Problema del Agente Viajero

Rafael de Jesús García García
Facultad de Ciencias
Universidad Nacional Autónoma de México

24 de marzo de 2018

Resumen

Dado un problema NP-duro, en este caso el Problema del Agente Viajero o TSP por sus siglas en inglés, se presenta una heurística llamada "Recocido Simulado" ó "Enfriamiento simulado" que se explicará más adelante. Esta heurística es capaz de obtener resultados muy cercanos a la solución en un tiempo mucho menor que si usáramos un algoritmo de fuerza bruta para resolverlo. Se modeló el problema usando el lenguaje de programación Scala, el cual es orientado a objetos y funcional.

1. Introducción

1.1. Problema del Agente Viajero

Ejemplar: Una gráfica G con un conjunto de vértices V que representa las ciudades y aristas E que representan los caminos entre las ciudades.

Problema: ¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad de origen?

Este problema es de los problemas más conocidos en las Ciencias de la Computación. Es un problema NP-duro dentro de la optimización combinatoria. [1]

En nuestro caso vamos a limitar nuestro problema a encontrar una ruta que visite cada ciudad exactamente una vez, pero sin que la última ciudad y la primera estén conectadas. Tendremos dos ejemplares del problema, uno con 40 ciudades y otro con 150.

1.2. Recocido Simulado

El recocido simulado es una meta heurística para aproximar al máximo global de una función dada en un espacio de búsqueda muy grande. El nombre de esta técnica viene del recocido en la metalurgia, una técnica que involucra calentamiento y el enfriamiento controlado de un material para incrementar el tamaño de sus cristales y reducir sus defectos. La simulación del recocido puede ser usada para encontrar una aproximación a un máximo global para una función con un número grande de variables. [2]

2. Modelo

El problema se dividió en distintas clases para su mejor funcionamiento. A continuación se detallaran brevemente el funcionamiento de cada una de ellas.

La primera clase `Solution` describe una solución, la cual contiene una lista de ciudades en orden que representa el camino, el valor de la función de costo que evalúa a esa solución y algunas constantes que ayudan en ese cálculo, las conexiones existentes, una función para obtener un vecino aleatorio de la solución, y una función para deteminar si es factible la solución.

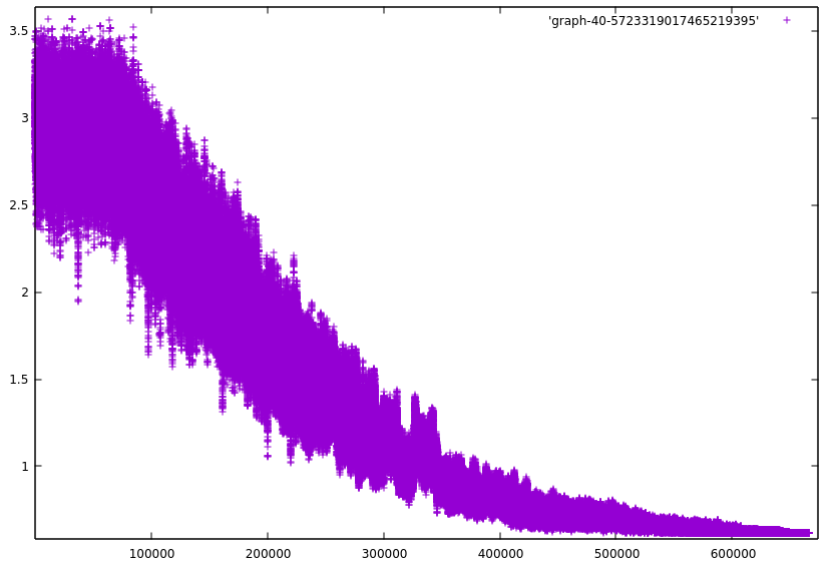
La clase `Temperature` contiene los algoritmos para calcular una temperatura inicial para el sistema basándose en un valor inicial que se define de manera experimental.

La clase `SimulatedAnnealing` contiene el algoritmo de aceptación por umbrales.

Finalmente, el objeto `Parameters` contiene los parámetros experimentales con los que va a trabajar el sistema.

