Coordinando agentes móviles para formar patrones tridimensionales

Jorge Moreno

Práctica de *Agentes inteligentes y sistemas multiagentes*,
Máster de Inteligencia Artificial, UPM
j.morenop@alumnos.upm.es

Resumen En este trabajo veremos una extensión del algoritmo SHAPEBUGS de Cheng et al., un algoritmo de coordinación de agentes móviles para la formación de figuras. Se aportarán una serie de nuevas características, siendo la más importante la adaptación de este algoritmo bidimensional a un espacio tridimensional. Otras adiciones alterarán la manera de introducir forma de la figura, el comportamiento de los agentes fuera de la figura o añadirán modificaciones de la figura en tiempo durante la simulación. Además, observaremos algunos de los resultados obtenidos con nuestra implementación en MASON de dicho algoritmo.

Keywords: Swarm ; multi-agent ; patterns ; self-repairing

1. Introducción

El algoritmo SHAPEBUGS, desarrollado en el paper de Cheng et al. [1], busca coordinar los movimientos de un conjunto de agentes para que se posicionen formando una determinada figura de densidad uniforme, pudiendo sobreponerse además a la pérdida de un sector de la población. Se propondrá una extensión de dicho algoritmo, adaptándolo a un espacio tridimensional, y añadiendo una serie de características adicionales que explicaremos más adelante. Además, se ha realizado una implementación de dicha extensión, de la que se mostrarán algunos de los resultados obtenidos en simulaciones.

En este algoritmo, la simulación empezará con solamente un subconjunto de agentes con conocimiento de su posición, a los que se denominará semillas. Los agentes cercanos que no conozcan su posición la irán deduciendo a partir de agentes que ya la conozcan mediante triangulación, que también se usará de forma esporádica para corregir posibles errores localización. En cada paso, los agentes se moverán en respecto a la posición de sus vecinos, sin salirse de la figura. Así, veremos como, cuando un sector de la población de partículas es eliminado, el resto de la población se recoloca para ocupar el espacio vacío.

El algoritmo ha sido implementado en MASON [2], una herramienta pensada específicamente para simulaciones multiagente. En las siguientes secciones, veremos cómo se ha realizado la implementación del algoritmo, así como algunos ejemplos prácticos donde se ha podido comprobar su funcionamiento.

2. Adaptación del algoritmo

Un resumen funcionamiento básico del algoritmo ha sido realizado en la sección anterior. A lo largo de esta sección, se estudiarán los diferentes apartados que forman este algoritmo, y indicará cómo ha sido implementado este algoritmo.

Características de los agentes Los agentes de la simulación serán homogéneos, y tendrán cada uno un pequeño programa interno que se ejecutará de forma independiente, sin la necesidad de ningún otro agente externo que interceda. Estos agentes contarán además con un sensor, con el que estiman la distancia a sus vecinos (con un pequeño error, que se ha recogido en la simulación), una brújula que proporciona una orientación perfecta, y serán capaces de realizar movimientos en línea recta en las tres dimensiones. Además, en nuestra simulación, los agentes contarán con un diámetro determinado, a diferencia del algoritmo SHAPEBUGS, donde este tamaño se despreciaba.

Todos los agentes conocerán la forma de la figura a representar, aunque solamente algunos conocerán de inicio su posición en el espacio, teniendo los demás agentes que calcularlos en base a estas. En la simulación, se han almacenado sus posiciones en dos variables distintas: una *realPosition*, que será la posición real exacta del agente, y que este desconocerá, y la *perceivedPosition*, que será la posición que el agente crea que es su posición real. Los agentes realizarán los diferentes cálculos con su posición percibida, que es la que conocen, y podrán informar de ella a otros agentes mediante una conexión inalámbrica.

El programa interno de los agentes se ha simulado como una clase (*Bug3D*) en nuestra implementación en MASON. En esta clase se almacenan como atributos las variables antes mencionadas (posiciones real y percibida y variables auxiliares con el estado del robot), además de métodos para calcular la posición del agente a partir de las de sus vecinos y para establecer el movimiento a efectuar, ambos explicados más adelante. La parte principal de este programa interno es un método *step()* que es llamado una vez cada etapa de la simulación, y en el que se decide, según el estado del robot y la posición de sus vecinos, los posibles cálculos o movimientos a realizar. En nuestra implementación en MASON, esta es la forma que ha adquirido este método:

```
Si no se conoce la posicion
   intentar localizar mediante la info de sus vecinos
   mover de forma aleatoria
else
   Si el agente se encuentra fuera de la figura
      OPCION 1: eliminar agente de la simulacion
      OPCION 2: Intentar encontrar a un vecino dentro de la figura
                Si se encuentra vecino dentro de la figura,
                   moverse en su direccion
                Si no se encuentra vecino dentro de la figura,
                   moverse de forma aleatoria
```

```
  Si el agente se encuentra dentro de la figura,
     calcular el vector de movimiento a partir de sus vecinos
        Si la futura posicion del agente esta dentro de la figura
           Moverse a dicha posicion
        Si no,
           Moverse de forma aleatoria
Actualizar la informacion sobre la posicion del agente
```

