

Simulación de comportamiento de una colonia de hormigas

Luis Enrique Corredera de Colsa

Doctorando en Ingeniería del software

Universidad Pontificia de Salamanca en Madrid.

luisenrique@flagsolutions.net

1 Tabla de contenidos

1	Tabla de contenidos	1
2	Introducción.....	2
3	El comportamiento de una hormiga	3
4	La hormiga artificial	4
5	Implementación del modelo	6
6	Resultado de la ejecución	11
7	Conclusiones.....	13
8	Bibliografía	13

2 Introducción

En los últimos años se vienen estudiando y aplicando métodos de optimización basados en colonias de hormigas para tratar de dar solución a problemas de optimización combinatoria complejos en campos tan diversos como son la economía, el comercio, la ingeniería, la medicina o la industria. Para este tipo de problemas, no existen algoritmos exactos que proporcionen una solución en un tiempo polinomial.

Dentro de los métodos para afrontar este tipo de problemas, tenemos los métodos exactos y los aproximados. Los algoritmos exactos intentan encontrar una solución óptima y luego demostrar que es un óptimo global. Dentro de este tipo de algoritmos encontramos procesos de vuelta atrás (backtracking), procesos de ramificación y poda (Branch and bound), programación dinámica, etc. Dado el bajo rendimiento que proporcionan los algoritmos exactos, se desarrollaron algoritmos aproximados, que se pueden clasificar en dos tipos: constructivos (que generan soluciones desde cero añadiendo nuevas construcciones a la solución, paso a paso) y algoritmos de búsqueda local (que pretenden de forma iterativa encontrar soluciones mejores migrando a soluciones vecinas (esperando que sean mejores)).

Por desgracia, los algoritmos de búsqueda local tienen mucha facilidad para estancarse en óptimos locales, que globalmente no son buenas soluciones. Por ello, investigadores en las últimas dos décadas han dedicado esfuerzos a desarrollar algoritmos que mejoren la efectividad de los métodos de búsqueda local basándose en la heurística para la construcción de soluciones (metaheurísticas).

Una metaheurística relativamente reciente es la optimización basada en colonias de hormigas, la cual se rige por el comportamiento que presentan las

hormigas a la hora de encontrar el camino más corto entre el hormiguero y la comida.

Si entrar en más detalle sobre la optimización basada en colonias de hormigas y los diferentes algoritmos empleados para los diferentes problemas, en este documento se va a presentar un modelado de una colonia de hormigas, con la finalidad de reproducir el comportamiento del que hemos aprendido para el desarrollo de la metaheurística OCH: encontrar el camino más corto entre una fuente de comida y el hormiguero.

3 El comportamiento de una hormiga

Las hormigas son insectos sociales que viven en colonias y que gracias a su mutua colaboración son capaces de mostrar comportamientos complejos y realizar tareas complicadas desde la óptica de una hormiga individual.

Un comportamiento característico de las colonias de hormigas es su capacidad de encontrar los caminos más cortos entre las fuentes de comida y el hormiguero. Este hecho no sería muy interesante si no tenemos en cuenta que la mayoría de las especies de hormigas son prácticamente ciegas, y no pueden apoyarse en pistas visuales para encontrar los caminos.

Algunas especies de hormigas depositan una sustancia química (feromona) mientras recorren el camino entre el hormiguero y la fuente de alimento. Si una hormiga no encuentra rastro de feromona a su paso, deambula al azar. Si existe un rastro de feromona, sigue el rastro como una tendencia probabilística. Si el rastro se divide en dos, una hormiga sigue elige con probabilidad más alta el rastro que contiene mayor concentración de feromona. Como a su vez, las hormigas depositan feromona, el camino se va reforzando consiguiendo caminos marcados con una cantidad elevada de feromona. La Figura 1 ilustra este comportamiento de las hormigas (Obtenida de [2]).

