4 | Recocido simulado

Benjamín Torres Saavedra
Verónica Esther Arriola Ríos

Objetivo

Que el alumno se familiarice con una estrategia de mejoramiento iterativa para la resolución de problemas conocida como recocido simulado.

Introducción

Recocido simulado

Este algoritmo puede ser visto como una mejora al algoritmo de ascenso de colinas, el cual mejora de manera iterativa una representación de la solución a un problema hasta que se encuentre el óptimo o se estanque en un máximo local. Recocido simulado integra una selección de soluciones estocástica, es decir, para elegir la siguiente solución no siempre se escoge la mejor, con la cual se le da la libertad de explorar zonas distintas del espacio de soluciones, en las cuales el ascenso de colinas puede detenerse fácilmente.

El pseudocódigo de este algoritmo es descrito en el Russell y Norving 2010 en la sección de algoritmos de mejoramiento iterativo y lo podemos ver en el Algoritmo 1.

Problema del agente viajero (TSP)

Este problema consiste en encontrar, dada una lista de ciudades, una ruta que pueda seguir un agente para recorrer todas las ciudades y volver a aquella en la cual inició el viaje. Para que este recorrido valga la pena debe ser lo más económico posible o, en su defecto, recorrer la menor distancia que permita visitar todas las ciudades.

Algoritmo 1 Recocido simulado

```
function RecocidoSimulado(problema, horario)
    # problema: un problema
    # horario: mapeo de tiempo a temperatura.
    actual ← HazNodo(EstadoInicial(problema))
    for t ← 1 a ∞ do
        T ← horario[t]
        if T = 0 then return actual
        end if
        siguiente ← un sucesor de actual elegido al azar.
        ΔE ← Valor(siguiente) - Valor(actual)
        if ΔE > 0 then
            actual ← siguiente
        else
            actual ← siguiente sólo con probabilidad e^(ΔE/T)
        end if
    end for
end function
```

Este problema es ampliamente conocido por ser de la clase NP-Completo, es decir, que no se puede resolver en tiempo polinomial por algoritmos deterministas, afortunadamente para esta práctica no usaremos un algoritmo de tal tipo y podremos aproximarnos a una solución en un tiempo razonable.

Desarrollo e implementación

Se les proporcionará código fuente en Java, que deberán completar con la finalidad de resolver el problema del agente viajero.

El código fuente se encuentra en los archivos `Solucion.java`, es que se encargará de representar a una ruta del agente viajero que pase por todas las ciudades y el archivo `RecocidoSimulado.java`, que implementará el algoritmo anteriormente descrito.

Implementación

Se debe programar una clase hija de `Solución`. Esta clase hija agregará los atributos del(de los) tipo(s) necesario(s) para representar una propuesta de solución al problema que se desea resolver.

En la página http://www.math.uwaterloo.ca/tsp/world/countries.html#DJ se pueden encontrar una serie de países con ciudades y sus correspondientes coorde-

nadas. Sin asumes una conectividad total de las ciudades y usas la distancia euclideana como métrica, tu tarea será tratar de aproximarte lo mejor posible a la longitud del camino óptimo de la ciudad seleccionada. Puedes incorporar la información a tu programa como te sea más conveniente.

Los métodos a implementar dentro de una clase hija de `Solución` son:

- Un constructor.

  Este método deberá inicializar una representación con una propuesta para solución de un problema, en nuestro caso, una ruta del problema del agente viajero. Deberás elegir cómo representar la solución. No es necesario que dicha solución sea correcta.

  En el caso del agente viajero puede ser que la solución visite todas las ciudades, pero que la distancia recorrida no sea mínima y/o que las visite más de una vez.

- `public Solución siguienteSolución()`

  Genera una nueva solución perturbando de manera aleatoria al objeto que lo llama. Observa que al sobreescribir el método puedes cambiar el tipo de regreso para evitar hacer audiciones (*castings*).

- `public float evaluar()`

  Califica la solución actual según la heurística elegida de acuerdo al problema a resolver.

Para la clase `RecocidoSimulado`:

- `public RecocidoSimulado()`

  Inicializa los parámetros del algoritmo. En principio ya funciona así, pero la puedes modificar si lo consideras necesario.

- `public float nuevaTemperatura()`

  Calcula la nueva temperatura, se espera que a lo largo de las iteraciones este valor decrezca, llegando a cero en el último paso.

- `public Solucion seleccionarSiguienteSolución()`

  Dada la solución actual, este método debe modificarla para obtener una solución nueva y elegir esta nueva solución con cierta probabilidad dependiendo de si es mejor o no, según el algoritmo presentado anteriormente.

- `public Solución ejecutar()`

  Ejecuta el algoritmo con los parámetros inicializados, puedes modificar la firma si lo consideras necesario.

Finalmente, en la clase `Main` únicamente se crea un objeto tipo `RecocidoSimulado` que ejecuta el algoritmo por algún número de iteraciones:

- Calcula los parámetros para el constructor de`RecocidoSimulado`:
  - ⋆ Crea un objeto de la clase hija de `Solucion` que implementaste.
  - ⋆ Calcula la temperatura inicial y decaimiento adecuados para ejecutar recocido simulado dependiendo del número de interacciones.
- Modificar el ciclo para monitorear la evolución del algoritmo entre interacciones.

Punto extra

Realiza las modificaciones que consideres pertinentes para poder utilizar una estrategia similar al recocido simulado para minimimzar la siguiente función:

$$f(x,y) = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos(2\pi x)+\cos(2\pi y))} + e + 20$$

para $-5 \leqslant x, y \leqslant 5$.

Requisitos y resultados

El código debe ser implementado de manera eficiente y estar documentado para esclarecer su funcionamiento. Además, debe encontrar una solución válida al problema del agente viajero.