Problema del Agente Viajero usando Recocido Simulado

César Lara

Heurísticas de Optimización Combinatoria, UNAM

Octubre 2019

1 Introducción

El problema de el agente viajero es uno de los problemas más famosos y más estudiados dentro de la teoría de la computación. En este proyecto uso una de las técnicas más conocidas para atacar el problema y conseguir soluciones no terribles en un intervalo de tiempo bastante humano y aceptable: La heurística de Recocido Simulado.

2 Problema del Agente Viajero

Se tienen n ciudades y un agente tiene que hacer un recorrido pasando por cada una de las n ciudades una sola vez. Entre cada par de ciudades existe una distancia y el punto del problema es minimizar la distancia total del recorrido.

2.1 Solución

De ésto se sigue que una solución es una permutación de las n ciudades donde la ciudad 0 representa el punto de partida, la ciudad 1 la siguiente ciudad visitada y la ciudad n el último destino. En esta implementación del problema, tendremos aristas con un costo exageradamente alto (llamaremos al conjunto de tales aristas como T) para introducir el concepto de **solución factible**.

2.2 Solución factible

Una solución se dice factible si ninguna de sus aristas pertenece al conjunto T. El conjunto T se puede entender como un conjunto de aristas trampa. Así sabemos que una solución factible es una solución que no es terriblemente, horripilantemente mala; tampoco significa que es buena. Se asegura que para cualquier conjunto de vértices existe un camino entre ellos que produce una solución factible.

2.3 Costo

De el concepto de solución factible se deriva el **costo** de una solución. Una solución factible tendrá *costo* ≤ 1, el costo de una no factible sera mayor a 1. No se debe de cometer el error de querer comparar dos costos con nociones más allá de las que son posibles de expresar por los símbolos $<$, $>$ y $=$, que un costo sea el doble de grande que otro no significa que una de las respectivas soluciones sea *el doble de mejor* que la otra o algo así, tal idea no es necesariamente cierta.

2.4 Vecino

Un vecino de una solución se define como un intercambio de cualesquiera dos elementos de la solución.

3 Heurística

La heurística usada en este proyecto es la del recocido simulado con aceptación por umbrales.
La idea es comenzar a explorar el espacio de búsqueda desde una solución inicial S_0. Manejamos el concepto de temperatura, la cual nos ayudará a decidir si aceptamos una solución S' (vecina de S) dependiendo de qué tan alta o tan baja la temperatura es. Mientras más alta la temperatura, más probable será aceptar una solución peor a S. Conforme el sistema avanza, la temperatura decrementa y tenemos menos margen de aceptar peores soluciones. Calcular un lote significa explorar el espacio de búsqueda sin decrementar la temperatura hasta llenar cierta cantidad de soluciones aceptadas para finalmente decrementar la temperatura y seguir buscando. Cuando la temperatura llega a un cero virtual, la ejecución termina.

3.1 Determinación de temperatura inicial

Queremos que para la solución inicial dada S_0, el numero de soluciones que el sistema aceptará recién se arranque sea alto, en especifico podemos declarar el porcentaje deseado P y en base a esto la temperatura se ajusta con una búsqueda binaria al inicio del recocido.

3.2 Barrido de vecindad

Esta técnica consiste en buscar dentro de todos los vecinos de nuestra solución alguna solución con un costo menor, si tal existe, buscamos dentro de sus vecinos una solución mejor y repetimos el proceso hasta que estemos en una solución que sea la mejor dentro de su vecindad.

3.3 Parámetros

3.3.1 Ritmo de decaimiento

Ritmo en que la velocidad irá disminuyendo conforme exploramos soluciones. Expresado en el código como *TEMPERATRE_DECAY*.

3.3.2 Épsilon de temperatura

Cero virtual que indica cuándo una temperatura es lo suficientemente pequeña. Expresado en el código como *EPSILON*.