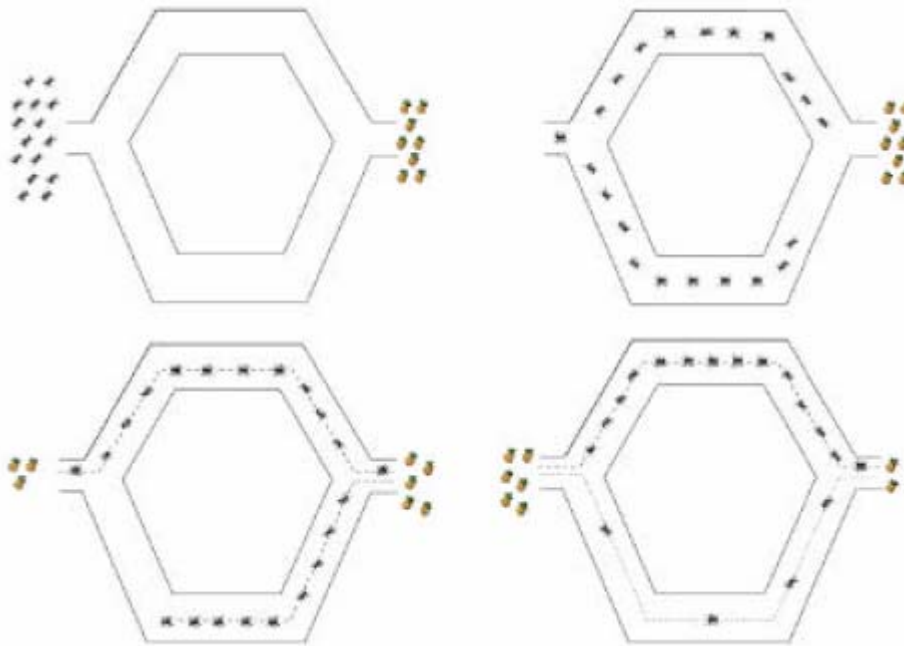


Figura 1: Encontrar el camino óptimo

Los caminos que son más cortos provocan el regreso más temprano de hormigas, y así el depósito de feromona más pronto. Por tanto siguientes hormigas verán condicionado su comportamiento por el camino más corto.

Igualmente, el entorno favorece a la eliminación de los caminos no óptimos provocando la evaporación de la feromona depositada. Sin embargo, los caminos más concurridos sufren menos la evaporación, ya que continúan recibiendo feromona de las hormigas con más frecuencia.

4 La hormiga artificial

Una hormiga artificial es un agente computacionalmente simple que intenta construir una solución a un problema explorando para ello los rastros de feromona disponibles y que debe presentar las siguientes características:

- Busca soluciones válidas de coste mínimo para el problema a resolver.

- Tiene una memoria en la que almacena el camino recorrido para poder reconstruir soluciones válidas, evaluar las soluciones generadas, o bien reconstruir el camino completo llevado por la hormiga.
- Lleva a cabo los movimientos aplicando reglas de transición, que en función de los rastros de feromona que están disponibles, de los valores heurísticos, de la memoria privada de la hormiga y de las restricciones del problema.
- Durante su recorrido, modifica el entorno depositando una cantidad de feromona.
- Una vez que una hormiga ha construido una solución válida puede recorrer el camino de vuelta actualizando los espacios visitados.

Para el desarrollo del simulador de colonia de hormigas, se ha elegido el modelo del autómata celular, donde cada hormiga lleva a cabo un movimiento cada vez (no concurrentemente), y se actualiza el entorno. Las premisas que sigue la construcción de la hormiga artificial son las siguientes:

- Cada hormiga realiza un evento unitario que consiste en desplazarse a una celda adyacente (arriba, abajo, izquierda o derecha), y después cede turno a otra hormiga.
- La hormiga solo es sensible a las señales químicas encontradas en el medio (la feromona depositada en el espacio).
- Las señales a las que es sensible la hormiga son una cantidad de feromona depositada que indica la presencia del hormiguero, y otra que indica la presencia de comida.
- Una hormiga puede tener tres estados: hambrienta, perdida, o con comida.
- Cuando una hormiga está en estado hambrienta, y a falta de señal de comida cercana, deambula al azar con igual probabilidad de desplazarse a cualquiera de las casillas adyacentes, excepto de la casilla de la que procede (cuya probabilidad es menor).

