El Problema de la Asignación Generalizado

Sandra del Mar Soto Corderi
No. cuenta: 315707267

20 de diciembre de 2020

1. Introducción

El problema del que se hablará en este reporte es el de **GAP**. Consiste en lo siguiente:

Dadas n tareas y m trabajadores o agentes, se quieren asignar todas las tareas a los trabajadores. Cada trabajador tiene cierta energía o $capacidad$ que se va gastando conforme se le asignan tareas, cada tarea gasta un valor de $capacidad$ dependiendo del trabajador al que se asigne, esto se puede ver como cuanta "dificultad"le puede costar al trabajador realizarla. Además cada tarea tiene un $costo$ que depende del trabajador que la vaya realizar.

La $capacidad$ total del trabajador, la capacidad que gasta una tarea y el $costo$ se pueden ver como las siguientes funciones:

$$capacidadTotal = b(w)$$
$$capacidad = a(t, w)$$
$$costo = \sum_{t=1}^{n} \sum_{w=1}^{m} c(t, w) \times x(t, w)$$

Donde w es un trabajador, t es una tarea y $x(t, w)$ es la función que representa si la tarea t está asignada al trabajador w, es decir:

$$x(t, w) = \begin{cases} 1 & \text{si } w \text{ realiza la tarea } t \\ 0 & \text{e.o.c} \end{cases}$$

Lo que busca optimizar el problema GAP es la función de costo. Además, una solución es factible si y sólo si se cumplen las siguientes condiciones:

1. Toda tarea debe estar asignada a algún trabajador.
$$\forall t \exists w (x(t, w) = 1)$$
2. Ninguna tarea debe estar asignada al mismo tiempo a dos o más trabajadores.
$$\forall t \neg \exists w_1, w_2 (x(t, w1) = 1 \ \wedge \ x(t, w_2) = 1)$$
3. Un trabajador no debe superar su $capacidadTotal$.
$$\forall w \left(\sum_{t=1}^{n} a(t, w) \times x(t, w) \ \leq b(j) \right)$$

1.1. Búsqueda Tabú *(Tabu Search)*

Ahora hablaremos de la heurística: La heurística que se uso fue el recocido simulado, la cual tiene su origen en la metalurgia, en el temple de metales. La idea de este es que si un metal se calienta a altas temperaturas y se le baja la temperatura abruptamente, el metal queda muy frágil, por otro lado si se baja la temperatura gradualmente, el metal se vuelve cada vez mas resistente. En pocas palabras, si se baja la temperatura lentamente, las propiedades del metal son mejores.

En este proyecto se uso la versión simplificada

2. Tecnologías usadas en el programa

- **Lenguaje de programación:** Kotlin 1.4.10.
  Este proyecto se inició en el lenguaje de programación Rust pero por problemas de configuración y por cuestiones de tiempo, se decidió cambiar a un lenguaje ya conocido como es Java, pero ya que el profesor no es muy aficionado a Java, se buscaron alternativas y la mejor opción encontrada en ese momento fue Kotlin. Kotlin tiene una sintaxis muy intuitiva y permite usar todas las bibliotecas de Java.
- **IDE:** IntelliJ IDEA
  Es la IDE más popular para Java y es compatible con Kotlin, por lo que se usó
- **Sistema de construcción:** Gradle 6.7
  Al investigar la documentación de Kotlin, se mencionaba a Gradle y a Maven como las mejores opciones para usar como sistema de construcción. Gradle tenía el tutorial más corto, así como manejaba las dependencias más fácilmente que Maven, por ello se escogió.
- **Documentación:** dokkaHtml 1.4.10.2
  Es el sistema de documentación oficial de Kotlin
- **Graficación:** Gnuplot 5.0 y Google Maps API
- **Insumos:** Los insumos (datos de entrada) fueron proporcionados por el profesor mediante una base de datos relacional *SQL*. El sistema manejador de la base de datos utilizado es SQLite 3.16.2. El controlador del *SMBD* es una biblioteca para Kotlin SQLite-JDBC 3.28.0.
- **Control de versiones:** Para mantener el control de versiones se utilizó Git 2.17.1 y el repositorio en línea se encuentra alojado en GitHub.

3. Diseño del programa

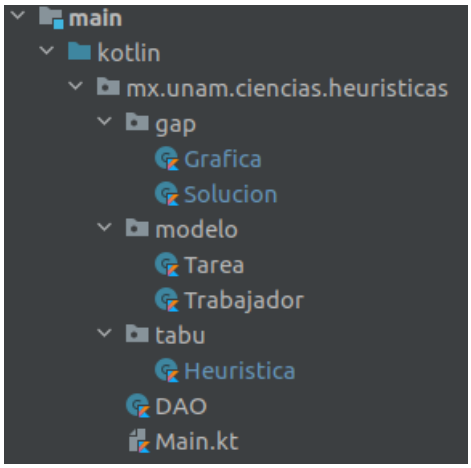

Figura 1: Estructura de las clases del proyecto

El proyecto se hizo con un enfoque orientado a objetos, por la naturaleza del lenguaje escogido.

