Universidad Nacional Autónoma de México

Facultad de Ciencias

Proyecto 1:
Travelling Salesman Problem

Emiliano Galeana Araujo
314032324

Seminario de Ciencias de la computación B
Heurísticas de Optimización Combinatoria
Dr. Canek Peláez Valdés
L. en C.C. Víctor Zamora Gutiérrez

October 8, 2019

CONTENTS

LIST OF FIGURES

ABSTRACT

Se conocen muchos problemas de optimización. El objetivo de este proyecto es el *Travelling Salesman Problem*(TSP) pero con la particularidad de que no se busca un ciclo, sino un camino que pase por n ciudades sin repetir ninguna.
Dada una base de datos con ciudades, distancias entre ellas y conexiones entre ciudades. ¿Cuál es el mínimo costo que podemos encontrar para ejemplares de 40 y 150 ciudades? Se emplea la heurística de recocido simulado, la cual sirve para resolver problemas de optimización. Los resultados arrojadas por la heurística se cree son los óptimos, sin embargo no se pueden demostrar.

1 INTRODUCCIÓN

Dadas n ciudades $(1, ..., n)$ y una distancia no negativa entre dos ciudades i, j, en donde suponemos que $d_{i,j} = d_{j,i}$. Queremos encontrar el tour menos costoso. Una solución al problema es enumerar todas las posibles soluciones, y tomar la mejor de estas, el problema con esta propuesta para solucionar el problema es que el algoritmo tomaría $O(n)$ en tiempo (Hay $\frac{1}{2}(n-1)!$ tours a considerar). Lo cual claramente no es polinomial y por ende, tardaríamos mucho en encontrar la mejor solución para instancias grandes (aunque no tan grandes) del problema. Existen heurísticas que funcionan bien y pueden regresar tours que no están tan alejados del óptimo. En este proyecto, veremos la aceptación por umbrales.

2 LENGUAJE

Para la implementación de la heurística se eligió como lenguaje `Go(Golang)` el cual *es un lenguaje open source que hace más sencillo construir software de manera simple, confiable y eficiente. [1]* Lo elegí básicamente porque había escuchado que era bueno y no era muy complicado de usar y porque nunca había hecho algo en `Go` . La curva de aprendizaje fue complicada, pues empecé con algunas funciones en una carpeta hecha para el curso, en donde tenía al *main* ahí mismo. Al momento de pasar al main a otro archivo comenzaron los problemas y después de buscar en internet descubrí que tenía que migrar mis archivos a la carpeta que `Go` crea en el *home*.
Honestamente creí que lo más complicado sería poder conectar mi programa a la base de datos, pero fue muy fácil, hay muchos tutoriales en internet sobre `Go` que me ayudaron con eso y con otras cosas del lenguaje como las firmas de las funciones y lo mejor de todo, que me sirvió para algunas funciones de este proyecto. Que puedes tener funciones que regresan más de un tipo.
Los contras es que no podía copiar un array tan sencillo, que aún después de crear un array, podía seguir insertando, que fue complicada la diferencia entre `a = b` y `a := b`. Y obviamente el hecho de tener que estar en la carpeta generada en el *home*. Los pros fueron los tipos de regreso, que cuando importabas una biblioteca si no la usabas no te dejaba compilar el programa, igual que cuando creabas una variable y esta no era usada no te dejaba compilar, y que una vez que más o menos le agarras la onda ya no es tan complicado hacer cosas sencillas en el lenguaje.
Pero, también vienen cosas no tan buenas (En lo que respecta a mi), como el uso de los operadores * y *&*, que si hacía todo en el mismo archivo no me generaban tantos problemas, pero cuando acabé (todo en un archivo, como debe de ser) y quise pasarlo a los distintos archivos que tenía planeado, me mandaba muchos erroes como `cannot use cis (type *[]Ciudad) as type []Ciudad in argument to TotalAristas` y pues la verdad fue muy complicado arreglar eso, hasta que me harté e hice un pseudomain en `funciones.go`, que es casi lo mismo que tenía en el programa de un solo archivo, y ya solo desde el main (`main/main.go`), llamo al pseudomain (`funciones/funciones .go/NewTSP`) y parece que funciona bien.
La verdad creo que es un buen lenguaje y debería practicarlo más (demasiado), por que estuvo complicado.

3 PACKAGES

1. main
2. funciones
3. argumentos

3.1 Main

En este "package" lo que se implementó es la función main, y se importan los "package" *"fmt"*, *"funciones"*, *"argumentos"*, *"os"*, *"math/rand"* .