- Cada paso que da una hormiga hambrienta deja en la celda en que se encuentra una señal de hormiguero. En el paso siguiente, dejará una unidad menos de la señal hormiguero. Cuando termine la señal de hormiguero, la hormiga pasará a estado perdida, hasta que encuentre el hormiguero o, en su deambular azaroso, encuentre comida.
- Cuando una hormiga encuentra comida, cambia su estado a “con comida”, y trata de volver atrás usando el mismo camino que siguió a la ida. En cada paso depositará una señal de comida, que irá disminuyendo paso a paso.
- Las hormigas perdidas intentan regresar al hormiguero asignando más probabilidad de movimiento a las celdas adyacentes con mayor cantidad de feromona de hormiguero.
- Las hormigas hambrientas se mueven con mayor probabilidad a casillas con cantidad de señal de comida más alta.
- Cuando una hormiga perdida llega al hormiguero se convierte en una hormiga hambrienta.

En cada iteración la cantidad de feromona de cada celda disminuye, siendo multiplicada por un número menor que 1 (para evitar quedar en mínimos locales).

5 Implementación del modelo

Una vez definido el modelo, se han desarrollado en el lenguaje de programación orientado a objetos Java las clases que ilustran la Figura 2.

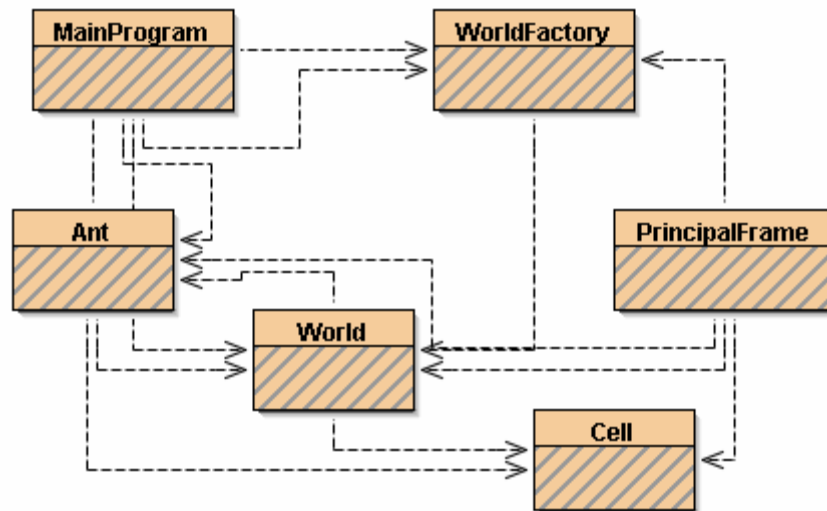


Figura 2: Diagrama de clases de la implementación

La clase World representa el mundo en que viven nuestras hormigas artificiales, y está compuesto por una matriz de objetos Cell y una colección de hormigas (Ant).

Una hormiga hace uso del mundo para conocer las restricciones inherentes al problema, para evaluar los rastros de feromona que encuentra a su paso y para establecer más cantidades de feromona durante su camino.

La clave del movimiento de la hormiga está determinada por el uso de técnicas Monte Carlo a la hora de decidir la casilla siguiente a la que moverse.