3.3.3 Épsilon de Porcentaje de soluciones aceptadas

Cero virtual que indica cuando un porcentaje de soluciones aceptadas es lo suficientemente cercano al porcentaje meta. Expresado en el código como *EPSILON*.

3.3.4 Porcentaje de soluciones aceptadas al inicio del recocido

Expresado en el código como *P*.

3.3.5 Tamaño de lote

Expresado en el código como *L*.

3.3.6 Alto

Se usa para salir de un lote después de *stop* soluciones exploradas. Expresado en el código como *STOP*.

4 Experimentación

Para poder experimentar con nuestro sistema, tenemos dos conjuntos de ciudades, uno de tamaño 40 y otro de 150. Notese que para encontrar la solución óptima formalmente para la instancia de 150 ciudades, necesitamos pasar por todas las soluciones posibles, que son más que todos los átomos en el universo. La buena noticia es que las soluciones posibles de la instancia de 40 ciudades son menos que todos los átomos.

Dentro de un batch de 2000 semillas, la mejor solución para la instancia de 40 tuvo un costo de **0.250989872** y corre en 23 segundos. Los parámetros usados fueron:

```
TEMPERATURE_DECAY = .95
EPSILON = 1e-4
P = .80
L = 1000
STOP = L * 31
```

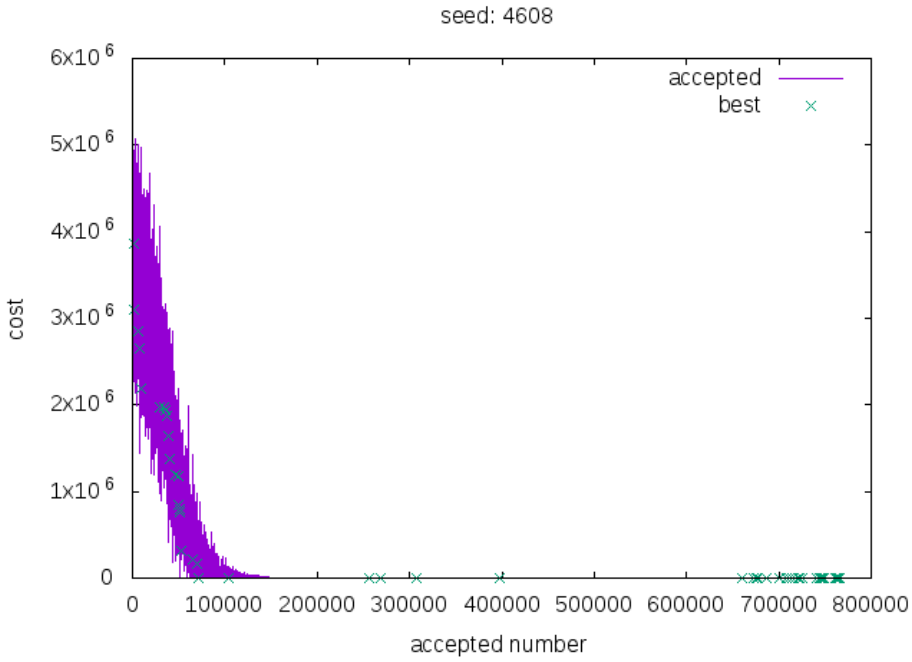

Figure 1: Mejor solución encontrada, 40 cuidades.

Notese que esta solución coincide con la menor conocida para esta instancia.

La mejor solución encontrada para la instancia de 150 ciudades dentro de un batch de 2000 semillas tuvo un costo de **0.142**, se sabe de una mejor solución con costo de 0.141.

Para tratar de mejorar este resultado se implementó el barrido de vecinos (aplicado a la mejor solución encontrada sin el barrido) y también cada semilla comienza en una solución random, haciendo que el espacio de búsqueda explorado sea mas diverso.

