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

September 21, 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 INTRODUCTION

A statement requiring citation [1].

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. [2]* 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.

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"*, *"os"*, *"math/rand"*, *"funciones"*, *"argumentos"* .

FUNCIÓN MAIN Es la función principal, se encarga de verificar la entrada esto quiere decir, que se le pasen los argumentos correspondientes. También se crea un "struct" Ciudades y Solucion que son llamados de "funciones".

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.

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*

FUNCIÓN PRINTCIUDAD

FUNCIÓN PRINTDATA

FUNCIÓN TOTALARISTAS

FUNCIÓN GETNORMALIZADOR

FUNCIÓN FUNCOSTO

FUNCIÓN FUNCOSTOSOLUCION

FUNCIÓN PORCENTAJEACEPTADOS

FUNCIÓN BUSQUEDABINARIA

FUNCIÓN TEMPERATURAINICIAL

FUNCIÓN CALCULALOTE

FUNCIÓN ACEPTACIONPORUMBRALES

FUNCIÓN PRINTSOL

FUNCIÓN NEWSOLUCION

FUNCIÓN NEWCIUDADES

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.

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

FUNCIÓN DISTANCIANATURAL Calcula la distancia en la vida real entre dos índices de, ciudades, no es la distancia exacta. Hace uso del archivo "sql.go".

FUNCIÓN OBTENERA Regresa el resultado de la siguiente fórmula:

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

FUNCIÓN PESOAUMENTADO Dados dos índices de ciudades y la máxima distancia, regresa la distancia entre las dos ciduades multiplicada por la máxima distancia.

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

FUNCIÓN SWAP Dados dos índices, y un `[]int`, cambia los índices en una copia del `[]int`.

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

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.

OBTENERLATLON Es la función encargada de regresar una tupla de `float64` la tupla representa a la latitud y longitud de las ciudades en el mundo.

COMPLETA Esta función recibe un `[]int` el cual contiene a los índices de las ciudades y regresa un `[][]float64` el cual representa a todas las conexiones que se quieren para una instancia de nuestro problema.

4 RESULTADOS

REFERENCES

[1] A. J. Figueredo and P. S. A. Wolf. Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330, 2009.

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