Seminario de Ciencias de la Computación B
Heurísticas de Optimización Combinatoria
Problema del Ajente Viajero
con Recocido Simulado

Profesor: Canek Peláez Valdés
Autor: Xin Wen Zhang Liu

Viernes, 17 de Marzo, 2023.

El problema del agente viajero

El problema del agente viajero es uno de los problemas de Optimización combinatoria más estudiados. Introducido por primera vez en 1930

Este problema plantea una pregunta fácil de hacer pero difícil de repsonder:

Dado un conjunto de ciudades y sus coordenadas en el plano cartesiano, ¿cuál es el camino más corto que visite cada ciudad exactamente una vez?

La manera má directa de resolver este problema sería listar todas las prosibles combinaciones de caminos que pasen por las ciudades deseadas y comparar sus costos. Sin embargo el número de combinaciones diferentes con n ciudades crece a la par de $n!$, lo que hace que la cantidad de posibles combinaciones crezca a un paso colosal cuando incrementamos n.

Imaginemos que tenemos 5 ciudades, entonces $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$, ahora dupliquemos el número de ciudades , entonces $10! = 10 \times 9 \cdots \times 1 = 3,628,800$, podemos ver a esta cantidad crecer

Recocido Simulado

El recocido simulado (simulated annealing) es un método probabilístico propuesto por Kirkpatrick, Gelett y Vecchi (1983) y Cerny (1985) para encontrar el mínimo global de una función de costo que puede poseer múltiples mínimos. Emulando el proceso físico por el cual un sólido es lentamente enfriado para que al llegar a un estado de congelación, este lo haya hecho con una configuración de energía mínima. [1]

Este método se basa en obtener vecinos randomizados, donde se define una temperatura que decrementa lentamente, lo que nos da una cota superior que permite ”subidas” que incrementan el costo, esperando que estas ”subidas” lleven a un mínimo local. Esta proceso nos da una funcioń que converge hacia la solución óptima.

Aceptación por umbrales

Diseño

El diseño de este programa usa el paradigma orientado a objetos, con una estructura bastante simple. Consiste de 5 clases structs

- `Path` este struct contiene funciones de todas las operaciones que puedes realizar sobre estas. Este struct contiene como atributo igual una instancia de todas las ciudades guardadas en un vector, así como
- `SimAnn`, como su nombre lo indica, este struct contiene los algoritmos del recocido simulado. Dentro de este, se encuentra la clase path como atributo
- `TSI` este struct se encarga de generar hilos con instancias de `SimAnn`, para correr varias semillas al mismo tiempo, lo que facilita mucho la experimentación, ya que correr grupos de cualquier cantidad sin necesidad de atención constante del programador.

Además de estas estructuras

Implementación

El lenguaje usado para este proyecto fue Rust. Una de las cuantas razones fue la rapidez y su confiabilidad. La rapidez es un factor muy influyente en la experimentación, ya que facilita la rápida obtención de resultados para poder mejorar los parámetros usados.

El algoritmo

Experimentación y resultados

La etapa de la experimentación

The higher the number of cities, a lower epsilon seems to work better. On the other hand for the case with 40 cities if the epsilo is 0.0001 then it seems to fall into a weird loop and the algorithm either doesn't end or take a long time to finish. So it's not reaching the minimum temperature for some reason.

Also increasing

Reproducir los resultados

Una parte importante y la razón por la que se usan semillas al generar los números aleatorios, es que los resultados encontrados sean reproducibles.

Esta es la lista de variables usadas para obtener estos resultados, y cómo reproducirlos.

Para el caso con 40 ciudades

Las variables usadas fueron

- Evaluación : 0.2637132755109184
- El camino:

$$[979, 493, 329, 163, 172, 496, 815, 657, 168, 1, 656, 2, 653, 490, 654, 7, 816, 982, 332, 820, 981, 333, 3, 165, 6, 5, 978, 817, 4, 489, 492, 491, 984, 331, 164, 327, 980, 186, 483, 54]$$

- Epsilon : 0.002
- Phi : 0.98
- Tamaño de lote : 1000
- Semilla de vecinos aleatorios : 16214040050592208640
- Semilla para generar la solución inicial : 16042267250931732492

Para poder correr esta instancia, ejecutar lo siguiente desde la línea de comandos.

```
cd target/release
./simulated_annealing --cities40 -e 0.002 --neigh 16214040050592208640
                      --init 16042267250931732492
```

Para el caso con 150 ciudades

- Evaluación : 0.14418250861220022196
- El camino:

  $[54, 652, 1075, 483, 171, 77, 183, 346, 75, 821, 512, 179, 671, 16, 520, 186, 190, 675, 340, 502, 151, 828, 12, 1038, 339, 826, 444, 17, 164, 11, 501, 25, 492, 491, 499, 347, 489, 4, 174, 817, 23, 176, 668, 352, 978, 5, 6, 988, 165, 3, 981, 990, 333, 991, 351, 185, 22, 676, 665, 173, 2, 656, 184, 815, 172, 182, 496, 19, 505, 168, 508, 986, 9, 1, 829, 657, 663, 661, 832, 667, 507, 653, 344, 490, 654, 26, 820, 345, 332, 181, 14, 982, 187, 816, 678, 823, 7, 673, 163, 329, 509, 493, 979, 837, 995, 984, 1003, 349, 331, 662, 8, 999, 674, 334, 343, 510, 660, 20, 680, 825, 500, 985, 504, 511, 327, 670, 350, 840, 336, 297, 980, 191, 822, 1001, 74, 166, 658, 666, 818, 655, 819, 330, 1073, 169, 1037, 326, 328, 167, 495, 494]$

- Epsilon : 0.0001
- Phi : 0.98
- Tamaño de lote : 1000
- Semilla de vecinos aleatorios : 11613177925935108993
- Semilla para generar la solución inicial : 8480295797754924014

Primero se debe de cambiar la siguiete línea de código dentro del archivo `sa.rs`

```
105 let mut batch_average =
```

después, ejecutar lo siguiente desde la línea de comandos.

```
cd target/release
./simulated_annealing --cities150 --neigh 11613177925935108993
                      --init 8480295797754924014
```

Conclusiones

References

[1] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *MIT Statistical Science*, 8, 1993.