Originalmente se iban a generar varias interfaces, pero al final se optó por un diseño más sencillo: El proyecto se dividió en tres paquetes:

- modelo
  En este paquete se crean los objetos que estaremos manejando en el proyecto y cuya información no cambiará, como son los valores de las ciudades y sus conexiones.
- tsp
  En este paquete tenemos todos los métodos o funciones necesarios para resolver el problema de tsp sin aplicar la heurística

- umbrales
  En este paquete tenemos los métodos o funciones necesarios para aplicar la heurística de recocido simulado con aceptación por umbrales al problema anterior.

A continuación se van a enlistar las clases usadas y lo que hace cada una:

- Ciudad.kt
  Esta clase almacena la información de las ciudades de nuestro problema, es decir, las coordenadas, nombre, id, población, de cada ciudad que se usará en el problema. Se declaró como una *data class* de Kotlin, ya que este tipo de clases se dedican únicamente a almacenar información, lo que hace que al construir el objeto, hayan varios métodos ya implementados. Igualmente es necesario comentar que no se incluyó la implementación de getters y setters debido a que en Kotlin en la documentación mencionan que no son necesarios.
- Conexion.kt
  Esta clase es análoga a la de Ciudad en la cuestión que es una *data class* donde guardamos las conexiones entre ciudades. Originalmente no se iba a usar este objeto, pero para facilitar el acceso a la base de datos con el DAO, se creó.
- Grafica.kt
  Esta clase es de las que cuentan con más métodos y es donde se realizan todas las funciones de la gráfica. Nuestra gráfica se representa como una matriz de adyacencia cuya forma de llenado sigue la definición de peso aumentado mencionado en la introducción. En esta clase se manejan los objetos de ciudad y conexion para obtener la información necesaria de acuerdo a la función. Cabe mencionar que para optimizar el tiempo en que corre el programa se agregó una función para obtener el peso de forma más directa, esta función optimiza la obtención del costo de la instancia una vez que se hace intercambio de índices al tomar las 4 aristas posibles de la gráfica que se ven afectadas con este intercambio y así obtener el nuevo costo restando los pesos de estas aristas eliminadas y la suma de agregar los nuevas aristas. Así obtenemos el peso durante las soluciones de forma constante y no lineal como sería en el caso de mandar a llamar a la función de costo de nuevo en cada caso.
- Solucion.kt
  Esta clase es bastante sencilla, ya que solo crea soluciones para el tsp, es decir invierte aleatoriamiente vecinos de la ruta para crear nuevas con la esperanza de que mejore. Esta clase originalmente iba a ir integrada con la clase de Grafica, pero era más limpio y fácil de comprender manejar los métodos de la clase heuristica con un objeto sencillo Solucion que con el objeto de una clase llena de métodos como es Grafica.
- Heuristica.kt
  Esta podría considerarse como la clase más ilustrativa del objetivo de este proyecto ya que es la que realiza el procedimiento de la heurística de recocido simulado con aceptación por umbrales. En pocas palabras el procedimiento que se sigue es obtener una temperatura con un valor lo suficientemente grande como para recorrer el espacio de búsqueda de forma rápida, esta temperatura se va disminuyendo al multiplicarla con un factor de congelamiento y mientras se enfría se van generando soluciones nuevas, de las cuales se toman las mejores. Los métodos de esta clase son la implementación los algoritmos que vienen descritos en el pdf llamado hoc que se encuentra dentro de la carpeta fuentes en este mismo directorio. Se había implementado un método que optimizaba los resultados, este método revisaba todos los vecinos de la mejor solución y tomaba el mejor de ellos, pero alentaba tanto al sistema que no se podían hacer las pruebas con muchas semillas.
- DAO.kt
  Esta clase funciona como nuestro Data Access Object, es la clase que se conecta con la base de datos directamente. Usamos un DAO por nuestro diseño basado en orientación a objetos y para mantener el patrón Modelo-Controlador. Normalmente los DAO no se aconsejan para aplicaciones donde el tiempo de ejecución importa pero es mucho más eficiente tener que hacer una o dos consultas a la base de datos mediante el DAO que hacer varias consultas dentro del modelo.
- Main.kt
  Este archivo no es una clase como tal, sino es el main donde se ejecuta el sistema. Lo que hace es tomar la lista de ciudades dada por el usuario con las cuales crea un objeto gráfica, este objeto gráfica manda a crear soluciones usando el rango de semillas proporcionado y manda a llamar a la heurística para mejorar la solución en los parámetros que se den. Al final se imprimen los costos de todas las mejores soluciones y se regresa la ruta de la mejor solución de todas.