(Pseudocódigo de la funcion *step()* de los agentes)

Cálculo de la posición. Triangulación. Como se ha mencionado, solamente algunos de los agentes tendrán un conocimiento inicial sobre su posición. El resto, deberá calcular su posición a partir de la de agentes que ya dispongan de esta información. Para ello, se ha implementado el método propuesto en el algoritmo SHAPEBUGS: un agente que quiera conocer su posición deberá encontrar a tres vecinos que ya conozcan su posición, y a partir de ellos triangular su posición. El agente buscará, por tanto, una posición (x_p, y_p, z_p), que se corresponda lo mejor posible a la información que ha obtenido de sus vecinos. Como conoce las posiciones percibidas por sus vecinos, y la distancia a la que estos se encuentran de él (proporcionada por el sensor de proximidad), el problema consistirá en minimizar la siguiente diferencia, donde (x_p, y_p, z_p) es la posición a conocer, V_i hace referencia a los vecinos del agente, y d_i la distancia a la que estos se encuentra, obtenida con el sensor de proximidad:

$$\arg \min_{(x_p, yp, zp)} \sum_{V_i} |\sqrt{(x_i - x_p)^2 + (y_i - y_p)^2 + (z_i - z_p)^2} - d_i| \qquad (1)$$

Para obtener un valor aproximado que minimice la función anterior, se ha seguido el consejo del paper de [1] de utilizar el método del gradiente. El cálculo de la posición del agente se realiza en nuestro programa a través del método *tryToLocalizate()*, que si encuentra tres vecinos con posición conocida llama a *calculatePosition(neighbors)*, que a su vez recibe una lista con sus vecinos, y aplica el método del gradiente para calcular su posición[1]. El método *tryToLocalizate()* será llamado desde el método *step()* si el agente no conoce su posición, o cuando quiera volver a comprobar su posición.

Reglas de movimiento. Durante el método *step()*, el agente calculará además cómo ha de moverse para que el conjunto de robots se mantenga en la figura de la forma más uniforme posible. Estos cálculos los efectuará, de nuevo, con la información que le proporcionan sus vecinos, siguiendo un comportamiento inspirado en el Modelo de Expansión de Gases propuesto por Payton et al. [3], tal y como se explica en [1]. Cada agente tiene un radio de repulsión sobre los demás agentes. De esta forma, en cada ejecución del método *step()*, cada

[1] El gradiente de la función ha sido calculado aparte, y se aplica con el método *evGradient()*

agente calculará las fuerzas, en forma de vectores tridimensionales, que los demás agentes ejercen sobre él. Los módulos de estos vectores varían según lo cerca que el agente este del agente repulsor, dentro del radio de repulsión, siendo mayor cuanto más cerca están los agentes. Estos vectores se sumarán entre sí, obteniendo así un vector final que reflejará el vector de movimiento del agente.

Si el vector resultante sacara al agente fuera de la figura, no se aplicaría dicho movimiento, si no que el agente se movería de forma aleatoria, sin salir de la forma. También se movería así en el caso de que el agente todavía no conociera su posición, o en el caso de que el agente pueda quedar fuera de la forma.

Otras características. Como se ha comentado en la introducción, se han añadido características adicionales sobre la implementación efectuada del algoritmo SHAPEBUGS, aparte de la ya citada ampliación a un espacio tridimensional. Una de ellas tiene que ver con la manera en la que se indica el patrón que deben formar los agentes. A la alternativa existente en el SHAPEBUGS, de utilizar un grid (matriz de 0's y 1's), indicando qué sectores del espacio pertenecen a la figura, se ha añadido la opción de aplicar condiciones a las coordenadas, lo cual permite, por ejemplo, el uso de inecuaciones con las que e definir de forma muy simple figuras como esferas, conos o espirales, que con el grid serían muy complejas. También se han implementado posibles modificaciones durante la simulación, con las que poder comprobar la auto-reparabilidad del sistema.

