Recocido Simulado

Emilio Martínez Hernández

1 Introducción

Existen problemas de optimazación que son intratables en la práctica, uno de los más populares es el Problema del Agente Viajero (TSP), que es un problema NP-completo, para este proyecto usaremos una variante del TSP en la que nos limitaremos a buscar sólo una trayectoria Hamiltoniana y usaremos la meta-heurística del recocido simulado para minimizar su peso tanto cómo sea posible.

2 Planteamiento

Sea $G = (V, E)$ una gráfica y $S \subset V$ un cjto. de vertices, el objetivo es encontrar una trayectoria $v_1v_2...v_{|S|}$ tal que $v_i \in S \wedge i \neq j \implies v_i \neq v_j$ que sea de peso mínimo.
El mejor algoritmo conocido para este problema revisa todas las trayectorias existentes y selecciona la de menor peso, pero al haber $O(|S|!)$ trayectorias, esto no es viable en la práctica, es por ello que para encontrar buenas soluciones es necesario utilizar métodos heurísticos, que aunque no garantizan la obtención de la mejor solución, resultan en soluciones bastante aceptables.

3 El recocido

El recocido es un método usado en la antigüedad en la que los metales eran enfriados lentamente para que los cristales que contenían se acomodaran en una configuración de mínima energía.
De esta manera el recocido simulado busca minimizar la función de costo de un problema mediante la simulación de un enfriado lento, en el que las soluciones al problema representan dichos cristales.

Para este proyecto se usará una variante del recocido simulado, llamado aceptación por umbrales, que consiste en lo siguiente:

- Un conjunto finito de soluciones S.
- Una función $f : S \to \mathbb{R}$ llamada función de costo.
- Una función $N : S \to \mathbb{P}(S)$ llamada función de vecindad.

- Un parámetro de control $T \in \mathbb{R}$, llamado Temperatura inicial.
- Un parámetro de control $\epsilon \in \mathbb{R}$, llamado Temperatura mínima.
- Un parámetro de control $\phi \in \mathbb{R}$, llamado Factor de enfriamiento.
- Un parámetro de control $L \in \mathbb{R}$, llamado Tamaño de lote.
- Una solución inicial $s_0 \in \S$.

Dados los anteriores parámetro el procedimiento es el siguiente:

Algorithm 1 Aceptación por Umbrales

Require: T, s
1: $p \leftarrow 0$
2: **while** $T > \epsilon$ **do**
3: $q \leftarrow \infty$
4: **while** $p \leq q$ **do**
5: $q \leftarrow p$
6: $p, s \leftarrow CalculaLote\ T\ s$
7: **end while**
8: $T = \phi T$
9: **end while**

Algorithm 2 Calcula Lote

Require: T, s
1: $r, c \leftarrow 0, 0.0$
2: **while** $c < L$ **do**
3: $s' \overset{R}{\leftarrow} N(s)$
4: **if** $f(s') < f(s) + T$ **then**
5: $s \leftarrow s'$
6: $c \leftarrow c + 1$
7: $r \leftarrow r + f(s')$
8: **end if**
9: **end while**
10: **return** $r/L,\ s$

En dónde se aplica Aceptación por Umbrales con parámetros T y s_0.

Cabe mencionar que en Calcula lote debe añadirse una condición arbitraria de término, pues es posible que nunca acepte a L soluciones. Además es importante llevar un control de la mejor solución encontrada durante la ejecución pues la mejor solución que se encuentre no necesariamente será la última.

4 Modelo

Para aplicar la heurística al problema, dados el conjunto de ciudades, sus conexiones (y costo de dichas conexiones) el conjunto S de ciudades a visitar, tomamos la subgráfica inducida por S, G_S y añadiremos aristas para para tener $K_{|S|}$ asignandole cómo costo a dichas conexiones del maximo costo de $G_S.E$ multiplicado por 3.
Esto se hace para facilitar la evaluación y la obtención de soluciones vecinas, manteniendo siempre en mente que una solución que no tenga sus aristas en G_S no deberá ser considerada "buena" pues realmente no es una solución al problema.

La función de costo que se usará es la siguiente:

$$f(s_1 s_2 ... s_n) = \frac{\sum_{i \in \{1..n-1\}} (w(s_i s_{i+1}))}{\frac{\sum_{e \in G_S.E} (w(e))}{|G_s.E|}(n-1)}$$

Esto se hace cómo un intento de acotar la función en el intervalo $[0,1]$, aunque no se cumpla precisamente esto, esta funcion suele tomar valores menores a 1, cuando se trata de soluciones factibles y mayor a 1 cuando se trata de soluciones no factibles.

La función de vecindad se definirá de la siguiente manera:

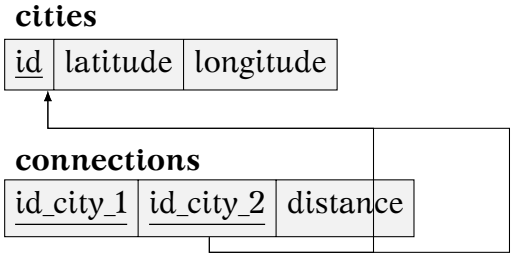
$$N(s_1 s_2 ... s_n) = \{a_1 a_2 ... a_n \mid \exists! a_i, a_j \ni a_i \neq n_i \wedge a_j \neq n_j\}$$

Es decir, ls permutaciones que sólo difieren por un intercambio.