Como puede verse en el fragmento de código adjunto, primero se actualiza la información relativa a la celda en que se encuentra la hormiga. Después se inicia el núcleo Monte Carlo, preparando las probabilidades que tendrán las casillas adyacentes para ser la nueva ubicación de nuestra hormiga, asignando un mayor peso en función del estado a las celdas marcadas con la feromona

que determina nuestro objetivo (la comida o el hormiguero). Finalmente se toma la decisión de movimiento.

```
public void moveNext() {
    // Random number and probabilities to go
    float randomNumber;
    float[] probability = new float[4];
    Cell myCell;
    randomNumber = random.nextFloat();

    //Here setup environment values for actual position
    myCell = world.getCellAt(currentX,currentY);
    if (myCell.isAnthole()){
        state = HUNGRY;
        antHoleSignal = ANT_HOLE_SIGNAL;
        pathToFood.clear();
    } else if (myCell.isFood()){
        System.out.println(whoAml + "\t" + pathToFood.size()+" saltos");
        /*for (int i=0; i < pathToFood.size(); i++){
            Cell miMiniCell = (Cell)pathToFood.remove(0);
            System.out.println("[ "+miMiniCell.getX()+", "+miMiniCell.getY()+"]");
        }*/
        antHoleSignal =0;
        foodSignal = FOOD_SIGNAL;
        state = GOING_HOME;
    } // Cell isn't food either anthole

    if (state == HUNGRY && antHoleSignal<=0 ){
        state = HUNGRY_AND_LOST;
    }

    //Init monte-carlo kernel
    for (int i =0; i<4; i++){
        probability[i]=1f;
    }
    probability[lastMove]-=0.8f; //less prob. going back
```

```
//Let's fix our feromona
if (state == HUNGRY && antHoleSignal > 0){
    myCell.setAntholeAmount(myCell.getAntholeAmount() + antHoleSignal);
    antHoleSignal--;
    pathToFood.add(myCell);
}

if (state == GOING_HOME && foodSignal > 0){
    myCell.setAntholeAmount(myCell.getFoodAmount() + foodSignal);
    //foodSignal--;
}

if (state == HUNGRY_AND_LOST){ //It deletes feromona
    //myCell.setAntholeAmount(myCell.getFoodAmount() * 0.9f);
    pathToFood.add(myCell);
}

//To go north
if (world.canMoveTo(currentX, currentY -1)){
    if (state == HUNGRY) {
        probability[NORTH] +=
100*(world.getCellAt(currentX,currentY+1).getFoodAmount());
    } else if (state == GOING_HOME || state == HUNGRY_AND_LOST){
        probability[NORTH] +=
100*(world.getCellAt(currentX,currentY+1).getAntholeAmount() );
    }
} else { //Can't go north
    probability[NORTH]=0;
}

//To go east
if (world.canMoveTo(currentX +1 , currentY)){
    if (state == HUNGRY) {
        probability[EAST] += 100* (world.getCellAt(currentX
+1,currentY).getFoodAmount() );
    } else if (state == GOING_HOME || state == HUNGRY_AND_LOST){
        probability[EAST] += 100*(world.getCellAt(currentX
+1,currentY).getAntholeAmount() );
    }
}
```

```
    }
    } else { // Can't go east
        probability[EAST]=0;
    }

    //To go south
    if (world.canMoveTo(currentX, currentY +1)){
        if (state == HUNGRY) {
            probability[SOUTH] += 100*(world.getCellAt(currentX,currentY-
1).getFoodAmount() );
        } else if (state == GOING_HOME || state == HUNGRY_AND_LOST){
            probability[SOUTH] += 100*(world.getCellAt(currentX,currentY-
1).getAntholeAmount() );
        }
    } else { //Can't go south
        probability[SOUTH]=0;
    }

    //To go west
    if (world.canMoveTo(currentX -1 , currentY)){
        if (state == HUNGRY) {
            probability[WEST] += 100*(world.getCellAt(currentX -
1,currentY).getFoodAmount() );
        } else if (state == GOING_HOME || state == HUNGRY_AND_LOST){
            probability[WEST] += 100*(world.getCellAt(currentX -
1,currentY).getAntholeAmount() );
        }
    } else { // Can't go east
        probability[WEST]=0;
    }

    //Let's normalize
    float allProbabilities=0;
    float[] probToGo = new float[4];
    for (int i=0; i<4; i++){
        allProbabilities += probability[i];
        probToGo[i] = allProbabilities;
    }
}
```

```
randomNumber*=allProbabilities; //Normalize random number with new prob.

// Here starts monte-carlo kernel decission

if (state == GOING_HOME){
    if (pathToFood.size() > 0){
        Cell nextCell = (Cell)pathToFood.remove(pathToFood.size()-1);
        nextCell.setFoodAmount(foodSignal);
        currentX = nextCell.getX();
        currentY = nextCell.getY();
    }
} else {
    if (randomNumber <= probToGo[0] && probability[NORTH] > 0.01) {
        currentY--;
        lastMove = NORTH;
    } else if (randomNumber <= probToGo[1] && probability[EAST] > 0.01) {
        currentX++;
        lastMove = EAST;
    } else if (randomNumber < probToGo[2] && probability[SOUTH] > 0.01) {
        currentY++;
        lastMove = SOUTH;
    } else if (randomNumber < probToGo[3] && probability[WEST] > 0.01){
        currentX--;
        lastMove = WEST;
    }
}
}
```