FUNCIÓN MAIN Es la función principal, se encarga de verificar la entrada esto quiere decir, que se le pasen los argumentos correspondientes. En la implementación actual tiene un ciclo para probar como semillas los primeros 1000 números naturales. La idea es quitar el ciclo una vez que encuentre la mejor solución, aunque es probale que eso tarde un poco.

3.2 Argumentos

En este "package" se encuentra el archivo "leer_ciudades.go".

FUNCIÓN LEER Es la función que recibe los argumentos pasados por el main. Una seed y un archivo con las ciudades que se buscan para el camino, este archivo tiene que ser de una línea y con los índices de las ciudades separados por coma (,). Los índices de las ciudades se pueden encontrar en la base de datos. La función regresa un `[]int` con los índices de las ciudades, regresa la semilla que es el segundo argumento en tipo `int`, y el nombre del archivo que se leyó, esto con la finalidad de escribir un archivo con los datos obtenidos y se tenga conocimiento de a qué resultados hacen referencia. Pero por el tiempo y por el ciclo, no uso (por ahora) el escribir un archivo.

3.3 Funciones

En este "package" se encuentran los archivos "funciones.go", "operaciones.go" y "sql.go", así como sus respectivos tests.

3.3.1 *funciones.go*

GETNORMALIZADOR Función que obtiene el normalizador como lo indica el PDF, es la suma de las k aristas más pesadas de nuestra instancia.

FUNCIONCOSTO Función que calcula la función de costo, o sea, la suma de la arista del vértice i con el $i+1$, esté o no esté en la gráfica, usamos la matriz que contiene toda la gráfica para esto.

VECINO Dado un `[]Ciudad`, que representa el acomodo de los índices de las ciudades, regresa un vecino de esta representación, eso es, el intercambio de dos índices aleatorios.

PORCENTAJEACEPTADOS Función que regresa el porcentaje de aceptados.

BUSQUEADBINARIA Función encargada de hacer búsqueda binara para encontrar una temperatura buena.

TEMPERATURAINICIAL Función que intenta encontrar una buena temperatura, inicia con temperatura de 8.

CALCULALOTE Función que calcula el porcentaje de aceptados en un lote , tiene un límite que es el tamaño de un lote al cuadrado. Creo que hago muchos copiar ciudades, así que eso se podría arreglar, al igual que crear nuevas soluciones.

ACEPTACIONPORUMBRALES Función que modela la aceptación por umbrales. Regresa una configuración de las ciudades, y su evaluación. Podría regresar una Solucion, en vez de regresar los datos individuales.

NEWTSP Como mencioné anteriormente, tuve problemas como *&,** , y este el pseudomain, se encarga de crear soluciones y los datos generales, para después conseguir una temperatura y finalmente hacer la aceptación por umbrales. Se puede arreglar, y demasiado, para que solo regrese un *TSP, y hacer los correspondientes (*NewGeneral, NewSolucion, NewCiudades*)pero por el momento creo que sirve.

3.3.2 *operaciones.go*

En este archivo se implementan todas las funciones que no tienen que ver tanto con la heurística o que como su nombre lo indica, son operaciones básicas.

COMPLETA Esta función recibe un arreglo de Ciudades y busca todas las aristas en la gráfica completa, hace uso de "sql.go", la idea es que después no tengamos que estar haciendo consultas, sino que todos los datos de las aristas, se encuentren aquí. Tuve problemas pues no llenaba bien la gráfica

RADIANES Cambia una coordenada (`float64`) en su representación en radianes (`float64`).

OBTENERA Regresa el resultado de la siguiente fórmula:

$$A = \sin(\frac{lat(v) - lat(u)}{2})^2 + \cos(lat(u)) \times \cos(lat(v)) \times \sin(\frac{lon(v) - lon(u)}{2})^2 \quad (1)$$

DISTANCIANATURAL Calcula la distancia en la vida real entre dos índices de, ciudades, no es la distancia exacta. Hace uso de la constante *radio*, definida en este archivo.

COPIARCIUDADES Copia un `[]int` y regresa la copia.

PRETTYPRINT Imprime un arreglo de ciudades de manera bonita (Solo los índices), para estructurar el programa, debería recibir una solución y usar su arreglo de ciudades. igual se podría usar una bandera para definir qué se quiere imprimir del arreglo (*nombre, país, población, etc*).

3.3.3 *sql.go*

En este archivo se hace la conexión con la base de datos, así como los queries necesarios durante la implementación de la heurística. Existen unas constantes que son el query, pero es para no escribir tanto después.

CIUDADES En esta función recibimos un arreglo de enteros, que contine a los índices de las ciudades para la instancia, y regresa un arreglo de tipo *Ciudad* que contiene todos los datos de la tabla cities para un índice, aunque no usamos algunos, creo que podemos regresar de manera más agradable las ciudade usadas.

