5 | Recocido simulado

Benjamín Torres Saavedra
Verónica Esther Arriola Ríos

Objetivo

Que el alumno se familiarice con la estrategia de mejoramiento iterativa para la resolución de problemas mediante búsquedas en el espacio de estados, conocida como recocido simulado.

Introducción

Recocido simulado

Este algoritmo puede ser visto como una mejora al algoritmo de ascenso de colinas. Ascenso de colinas comienza con la propuesta de una solución parcial o no óptima a un problema, la cual mejora de manera iterativa hasta encontrarse con la óptima o estancarse en un máximo local. Recocido simulado integra una selección de soluciones estocástica, es decir, para elegir la siguiente solución no siempre escoge a la mejor vecina, con lo 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 libro de Russell y Norving 2010 en la sección de algoritmos de mejoramiento iterativo y lo podemos ver en el Algoritmo 2. Observa que, en esta versión, se asume que una mejor solución tendrá un valor más alto. Si se desea minimizar el valor hay que invertir las indicaciones.

Algoritmo 2 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 < ε 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
```

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.

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

Desarrollo e implementación

Para esta práctica cuentas con código auxiliar en Java, que deberás completar con la finalidad de resolver el problema del agente viajero.

El código fuente consta de los archivos:

- `recocido/Solución.java`. Se encargará de representar una ruta del agente viajero que pase por todas las ciudades,

- `recocido/RecocidoSimulado.java`. Implementará el algoritmo anteriormente descrito.
- `recocido/DatosPAV.java`. Lee la información desde los archivos de datos con la información de las ciudades que se utilizarán.
- `recocido/Main.java`. Contiene el esqueleto con la idea principal de cómo utilizar el resto del código para buscar soluciones.

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 coordenadas. Se incluye un ejemplar pequeño de esta página con esta práctica para que te sirva al probar tu código. Si 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 para recorrer todas las ubicaciones de las ciudades en el país seleccionado. 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. Puedes modificar la firma del constructor como consideres necesario para crear un propuesta de solución inicial aleatoriamente a partir de la especificación del problema a resolver.

  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. La especificación del problema está dada por los archivos `.tsp` con la información sobre las ciudades.

- `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.

  En este caso la evaluación estará relacionada con la longitud del recorrido propuesto por la solución actual. La función de evaluación que elijas debe cumplir que el recorrido óptimo del problema del agente viajero satisface que $\acute{o}ptimo.evaluar() \leqslant s.evaluar()$ para toda solución posible.

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 Solución seleccionarSiguienteSolución()`

  Dada la solución actual, este método debe obtener una modificación suya 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 y devuelve la mejor solución encontrada.

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

- Usa el argumento en `args[0]`, que deberá ser la ubicación de un archivo `.tsp`, para cargar la descripción de una ciudad. Se te da la clase `DatosPAV` para facilitar esta tarea.
- Calcula los parámetros para el constructor de `RecocidoSimulado`:
  - ⋆ Crea un objeto de la clase hija de `Solución` que implementaste, de modo que represente una primer ruta a través de la ciudad descrita en el archivo `.tsp`.
  - ⋆ Calcula la temperatura inicial y decaimiento adecuados para ejecutar recocido simulado dependiendo del número de iteraciones.
- Modifica el ciclo para monitorear la evolución del algoritmo entre iteraciones.

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.