6 Resultado de la ejecución

La ejecución del programa que implementa nuestro modelo de colonia de hormigas ha establecido como espacio vital una matriz de celdas de 48 filas y 64 columnas. Tiene el hormiguero centrado en la fila 24, columna 32 y una fuente de comida relativamente cercana en la fila 35, columna 30. La Figura 3 ilustra una imagen del mundo de las hormigas.



Figura 3: Hormigas deambulando por el mundo

Hormigas de color negro son hormigas en estado hambriento. El color verde significa hormigas perdidas, y el color rojo son hormigas con comida que regresan al hormiguero.

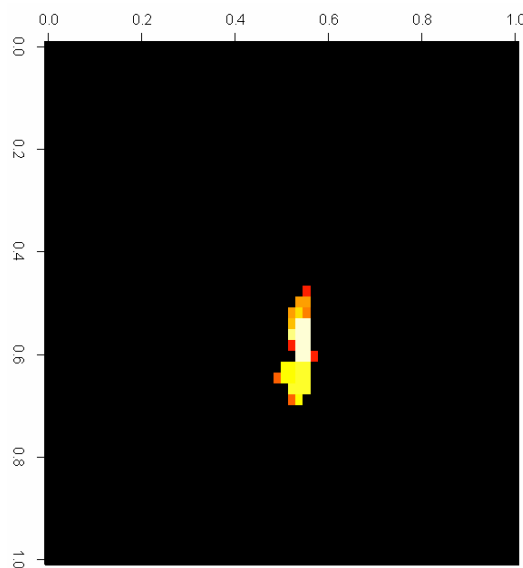


Figura 4: Mapa de intensidad de feromona

La ejecución iterativa del programa permite obtener soluciones en forma del rastro que dejan las hormigas de feromona que marca la comida. Tras diez mil iteraciones, se obtuvo la matriz que contiene las cantidades de feromona de comida emitidas por las hormigas, y se generó con un paquete estadístico la imagen que se muestra en la Figura 4. La zona negra marca ausencia de

feromonas, y los colores cuya luminosidad es mayor (hasta el blanco) marcan una mayor concentración de feromona.

7 Conclusiones

Como podemos observar, el mapa de concentraciones de feromona tras un proceso iterativo (Figura 4), al superponerlo con el mapa del mundo (Figura 3), obtenemos marcada un área del espacio que contiene la distancia mínima entre el hormiguero y la comida. Por tanto nuestro modelo de colonia de hormigas actuando de forma autónoma nos ha provisto de un espacio de soluciones que contiene la solución óptima.

8 Bibliografía

- S. Alonso, O. Cordón, I. Fernández de Viana, F. Herrera: *La metaheurística de optimización basada en colonias de hormigas. Modelos y nuevos enfoques*. Grandada. España. 2003.
- E. Bonabeau, M. Dorigo, G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.