Finalmente, otra característica que se ha añadido han sido los diferentes comportamientos que se le otorga a los agentes que, en algún momento, están fuera de la forma. En el algoritmo original, los agentes que quedan fuera de la forma se mueven de forma aleatoria hasta que vuelven a entrar a la forma. Sin embargo, en este trabajo, planteamos 2 alternativas: (a) los robots intentan encontrar vecinos que estén dentro de la forma para moverse hacia ellos, o (b), como los robots simulan partículas de gas, al quedarse fuera de la forma simulamos que se pierden en la atmósfera, con lo que salen de la simulación. Esta última forma también nos ha permitido comprobar cómo se ajusta el enjambre ante la pérdida de una parte de su población, y lo hace perfectamente.

3. Experimentación

Gracias a nuestra implementación del algoritmo en MASON, se han podido efectuar simulaciones visuales del comportamiento de un sistema multiagente con este algoritmo. A continuación se muestran un par de ejemplos de dos simulaciones distintas, ambas en 3 dimensiones: en la primera, los agentes se deben adaptar a un toro, mientras que en la segundaestán formando una espiral. En ambos casos, se han expuesto estos sistemas a la pérdida de un sector de agentes, mostrando las diferentes fases de la recuperación de la forma original.

Podemos ver como, en ambos casos, los agentes restantes son capaces de sobreponerse a la pérdida del resto de agentes, ocupando las posiciones que estos ocupaban para formar de nuevo una figura uniforme, aunque esta vez con menor densidad.

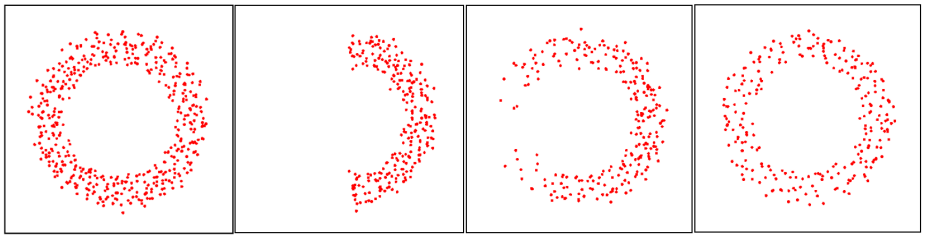

Figura 1. Ejemplo del comportamiento de los agentes para mantener la forma ante la pérdida de un subconjunto de la población (toro tridimensional, visto desde arriba).

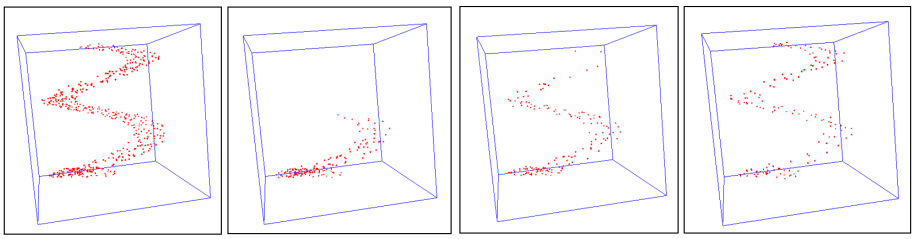

Figura 2. Ejemplo del comportamiento los agentes para mantener la forma ante la pérdida de un subconjunto de la población (espiral).

4. Conclusiones

En este trabajo, se ha implementado en MASON el algoritmo SHAPEBUGS de Cheng et al. [1]. Además, se ha adaptado el algoritmo a un espacio tridimensional, y se han añadido algunas características adicionales. Hemos comprobado como esta herramienta, especfica para simulaciones con sistemas multiagentes, es bastante cómoda y manejable a la hora de implementar este algoritmo.

El algoritmo en sí es bastante simple y efectivo. A pesar de estar pensado para un espacio bidimensional, hemos visto como adaptarlo a una mayor dimensionalidad apenas modifica, más allá de la agregación de unos pocos cálculos para las coordenadas de las dimensiones adicionales. Además, con este paso a tres dimensiones, puede ayudarnos a apreciar todavía mejor la similitud del comportamiento del enjambre de agentes con el comportamiento de los gases en recipientes cerrados.

Referencias

1. Cheng, J., Cheng, W., & Nagpal, R. (2005, July). Robust and self-repairing formation control for swarms of mobile agents. In AAAI (Vol. 5, No. 2005).
2. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7), 517-527.
3. Payton, D., Daily, M., Estowski, R., Howard, M., & Lee, C. (2001). Pheromone robotics. Autonomous Robots, 11(3), 319-324.