Trabajo Nº 3
“Problem Solving”

Sistemas Inteligentes

Sergio Troncoso Duque
08/05/2017

Introducción

El problema del vendedor viajero es un famoso problema (TSP, por sus siglas en inglés Travelling Salesmen Problem) de computación que responde a la siguiente pregunta ¿Cuál es la ruta mas corta que puede recorrer un viajero al pasar por todas las ciudades una vez y devolverse al origen? El problema es parte del conjunto de problemas NP-Completo, es decir que no tiene una solución polinomial. Este problema se puede extrapolar a diversas disciplinas como la planificación, la logística, la fabricación de circuitos electrónicos o como un sub-problema en la decodificación de la secuencia de ADN.

Al ser este un problema complejo su solución crece en complejidad a medida que va creciendo el problema, es decir, mientras más ciudades tenga el problema, más tiempo nos llevara encontrar una solución al problema. Por ende el gran problema es encontrar tanto una solución óptima para el problema como para el costo computacional de este. Para esto son varios los algoritmos que eligen distintos enfoques para solucionar el problema del vendedor viajero, con distintos resultados.

Los algoritmos genéticos hacen evolucionar una población sometiéndolas a acciones aleatorias semejantes a las que actúan en la evolución genética, luego se van descartando según un parámetro los peores candidatos, dejando los más aptos. En este caso, las modificaciones son a las rutas a tomar, mientras que los candidatos son los distintos tour que pueden resultar. Por otro lado, el algoritmo de la colonia de hormigas (ACO, en inglés) que busca el camino más óptimo utilizando el sistema de feromonas que utilizan las hormigas para encontrar su hormiguero, para nuestro problema cada hormiguero es un ciudad, donde las hormigas viajan por los caminos desde uno a otro, guiándose por las feromonas. Además de los nombrados también existe algoritmo de colonias de abejas, basado en el comportamiento de cómo las abejas buscan miel en base a danzas que generan tres tipos de abejas: empleadas, en espera y exploradoras.

Cualquiera de estos algoritmos puede entregar una solución al problema de TSP, pero dependiendo de su implementación se reflejaran distintos resultados a nivel de resultados, complejidad de tiempo y de recursos. Pero entonces, ¿Cómo obtenemos el mejor resultado?, ¿Se puede llegar a la mejor solución conocida por algunos de estos algoritmos? y ¿Cómo podemos mejorar nuestra implementación para conseguir un mejor resultado?

Desarrollo del problema

Se debe implementar uno o más algoritmos para poder realizar problemas TSP de la biblioteca de problemas de la Universidad Heidelberg, que entrega un archivo .tsp con la información de los nodos y los caminos:

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

Se deben probar varias instancias del problema con distinta cantidad de ciudades y analizar el tiempo de ejecución, además de la cantidad de iteraciones.

También se deben probar diferentes parámetros según el algoritmo seleccionado, y analizar cuales son los que mejoran o empeoran el desempeño del cálculo de la solución.

Se tienen que analizar la(s) mejor(es) solución(es) con las soluciones óptimas asociadas que entrega la misma biblioteca:

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

Por último se debe plantear una mejora a la implementación realizada, para lograr acercarse o igualar la solución óptima.

Análisis de resultados

Implementación

Para la obtención de los datos se utilizara un algoritmo genético desarrollado en Python 2, la implementación de este cuenta con 4 archivos .py y un archivo de texto.

En el archivo de texto esta solo la información de los caminos extraída desde el archivo .tsp. Mientras que 3 de los archivos Python son objetos: el primero salesMan.py tiene 3 atributos que son el fitness, el camino (como una lista) y una probabilidad, además cuenta con 3 métodos y 4 setters y getters, luego esta el archivo Town.py que es el que almacena la información de cada ciudad desde el archivo de texto y finalmente el archivo costMatrix.py es la que ve la matriz de costos, tiene solo dos métodos: uno para ver las distancia entre puntos y otro para inicializarla.

Finalmente el archivo main.py contiene toda la lógica del programa, utilizando todas las demás clases para generar el procedimiento.

Al ser un código tan extenso el detalle de cada línea de código se puede ver en los comentarios en el código entregado con el informe.

Cantidad de iteraciones vs Tiempo de ejecución

Para la cantidad de iteraciones, se utilizo el problema kroA200 con distintas cantidades de iteración, a continuación podemos ver los resultados:

Iteraciones	Tiempo (s)	Tour Óptimo
1	0,416	305324
10	0,606	292518
50	1,314	270635
100	2,188	253969
500	8,866	257924
1000	17,358	287789

Tabla 1. Iteraciones vs Tiempo

Como podemos ver el mejor caso es con 100 iteraciones, con un tiempo promedio de 2,188 segundos lo cual es bastante razonable en tiempos de ejecución. Se puede apreciar que el tiempo va aumentando a medida que el número de iteraciones crece, esto se puede apreciar mejor en el siguiente gráfico:

