno más de una vez. En caso de que los cuatro nodos adyacentes hayan sido visitados caben dos posibilidades.

-Que aún queden nodos sin visitar: en tal caso la hormiga cancela el camino que estuviera siguiendo y vuelve a empezar (Nota: Tal vez sería más eficiente borrar nodos del principio del camino hasta liberar uno de los cuatro nodos, creando algo similar al juego de la serpiente de los populares teléfonos Nokia)

-Que no queden nodos por visitar: en tal caso la hormiga ha encontrado un camino válido y pasa a aplicar en cada arista una cantidad de feromona según la fórmula $\cos(C_p^{\delta} * \pi/2)$, donde C_p es el coste total del camino hallado y δ es un modificador. Ésta fórmula se reveló bastante ineficiente, aparte de imponer limitaciones a la hora de buscar optimizaciones.

Pese a todas las ineficiencias, el programa funcionó, después de un periodo de busqueda inicial, a partir del hallazgo de la primera solución, se puede ver como las hormigas comienzan a establecer variaciones de la solución, mejorandola progresivamente.



tsp.py

El algoritmo se mostró ineficiente son redes de mas de 50 nodos, si bien tal vez se podría aumentar su rango de actuación mediante heurísticas que proporcionen una rápida solución inicial, una correcta administración del refuerzo y debilitamiento de las feromonas e incorporando mecanismos de busqueda local y soluciones parciales.

4. QAP.py

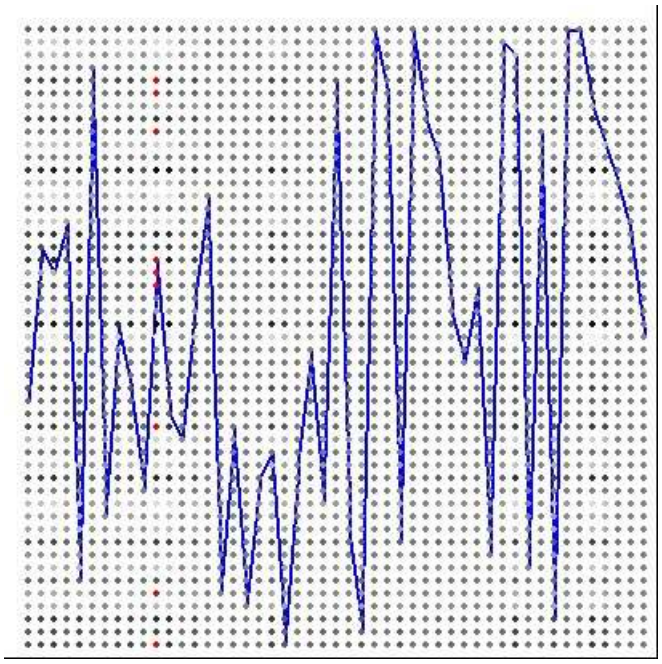
El Problema del Asignamiento Cuadrático se puede entender como el problema de asignar de modo óptimo N tareas a N trabajadores, donde el beneficio de la combinación se obtiene multiplicando la eficacia del trabajador por el beneficio bruto de la tarea.

Para la implementación de este algoritmo mediante ACO se buscó el modo de representarlo como la busqueda del camino óptimo en un grafo. Esto se consiguió creando una matriz de $N^2 \times N^2$, correspondiendo el elemento i,j al beneficio obtenido al asignar la tarea i al trabajador j. Así, el problema se reducia en encontrar un camino que pasase una vez por cada fila y una vez por cada columna, de longitud N.

La programación de este problema supuso un paso más de abstracción desde las hormigas de la naturaleza hasta las de la algorítmica, el problema se reveló intratable si fuera considerado del mismo modo asíncrono con el que se trató el TSP. Así se escogió que todas las hormigas avanzasen al mismo ritmo por la matriz, usando la misma matriz de probabilidades para escoger un camino u otro, cuando las hormigas llegan al final se borran las feromonas que existieran y se vuelven a imprimir según los resultados de esta última iteración.

En este caso se aplicaron algunas estrategias más, por ejemplo solo los caminos que no fueran peores que el peor camino anterior serían reforzados con feromonas.

Una vez más, el principal problema se reveló el coste computacional, algo superior a N^2, así como el problema de *stagnation,* que con frecuencia bloqueaba el programa en una solución subóptima. Existen soluciones que merecen ser investigadas para ambos problemas, pero quedan pendientes para un estudio futuro.



qap.py en funcionamiento

5. Conclusiones

En el presente estudio se analizó la implementación de tres problemas básicos mediante algoritmos basados en colonias de hormigas. Las particularidades de éstos algoritmos que fueron aprendidas comprenden:

Que básicamente un algoritmo ACO es capaz de encontrar un camino de coste mínimo (máximo) en un grafo, teniendo en cuenta detalles particulares si fuera necesario.

Que el modo de depositar las feromonas en el grafo dado es importante, e interesante tener en cuenta detalles como el refuerzo solo de determinadas soluciones, o el ritmo de degradación de las feromonas, o la cantidad máxima de feromona que puede existir en un punto.

Que el problema de estancamiento (stagnation) debe ser tenido en cuenta, posiblemente mediante puestas a cero del problema recordando la solución encontrada.

Que el coste cmputacional de los algoritmos ACO respecto a la talla del problema en un principio no es lineal, sino más bien cuadrático, y que sería muy interesante el hallar modos de aplicar estos algoritmos a problemas de gran talla.