Tercera evaluación de Algorítmica II

Germán Alejo Domínguez

Introducción

Tratamos de implementar un algoritmo para resolver un problema: Optimización de enjambre de partículas.

Dentro del programa de remodelado de la Universidad Pablo de Olavide, queremos reubicar los 6 edificios administrativos (reubicándolos físicamente), para optimizar dentro del campus el transito al mínimo tiempo.

Disponemos de 6 ubicaciones físicas y conocemos su distancia que sería la matriz cuadrada de 6x6 simétrica y con la diagonal principal a cero.

Se ha calculado el flujo de personas que circula entre edificios, que será también matriz cuadrada, suponiendo el flujo simétrico.

El objetivo de este trabajo es minimizar el flujo de personas entre los edificios.

Solución Implementada

Para implementar este problema he decidido usar 4 clases:

- Location: la clase location modelará la situación de cada partícula, guardando para ello un array de enteros que representará una posible solución. Entendemos una solución a un array con valores entre 0 y 5 sin repetir representando así el orden de los edificios.

- POSProblem: Clase principal que ejecutara el algoritmo.

- Particle: Clase que modela las partículas, guardamos en esta clase dos localizaciones, una será la que mejor fitness tenga hasta el momento y otra la actual. Además de la velocidad de la partícula y de ambos fitness, el actual y el mejor correspondiente a esa partícula.

- PSOProcess: Esta clase guardara todas las constantes del problema además de la lista de partículas que forman el enjambre. Además, es la que dispone de los métodos para inicializar el problema (matrices y enjambre), calcular el fitness de las partículas, y ejecutar el algoritmo como tal.

Funcionamiento del Problema y variables

Para resolver este problema hemos implementado la clase PSOProcess con los principales métodos y variables estáticas, además esta clase guarda las dos matrices que representan el flujo de personas (flow[][]) y las distancias entre edificios (loc[][]). Estas matrices han sido inicializadas con valores aleatorios entre dos valores máximo y mínimo que determinan la distancia máxima y mínima entre edificios.

Además, inicializamos el enjambre con valores de las localizaciones entre 1 y 6 de forma aleatoria. También se genera de forma aleatoria la velocidad de cada partica, estando está determinada por dos variables.

Los cálculos del fitness se han implementado de acuerdo a la fórmula de la asignación cuadrática.

Los cálculos de la velocidad y el array de soluciones mediante la fórmula del algoritmo de PSO aprendido en teoría.

$$V_{id} = V_{id} + \varphi_1 \cdot rnd() \cdot (pBest_{id} - X_{id}) + \varphi_2 \cdot rnd() \cdot (g_{id} - X_{id})$$
$$X_{id} = X_{id} + V_{id}$$

Resultados obtenidos:

Debido al reducido margen del problema, la solución siempre es obtenida en la primera iteración de ejecución además de que los tiempos de ejecución siempre son muy reducidos (ente 2 y 3 milisegundos).

En el siguiente ejemplo podemos observar como en la primera iteracion se obtiene el resultado con tan solo dos intentos siendo el coste bajo y el tiempo de ejecucion poco mas de dos milisegundos.

```
new best solution found:  cost:136 Location: [3, 2, 0, 5, 1, 4]

new best solution found:  cost:118 Location: [1, 5, 2, 3, 4, 0]

Iteration: 0 Best Solution found in current iteration:  cost:154 Location: [4, 3, 2, 1, 0, 5]
Iteration: 1 Best Solution found in current iteration:  cost:179 Location: [7, 10, 9, 11, 6, 8]
Iteration: 2 Best Solution found in current iteration:  cost:157 Location: [3, 1, 4, 5, 0, 2]
Iteration: 3 Best Solution found in current iteration:  cost:193 Location: [7, 8, 10, 9, 6, 11]
Iteration: 4 Best Solution found in current iteration:  cost:136 Location: [3, 2, 0, 5, 1, 4]
Iteration: 5 Best Solution found in current iteration:  cost:149 Location: [-1, -4, -3, -2, -5, -6]
Iteration: 6 Best Solution found in current iteration:  cost:118 Location: [1, 5, 2, 3, 4, 0]
Iteration: 7 Best Solution found in current iteration:  cost:158 Location: [3, 4, 6, 2, 5, 7]
Iteration: 8 Best Solution found in current iteration:  cost:170 Location: [-1, 1, 2, 0, -3, -2]
Iteration: 9 Best Solution found in current iteration:  cost:156 Location: [6, 9, 7, 8, 5, 4]

Best Solution: cost:118 Location: [1, 5, 2, 3, 4, 0] Iteration:0
Time: 2305.647
BUILD SUCCESSFUL (total time: 0 seconds)
```

En este otro ejemplo obtenemos la solución al problema también al segundo intento de la primera iteración y después el problema no es capaz de encontrar ninguna mejor, el tiempo es menor debido a que las matrices tienen valores distintos en las dos ejecuciones, debido a que se generan de forma aleatoria.

```
new best solution found:  cost:327 Location: [2, 5, 3, 1, 0, 4]

new best solution found:  cost:291 Location: [5, 1, 2, 4, 0, 3]

Iteration: 0 Best Solution found in current iteration:  cost:341 Location: [1, 2, 4, 5, 3, 0]
Iteration: 1 Best Solution found in current iteration:  cost:418 Location: [10, 9, 6, 7, 11, 8]
Iteration: 2 Best Solution found in current iteration:  cost:349 Location: [1, 0, 3, 4, 2, 5]
Iteration: 3 Best Solution found in current iteration:  cost:373 Location: [-6, -5, -3, -4, -2, -1]
Iteration: 4 Best Solution found in current iteration:  cost:327 Location: [2, 5, 3, 1, 0, 4]
Iteration: 5 Best Solution found in current iteration:  cost:337 Location: [-2, 0, -4, -1, -3, 1]
Iteration: 6 Best Solution found in current iteration:  cost:333 Location: [3, 5, 1, 2, 4, 0]
Iteration: 7 Best Solution found in current iteration:  cost:291 Location: [5, 1, 2, 4, 0, 3]
Iteration: 8 Best Solution found in current iteration:  cost:340 Location: [1, 3, 2, 0, 5, 4]
Iteration: 9 Best Solution found in current iteration:  cost:376 Location: [7, 11, 6, 10, 8, 9]

Best Solution: cost:291 Location: [5, 1, 2, 4, 0, 3] Iteration:0
Time: 2195.736
BUILD SUCCESSFUL (total time: 0 seconds)
```

Si aumentamos el número de iteraciones y partículas la solución apenas se ve afectada, la diferencia de costes es imperceptible y los tiempos de ejecución son prácticamente los mismos.

Ejecución con 100 iteraciones:

```
Iteration: 94 Best Solution found in current iteration:  cost:291 Location: [4, 8, 7, 9, 5, 6]
Iteration: 95 Best Solution found in current iteration:  cost:268 Location: [-3, -1, 0, -5, -2, -4]
Iteration: 96 Best Solution found in current iteration:  cost:260 Location: [1, -2, -1, 0, -3, 2]
Iteration: 97 Best Solution found in current iteration:  cost:232 Location: [2, 4, 3, 1, 0, 5]
Iteration: 98 Best Solution found in current iteration:  cost:298 Location: [-1, -2, 3, 0, 2, 1]
Iteration: 99 Best Solution found in current iteration:  cost:271 Location: [6, 4, 7, 3, 5, 8]

Best Solution: cost:232 Location: [2, 4, 3, 1, 0, 5] Iteration:0
Time: 20522.307
BUILD SUCCESSFUL (total time: 0 seconds)
```