Después de implementar estos mecanismos, se volvieron a utilizar al rededor de 2 mil semillas (no exactamente las anteriores) para encontrar soluciones. Desgraciadamente no se pudo llegar muy cerca a los resultados anteriores o a la mejor solución conocida. El costo esta vez es fue de **0.14602046**.

Los parámetros fueron idénticos a los anteriores (de 40 ciudades) excepto que el tamaño del lote fue cambiando a 2000 porque por observación, la cantidad de soluciones no factibles en la instancia de 150 ciudades se redujo de alrededor de 3 por cada 10 semillas (con lote de 1000) a 2.5 con lote de 2000.

Aquí podemos observar que el barrido de vecinos hizo su trabajo al final de la heurística y muchas soluciones mejores fueron encontradas. Igual notamos que el cero virtual de la temperatura es muy bajo, me podría ahorrar unos segundos

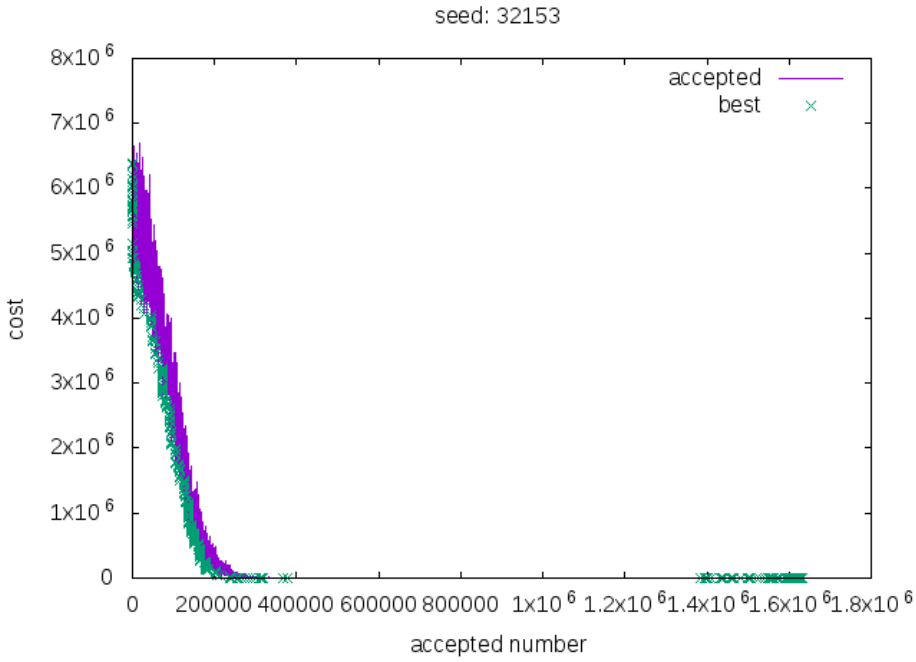

Figure 2: Mejor solución encontrada después de los cambios, 150 cuidades.

de ejecución haciendo este cero un poco mas pequeño.

Igualmente me pasó que al dejar ejecutándose muchas semillas, la ejecución del barrido de vecinos tardó demasiado en terminar en una semilla y tuve que detenerlo porque no se llegaba a un mínimo local. Puse un limite en el numero de soluciones aceptadas por barrido para que esto no vuelva a pasar.

5 Conclusión

Usar una implementación y un lenguaje que funcionen rápido es fundamental para poder experimentar con distintas semillas y técnicas para así llegar a soluciones buenas o excelentes. Igualmente me parece increíble que estas técnicas den soluciones buenas en cuestión de segundos cuando el espacio de búsqueda es enorme y la solución promedio es terrible. El concepto que recocido simulado por aceptación por umbrales emplea es sencillo de comprender y no creo que sea difícil de que a alguien se le ocurra de la nada, aún así da buenas soluciones y es una gran manera de hacer que el problema sea menos monstruoso.

References

[1] https://www.wolframalpha.com/input/?i=atoms+in+the+universe+<+150!