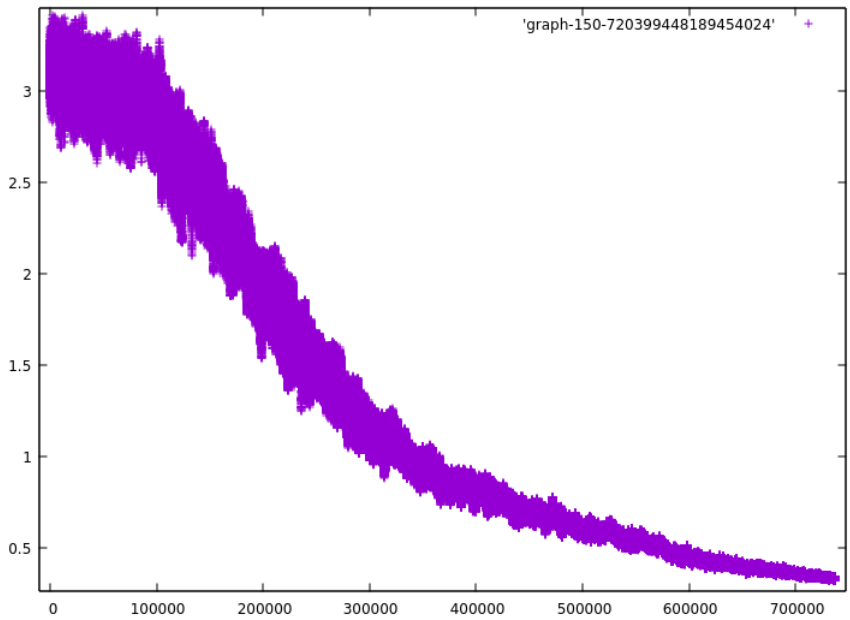
3. Experimentación

Usando la heurística de recocido simulado se pueden obtener muy buenos resultados, sin embargo, la función `neighbor` de la solución no es determinista, por eso se usa un generador de números aleatorios y se guarda la semilla para poder reproducir los resultados. A continuación se mostrarán unas gráficas de cómo el sistema va mejorando su función de costo hasta llegar al resultado final para esa ejecución.



Esta gráfica es la ejecución para el ejemplar de 40 ciudades con evaluación de 0.616235865 con los siguientes parámetros:

- T_i: 8
- Tamaño de lote: 1000
- Porcentaje de aceptados (P): 95 %
- ε: 0.001
- ε_T: 0.001
- ε_P: 0.001
- φ: 0.99



Esta gráfica es la ejecución para el ejemplar de 150 ciudades con evaluación de 0.285108488 con los siguientes parámetros:

- T_i: 8
- Tamaño de lote: 1000
- Porcentaje de aceptados (P): 95 %
- ε: 0.001
- ε_T: 0.001
- ε_P: 0.001
- φ: 0.99

4. Conclusiones

Usando la heurística de recocido simulado se pudo ver que se encuentran soluciones suficientemente buenas para un problema que de otra forma sería imposible de resolver por cuestiones de tiempo. Otro factor que influyó en la realización del proyecto es el modelo. Haciendo un diseño bastante adaptivo desde el inicio permite mejorar el sistema después. Por ejemplo, se pueden agregar optimizaciones para que el sistema se ejecute lo más rápido posible, ya que el tiempo es vital en experimentos de este estilo. En este caso, se agregó al final una optimización para no calcular la función de costo cada vez que se cree un vecino nuevo, sino calcular solo las distancias que cambian modificar la función de costo actual, esto se logra en tiempo constante, en vez de tiempo lineal como se hacía al inicio para calcular la función.

El problema del agente viajero tiene muchas aplicaciones como en genética, por mencionar alguna. La importancia de este tipo de técnicas para parcialmente resolver problemas NP-duros es relevante en el mundo actual, ya que con cantidades tan grandes de información creciendo cada día es necesario resolver problemas cada vez más rápido.

Referencias

[1] Travelling Salesman Problem,
`https://en.wikipedia.org/wiki/Travelling_salesman_problem`

[2] Simulated Annealing,
`https://en.wikipedia.org/wiki/Simulated_annealing`