TOTALARISTAS Esta función se encarga de regresar las aristas que existen en nuestra instancia, aparte las regresa ya ordenadas.

PESOAUMENTADO La función regresa la distancia (si es que existe en la base de datos), sino existe, regresa la distancia aumentada. Tuve algunos conflictos con esta, pues no había tomado en cuenta que en la base solo están las distancias de las aristas $i, jsyssi < j$, entonces la función actualiza al mayor.

4 RESULTADOS

La siguiente gráfica muestra el cómo van mejorando las soluciones aceptadas en la instancia de 40 ciudades. No es la mejor solución, pero es la mejor que he encontrado con aproximadamente 1300 semillas.

En cuanto tenga mis mejores soluciones, añadiré la gráfica.

5 CONCLUSIONES

Aún no llego a la mejor solución con 40 ciudades y no he probado mas que un par de veces la de 150. Tengo algunas configuraciones y con ellas algunas buenas soluciones, por ejemplo con *P=.95, EPSILON=0.0001, EPSILONP=0.001, L=1000, PHI=.90*, la mejor ejecución es la semilla *326* y devuelve una evaluación de *0.2642130044604 34*. Pero no es la mejor (la posible mejor), así que aún me falta hacer un poco de corridas o modificar mis parámetros . Corrí aproximadamente 1300 semillas y solo tres pasaron el 0.27, entonces, creo que faltan muchas más pruebas, quizá modificar los datos y seguir probando. Lo bueno es que tarda como 35 segundos en dar esa solución. No sé si go fue la mejor opción, desde primer semestre no me sentía tan malo en algo, pero estuvo divertido, más o menos, e interesante. Aún hay cosas que puedo hacer o mejorar y espero hacerlas.

REFERENCES

[1] Go. Go. https://golang.org/. Accesed: 2019-09-21.

[2] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesle Longman, 1994.

[3] C. Peláez. Heurísticas de optimización combinatoria. pages 3–13.

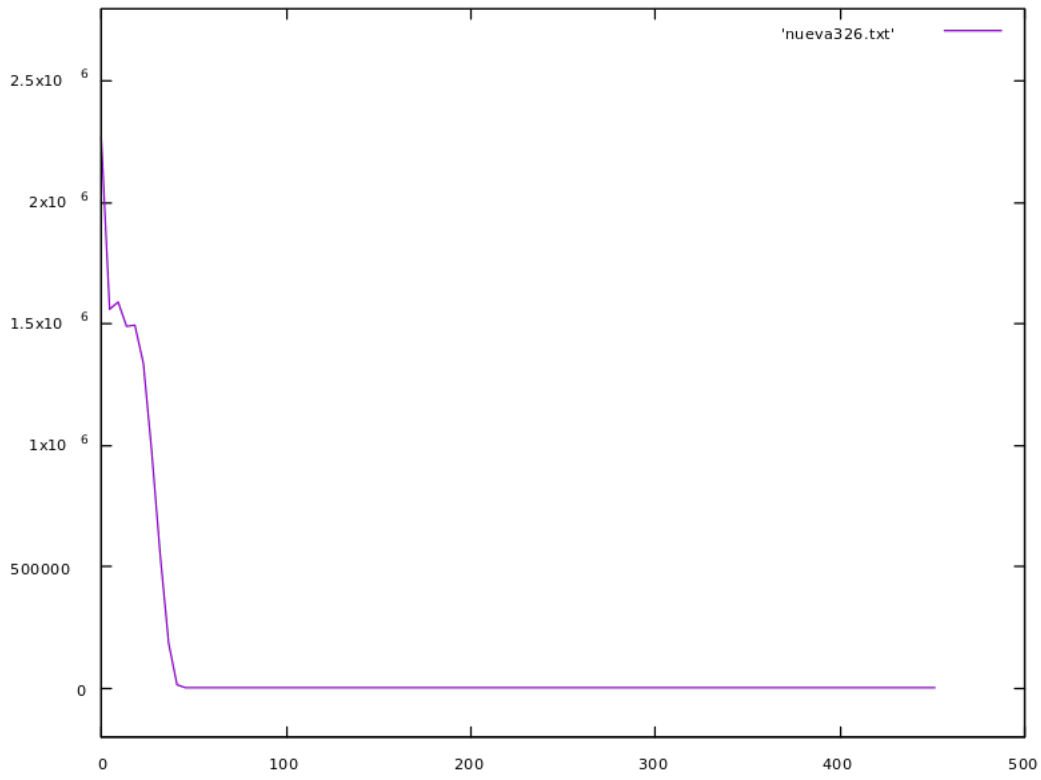

Figure 1: seed: 326