Implementación y Optimización del método heurístico Hill Climbing para resolver el Traveling Salesman Problem (TSP)

Miguel López Cruz Carlos Román López Sierra José Luis R. Zárate Cortés

Primavera 2021

Índice

Resumen

Si bien es cierto que ...

1. Introducción

En optimización matemática y ciencias de la computación, los métodos heurísticos y metahurísticos se desarrollaron para resolver problemas del tipo large scale (PL´s o de optmización combinatoria) que son muy grandes y complicados como para resolverlos por medio de algoritmos exactos. Típicamente se ajustan a problemas especificos.

En nuestro caso, seleccionamos Hill Climbing, método de optimización que pertenece al grupo de búsqueda local. Encuentra la solución óptima en el caso de problemas convexos (como el algoritmo símplex que resuelve problemas de optimización convexa mediante hill climbing) y soluciones óptimas locales (no necesariamente son soluciones óptimas globales) entre todas las posibles soluciones del espacio de búsqueda. Por lo tanto, es importante remarcar que no existe garantía acerca de la calidad de la solución que se obtiene.

Existe un problema importante conocido como el Traveling Salesman Problem (TSP), el cual plantea la siguiente interrogante:

"Dado un conjunto de ciudades y la distancia que existe entre cada par de ellas... ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y retorna a la ciudad orígen?"

Responder a estar interrogante es equivalente a calcular todas las rutas posibles (permutaciones) y determinar la de menor distancia (algortimo de fuerza bruta), lo cual implica un tiempo de ejecución que cae en un factor polinommial el orden $O(n!)$ (el factorial del número de ciudades), lo cual es impráctico tan sólo para 20 ciudades.

El método Hill Climbing se puede aplicar para resolver el TSP y obtener una solución óptima para un conjunto pequeño de ciudades de forma muy rápida. Conforme se incrementa el tamaño del conjunto, este método requiere mayor tiempo para explorar el espacio de posibles soluciones de forma suficiente para encontrar una solución óptima global. Por lo que, nuestro objetivo es explorando diversas opciones que permiten una implementación más eficiente del método, ya sea

- Modificaciones en la técnica, p.e. random restart
- Integrar eficiencia de código como cómputo en paralelo o multiprocessing
- Utilizar estrategias de solución como el uso de Kubernetes (minikube) con contenedore de Docker buscando obtener soluciones más cercanas al óptimo global en un periodo de tiempo razonable.

2. Optimización Convexa

3. Optimización Combinatoria

INVESTIGAR Y SINTETIZAR OPTIMIZACIÓN COMBINATORIA

4. Sobre métodos heurísticos en optimización combinatoria

NOTA JL -¿REVISAR EN WIKI Y EXTRAER UNA SINTESIS

Los métodos heurísticos y meta heurísticas (estrategias para mejorar o diseñar métodos heurísticos) encuentran una buena solución factible que al menos está razonablemente cerca de ser óptima, también pueden reportar que no se encontró tales soluciones. Sin embargo son métodos que no dan una garantía acerca de la calidad de la solución que se obtiene.

Los métodos heurísticos se desarrollaron principalmente para el manejo de problemas large scale sean PL's o de optimización combinatoria. Típicamente se han ajustado a problemas específicos en lugar de una variedad de aplicaciones.

No existe garantía de que la mejor solución que se encuentre con un método heurístico sea una solución óptima o incluso que esté cerca de serlo. Estos métodos son utilizados para abordar problemas que son muy grandes *(Large Scale)* y complicados como para resolverlos por medio de algoritmos exactos.

5. Busqueda local en problemas de optimización

Los algoritmos de búsqueda local operan buscando desde un estado inicial a estados vecinos, sin realizar un seguimiento de las rutas ni del conjunto de estados que se han alcanzado. Eso significa que no son sistemáticos; es posible que nunca exploren una parte del espacio de búsqueda donde un la solución reside realmente. Sin embargo, tienen dos ventajas clave que son:

1. Utilizan muy poca memoria
2. A menudo pueden encontrar soluciones razonables en espacios de estados grandes o infinitos para los cuales los algoritmos sistemáticos no son adecuados.

Para comprender la búsqueda local, considere los estados de un problema presentado en un espacio de estados, como se muestra en la Figura 1 . Cada punto (estado) tiene una .elevación", definida por el valor de la función objetivo. Entonces el objetivo es encontrar el pico más alto, un máximo global.