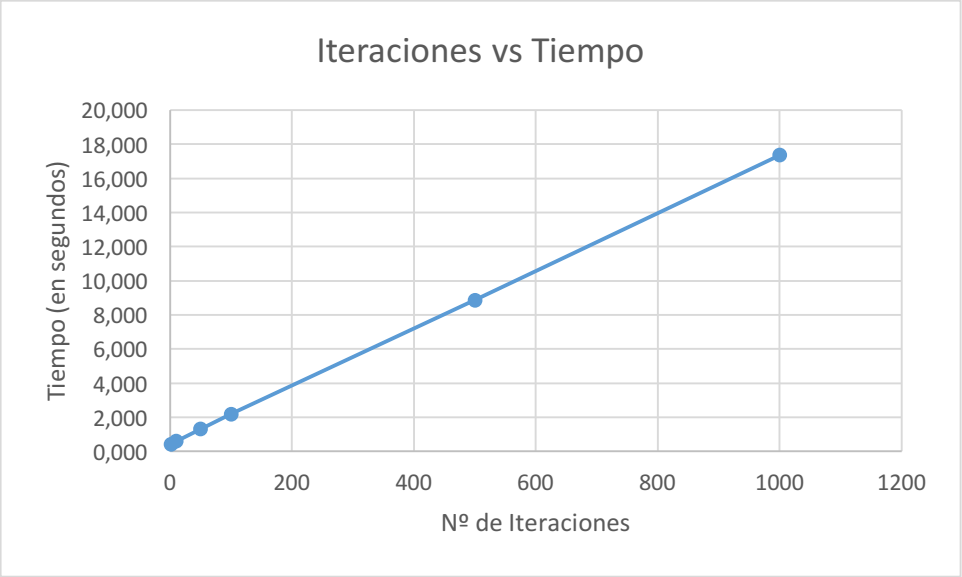

Gráfico 1. Iteración vs Tiempo

Variación de parámetros

El primero de los parámetros que se analizó fue el tamaño de la población (MAX_POPULATION en nuestro código) en el cual se aumento y disminuyo el valor inicial utilizado al principio. En la siguiente tabla se muestran los resultados:

Población	**20**	**10**	**50**	**100**
Tour Óptimo	**253969**	**266978**	**272580**	**260380**

Tabla 2. Variación de población

Obtenemos que el valor 20, el cual era nuestro valor por defecto es el que obtiene mejor desempeño, por ende lo dejamos para la siguiente variación.

Luego cambiamos el parámetro de mutación, obteniendo los siguientes resultados:

Mutación	**0.05**	**0.01**	**0.5**	**1**
Tour Óptimo	**253969**	**251011**	**292752**	**205782**

Tabla 3. Variación de mutación

El valor por defecto que se estaba usando era 0,05, pero como se puede ver si se disminuye el valor se obtiene un mejor resultado, por ende, nos quedamos con este valor.

El último parámetro es el de cuantas generaciones evoluciona, el cual por defecto teníamos con el valor de 0.8. En la tabla, los datos de la variación:

Generaciones	**0.8**	**0.5**	**1.3**	**2.1**
Tour Óptimo	**253969**	**268823**	**228259**	**257627**

Tarea 4. Variación de generaciones

El valor que se obtienen mejor resultados es al aumentar el valor, en un rango del 1 al 2. Por ende nos quedamos con este valor de variable.

Solución obtenida vs Solución optimizada

En la librería que se obtuvieron los problemas de TSP, también podemos obtener la solución optimizada para cada problema, es decir, el mejor valor posible que puede tener el problema bajo cualquier algoritmo. El valor para el problema kroA200 es de 29368, mientras que el mejor obtenido por la implementación del algoritmo genético fue de 228259 lo cual nos dice que el algoritmo empleado esta muy lejos del óptimo alcanzable. Esto se puede haber dado por algunos errores en el código o por que la implementación solo del algoritmo no es suficiente para obtener el camino mas optimizado.

Mejorar la solución

Como se dijo anteriormente el desempeño de la solución puede variar de muchas formas con respecto al algoritmo que se elige y los parámetros que puede tener en ese algoritmo. Asumiendo que se llevo una correcta implementación del algoritmo, aún podemos obtener mejores soluciones si se agregan ciertas algoritmos a la implementación total.

Para nuestro problema del vendedor y además del algoritmo genético se pueden usar los siguientes optimizaciones al código:

- **2-opt/3opt**: La técnica consiste en seleccionar 2 o 3 cortes en el grafo solución y realizar cambios hasta que se encuentren mejoras. Estos cambios pueden ser de forma aleatoria o siguiendo alguna heurística.
- **Mejorar el algoritmo genético**: Agregar los conceptos de elitismo a la implementación del algoritmo puede ayudar a la obtención de mejores resultados.
- **Estrategia de población inicial**: Si se tiene una estrategia para elegir la población inicial (en el algoritmo se hace al azar), esta comprobado que se pueden obtener

Conclusiones

Después del diseño y la implementación del algoritmo genético para la obtención de los datos a analizar, se concluyó que

Además, se identificó que el número de iteraciones puede ayudar a mejorar la obtención de una mejor respuesta, pero también aumenta exponencialmente el tiempo de espera para obtenerla.

Con respecto a la variación de parámetros, se obtuvo que si al parámetro de población si se le daba un valor de 100, al de mutación uno de 0.01 y al de generaciones 1.3 para obtener la mejor solución posible, variando de diversa forma los parámetros.

La solución óptima obtenida a través de la biblioteca era mucho menor que la solución que se obtuvo a través del algoritmo, esto se debe haber dado por diversas razones en términos de implementación o porque la solución mostrada ahí contempla otras técnicas que pueden mejorar la solución final.

Algunas de los algoritmos que se pueden implementar para mejorar el algoritmo genético presentado aquí pueden ser aplicar 2/3-opt a la solución final, agregar otros conceptos al algoritmo genético como elitismo y tener una estrategia inicial para la población inicial.