Para ver más detalladamente la implementación, favor de generar la documentación con dokkahtml.

4. Resultados

Probando diferentes semillas de un lote de 1000 semillas para ambas instancias de TSP (la de 40 y 150 ciudades), se logró encontrar una semilla para cada ejemplar de forma que se obtuvieran soluciones factibles y se tomó la mejor solución de todas. Para el caso de 40 ciudades el programa tuvo que ejecutarse por un aproximado de 5 horas mientras que para el caso de 150 tuvo que ejecutarse 16 horas.

El sistema tiene la cualidad de que cada vez que lo ejecutas aunque sea con la misma semilla, no te dará la misma función de costo, esto es por como se implementó la aleatoriedad en las soluciones dentro del programa. Por esa razón siempre se devuelve la ruta de la mejor solución, para que no se pierda.

Los parámetros usados en el sistema de la heurística fueron cambiando de acuerdo a pruebas hechas con pocas semillas y las recomendaciones del profesor y la clase, después de encontrar mejores soluciones satisfactorias, los parámetros quedaron de la siguiente forma:

- Epsilon usada en la heurística de aceptación por umbrales 0.00001
- Epsilon usada en el algoritmo de obtención de la temperatura inicial 0.00001
- Temperatura inicial del sistema 8.0
- Límite superior para las iteraciones al calcula un lote L = 2000
- Factor de enfriamiento que determinará que tan rápido descenderá la temperatura T 0.95
- Porcentaje de soluciones vecinas 0.9
- Valor de vecinos aceptados usados para calcular la temperatura inicial. 2000
- Número máximo de iteraciones al calcular un lote L * 21

4.1. Instancia con 40 ciudades

Para la intancia de 40 ciudades tomaron los siguientes ids:

$[1, 2, 3, 4, 5, 6, 7, 75, 163, 164, 165, 168, 172, 327, 329, 331, 332, 333, 489, 490, 491, 492, 493, 496, 652, 653, 654, 656, 657, 792, 815, 816, 817, 820, 978, 979, 980, 981, 982, 984]$

Y se obtuvo un costo inicial de 3305585.454990047.

Después de ejecutar la heurística con 1000 semillas diferentes se obtuvieron soluciones mucho mejores, de las cuales se escogió la solución que generaba el menor costo, esta solución fue la generada por la semilla 635.

Si se desea ver los costos que generaron las otras semillas durante esta ejecución, se pueden ver en el txt 40-1000-semillas.txt que se encuentra dentro del directorio resultado/resultadoIterandoSemillas

Al ejecutar el programa con 1000 semillas distintas vemos que en todos los casos se generaron soluciones factibles que oscilaban en el rango de 0.21-0.29 como valor de la función de costo

Usando la semilla **635** nos da los siguientes resultados:

Ruta: $[980, 327, 871, 331, 164, 984, 491, 492, 489, 4, 817, 978, 5, 6, 165, 3, 333, 981, 820, 332, 982, 816, 823, 7, 654, 490, 653, 656, 2, 661, 657, 168, 1, 815, 496, 172, 163, 329, 493, 979]$

Costo: 0.21751778855705842

Así mismo se ejecutó el sistema de nuevo con la mejor semilla pero se guardaron los valores de sus evaluaciones y de sus mejores soluciones durante la ejecución, para poder graficarlas y compararlas. Podemos ver que el cambio en las soluciones tiene una mejor logística ya que en un punto baja rápidamente de valor y empiece a disminuir poco hasta que se acaba la temperatura.

4.2. Instancia con 150 ciudades

Con los identificadores de las ciudades:

$[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 16, 17, 19, 20, 22, 23, 25, 26, 27, 74, 75, 151, 163, 164, 165, 166, 167, 168, 169, 171, 172, 173, 174, 176, 179, 181, 182, 183, 184, 185, 186, 187, 297, 326, 327, 328, 329, 330, 331, 332, 333, 334, 336, 339, 340, 343, 344, 345, 346, 347, 349, 350, 351, 352, 353, 444, 483, 489, 490, 491, 492, 493, 494, 495, 496, 499, 500, 501, 502, 504, 505, 507, 508, 509, 510, 511, 512, 520, 652, 653, 654, 655, 656, 657, 658, 660, 661, 662, 663, 665, 666, 667, 668, 670, 671, 673, 674, 675, 676, 678, 815, 816, 817, 818,$

819, 820, 821, 822, 823, 825, 826, 828, 829, 832, 837, 839, 840, 871, 978, 979, 980, 981, 982, 984, 985, 986, 988, 990, 991, 995, 999, 1001, 1003, 1004, 1037, 1038, 1073, 1075]

Se obtuvo un costo inicial de 6152051.625245281

Después de ejecutar la heurística con 1000 semillas diferentes se obtuvieron soluciones con costos mucho mejores, de las cuales se escogió la solución que generaba el menor costo, esta solución fue la generada por la semilla 727.