5 Implementación

Para realizar la implementación se eligió el lenguaje de programación *Ocaml* debido a sus buenas cualidades de compilación, expresividad y facilidades hacia el paradigma imperativo.

Para la obtención de los datos se usa SQLite3 bajo el siguiente esquema:



Para variar los resultados del programa se varó la semilla del generador pseudo-aleatorio. Explorando así una mayor porción del espacio de búsqueda y aumentando la probabilidad de encontrar mejores resultados.

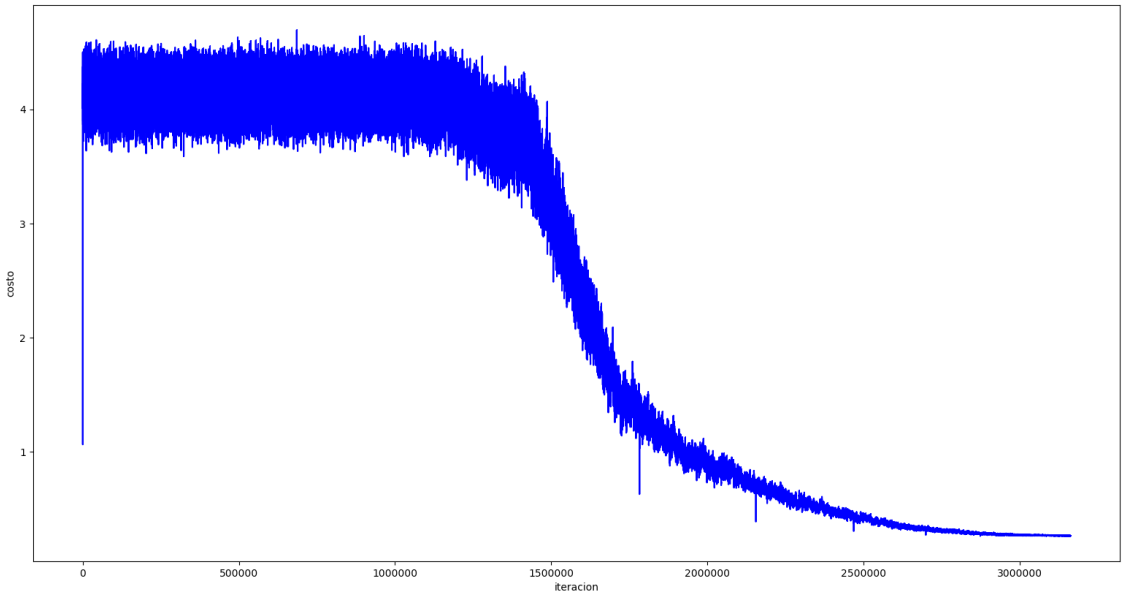
También se implemento un barrido sobre la vecindad de las mejores soluciones encontradas, para asegurar que se descienda hasta un mínimo local y no exista una

solución "ligeramente diferente" que sea mejor a la obtenida por el programa.

El código fuente puede ser encontrado en `https://github.com/emilio-mh/recocido_simulado`

6 Resultados

La implementación fue probada con instancia de 150 ciudades dentro de una gráfica de 1092 ciudades y 123403 conexiones en la cual se desempeñó de acuerdo a la siguiente gráfica.



En la fráfica se puede apreciar descensos verticales, que corresponden a la ejecución del barrido, además de contar con la forma característica de un recocido simulado que logró reducir significativamente el costo de la solución.

La secuencia obtenida fue:
795 748 571 69 470 67 40 735 96 896 492 614 1016 347 203 489 877 393 563 885 159 254 981 919 271 319 256 991 591 413 957 308 213 717 872 587 332 206 219 654 576 447 477 278 233 639 95 611 986 111 79 281 505 486 1069 98 355 688 358 528 893 766 935 257 837 284 705 323 796 741 890 871 670 414 1034 1080 853 697 876 757 297 438 359 1007 190 933 186 675 12 107 151 44 299 115 408 74 86 385 302 293 627 637 93 10 854 395 607 785 1002 189 690 156 557 341 1037 398 119 94 547 234 582 679 535 13 583 410 857 712 865 517 1072 313 700 454 552 955 833 1057 478 243 515 498 497 178 39 669 310 337 117 830

El costo obtenido fue: 0.262824843

Para llegar a este resultado se llevaron a cabo alrededor de 10,000 ejecuciones diferentes en las que la media estuvo alrededor de 0.28

7 Conclusión

Mediante este proyecto hemos logrado comprobar la efectividad de esta meta-heurística para optimizar el problema del agente viajero, lo cual nos da una idea de la efectividad que tiene en problemas combinatorios, sin embargo debemos recordar que no existe heurística que sea buena para todos los problemas (Non-Free-Lunch Theorem), por lo que la mayor conclusión es que si se elige bien una heurística acorde al problema los resultados pueden ser muy buenos.