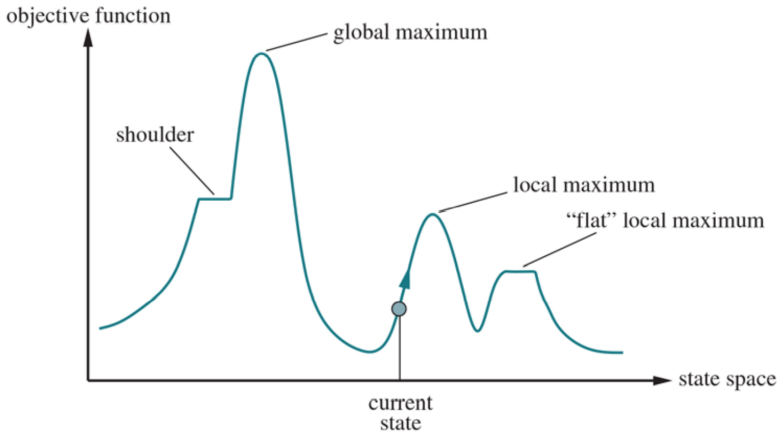

Figura 1: Espacio de estados para la función objetivo

6. Hill Climbing

El algoritmo de Hill Climbing realiza un seguimiento de un estado actual y en cada iteración se mueve al estado vecino con el valor más alto, es decir, se dirige en la dirección que proporciona el ascenso más empinado. Termina cuando alcanza un "Pico" donde ningún vecino tiene un valor más alto. Hill Climbing no mira hacia el futuro más allá de los vecinos inmediatos del estado actual. Esto se asemeja a tratar de encontrar la cima del Monte Everest en una espesa niebla mientras sufre de amnesia.

Por otro lado, otra forma de utilizar Hill Climbing consiste en utilizar el negativo de una función de coste heurística como función objetivo; que ascenderá localmente al estado con menor distancia heurística a la meta.

A este tipo de algoritmos a veces se les llama búsqueda local greedy porque toma un buen estado vecino sin pensar en los estados futuros a dónde moverse. Hill Climbing puede progresar rápidamente hacia una solución porque, por lo general, es bastante fácil mejorar un mal estado. Desafortunadamente, Hill Climbing tiene problemas en áreas donde espacio de posibles soluciones puede atascarse, estos son:

- **Maximo Local:** Es un pico que es más alto que los estados vecinos pero por menor que el máximo global. Los algoritmos Hill Climbing que alcanzan las proximidades de un máximo local tomaran estados vecinos que lo lleven en esta dirección, pero luego de llegar a este punto les será imposible salir de él.
- **Cordilleras y corredores:** Este tipo de puntos en el espacio de soluciones resulta en una secuencia de máximos locales que es muy difícil de navegar para los algoritmos greedy, esto por el gran número de soluciones vecinas que te llevan a puntos no óptimos de la función objetivo.

- **Mesetas:** Esta se encuentra cuando el espacio de búsqueda es plano, o suficientemente plano de manera tal que el valor retornado por la función objetivo es indistinguible de los valores retornados por regiones cercanas debido a la precisión utilizada por la máquina para representar su valor. En estos casos el algoritmo puede que no sea capaz de determinar en qué dirección debe continuar y puede tomar un camino que nunca conlleve a una mejoría de la solución.

6.1. Variantes de Hll climbing

Se han inventado muchas variantes de este algoritmo. Por ejemplo Hill climbing estocástica elige al azar entre los movimientos cuesta arriba; la probabilidad de selección puede variar con la inclinación del movimiento cuesta arriba. Esto suele converger más lentamente que el steepest ascent, pero en algunos puntos de espacio de soluciones, encuentra mejores soluciones. Firs-choice hill climbing implementa el método estocástico generando sucesores aleatoriamente hasta que se genera uno que es mejor que el estado actual. Esta es una buena estrategia cuando un estado tiene muchos posibles estados siguientes a donde moverse.

También existe el algoritmo *random-restart Hill climbing* está construido sobre la base de hill climbing. Este realiza, iterativamente, el hill-climbing, cada vez con una condición inicial aleatoria . Una primera mejor solución es guardada; si una nueva corrida del hill climbing produce una mejor que el estado guardado, lo reemplaza. Hill climbing de reinicio aleatorio es un algoritmo sorprendentemente efectivo en muchos casos. De esto se deriva que con frecuencia es mejor gastar tiempo de CPU explorando el espacio, que cuidadosamente optimizar desde una condición inicial.

7. Implementación

8. Conclusiones