Si se desea ver los costos que generaron las otras semillas durante esta ejecución, se pueden ver en el txt 150-1000-semillas.txt que se encuentra dentro del directorio resultado/resultadoIterandoSemillas

Al ejecutar el programa con 1000 semillas distintas vemos que no siempre se generaban soluciones factibles, pero se puede confirmar que más de la mitad fueron soluciones factibles. Esto se discutió en el seminario era normal debido a la cantidad de ciudades de la instancia.

Usando la semilla **727** se obtuvieron los siguientes resultados:

Ruta: [353, 990, 27, 333, 981, 3, 165, 988, 174, 4, 489, 817, 176, 668, 23, 352, 978, 6, 5, 1004, 351, 991, 185, 22, 490, 653, 507, 2, 656, 667, 184, 815, 9, 832, 661, 663, 657, 829, 1, 986, 508, 168, 505, 19, 839, 496, 182, 172, 173, 673, 665, 344, 654, 7, 823, 678, 816, 187, 982, 14, 181, 345, 332, 26, 820, 676, 163, 329, 509, 493, 979, 837, 995, 347, 499, 491, 492, 25, 501, 984, 1003, 349, 331, 662, 8, 999, 674, 871, 343, 510, 660, 20, 825, 500, 985, 670, 350, 511, 327, 504, 334, 164, 11, 17, 444, 826, 840, 336, 980, 297, 828, 12, 1038, 502, 340, 183, 75, 346, 171, 483, 652, 1075, 821, 512, 179, 671, 16, 520, 186, 675, 339, 151, 822, 1001, 74, 166, 658, 666, 818, 655, 819, 330, 1073, 169, 1037, 326, 328, 495, 167, 494]

Costo: 0.14676221976439466

Así mismo se ejecutó el sistema de nuevo con la mejor semilla pero se guardaron los valores de sus evaluaciones y de sus mejores soluciones durante la ejecución, para poder graficarlas y compararlas. Podemos ver que el cambio en las soluciones tiene una mejor logística ya que en un punto baja rápidamente de valor y empiece a disminuir poco hasta que se acaba la temperatura.

5. Conclusiones y Comentarios

En conclusión, se hizo un proyecto que cumplía el objetivo y lo hizo de un forma bastante buena.

El tener que cambiar de un lenguaje a otro e iniciar desde cero el proyecto me atrasó bastante, lo que causó no pudiera refinar el diseño y la velocidad del sistema como quería. Además tuve varios problemas para usar gradle, lo que hacía que no pudiera probar lo que iba implementando. Igualmente tuve problemas para entender como funcionaba la heurística. A pesar de eso estoy satisfecha con mis resultados, mis mejores soluciones se acercaron bastantes a las del profesor y mi sistema no es tan lento como esperaba, se tardo demasiado con mis pruebas para las 1000 semillas, pero al ser tantas semillas no lo veo como algo malo.

En este proyecto estuve muchas veces en la encrucijada de si es mejor tener un sistema rápido o un sistema eficiente, en otro momento respondería siempre que es mejor un sistema eficiente, porque al final lo que queremos son buenas respuestas, pero cuando implementé una última optimización (hill descending) que hacía a mi sistema el triple de lento, me di cuenta que ganaría más tiempo probando diferentes parámetros con el sistema viejo que con este optimizado. Y así fue, obtuve casi las mismas soluciones probando con variaciones de la temperatura del sistema sin hill descending que esperando mucho más con la última optimización implementada. Por lo que opté no usar hill descending en la versión final.

Este proyecto me dio una mejor perspectiva para los próximos proyectos, los cuales haré con una mejor planeación. Me gustaría poder hacerlos en un lenguaje como Nim, Rust o C++ para que sean mucho más rápidos, aún así Kotlin fue bastante amigable y no me molestaría usarlo de nuevo.

Fue muy divertido ver los cambios tan radicales de las soluciones cuando se implementó la heurística.

Referencias

[1] Día Juan A., Fernández E. A Tabu search heuristic for the generalized assignment problem. European Journal of Operational Research. 2001;132:22–38.

[2] Glover F, Laguna M. Tabu search. S.L.: Kluwer Academic; 1993.

Todas las fuentes que no son links se encuentran dentro del directorio fuentes ubicado en el directorio donde se encuentra este reporte.