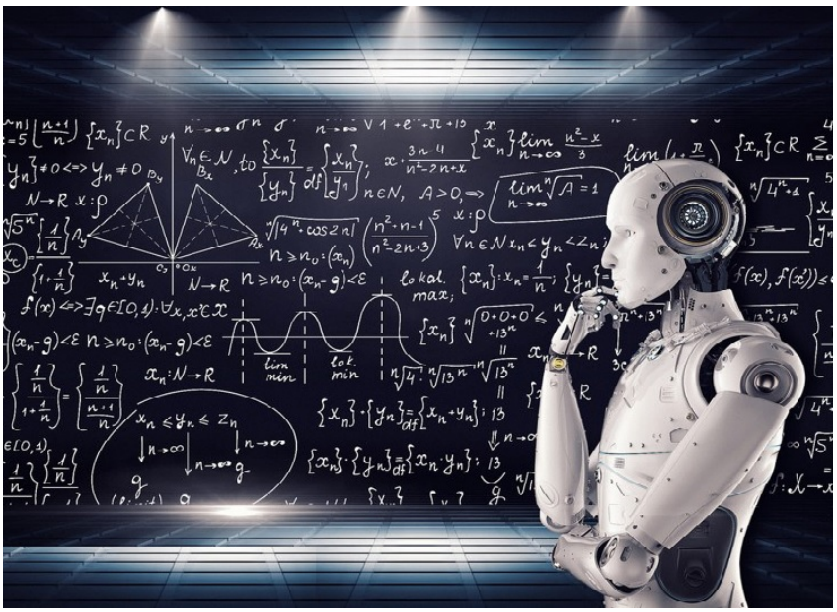
Tecnológico de Monterrey

TC3023

Inteligencia Computacional

Grupo 01



Proyecto final

Traveling Salesman Problem usando Algoritmos Genéticos

Jesús Abraham Haros Madrid	A01252642	a01252642@itesm.mx
Mario Emilio Jiménez Vizcaíno	A01173359	A01173359@itesm.mx

Profesor: Gabriel Gonzalez Sahagún

Lunes 29 de noviembre del 2021
Monterrey, Nuevo León, México

Índice

Descripción del problema

El problema del vendedor viajero (o mejor conocido como TSP) es un problema famoso por ser muy fácil de describir y difícil de resolver [1]. El problema puede plantearse de forma sencilla: si un vendedor ambulante desea visitar exactamente una vez cada una de una lista de ciudades y luego regresar a la ciudad inicial, ¿cuál es la ruta más corta que el vendedor puede tomar?

Obviamente existen múltiples rutas y caminos por elegir, pero encontrar el camino más eficiente es una tarea difícil de diseñar y con mucha complejidad en tiempo. De hecho, TSP pertenece a la clase de problemas de optimización que se conoce como ***NP-completo*** debido a que la complejidad de resolverlo aumenta demasiado cada que se añaden nuevas ciudades al problema.

¿Para qué sirve resolver TSP en la vida real?

Existen problemas que, aunque no se tratan de un vendedor ambulante son muy similares, por ejemplo, empresas de envíos que requieren planear la ruta de entregas del día, un robot en una bodega que se encarga de visitar diferentes puntos. Una aerolínea tratando de diseñar la mejor ruta para conectar escalas de aviones, entre muchas otras.

Ejemplo de TSP; dadas *n* ciudades, encontrar la ruta más eficiente.

En las ciencias computacionales una metaheurística es un procedimiento diseñado para encontrar, generar o seleccionar una heurística que ayude encontrar una solución suficientemente aceptable para un problema de optimización.
Los algoritmos genéticos son una metaheurística inspirada en el proceso de selección natural, son altamente usados para generar soluciones a problemas de búsqueda y optimización a través del uso de mutación, combinación y selección.

Este proyecto consiste en implementar algoritmos genéticos para resolver el problema del viajero, adicionalmente se compararán diferentes métodos de crossover, mutación y algoritmos genéticos para poder observar el comportamiento en la resolución de problemas al utilizar diferentes parámetros.

Trabajo relacionado

Genetic algorithms for the travelling salesman problem: A review of representations and operators

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Artificial intelligence review, 13(2), 129-170.

Los autores nos muestran el problema del vendedor viajero como sujeto de pruebas a diferentes métodos de cruce y métodos de mutación, este trabajo es relevante para el proyecto debido a que se requiere comprensión de los métodos de mutación y cruce para poder compararlos y hacer conclusiones de sus rendimientos.
Originalmente se pensaba crear nuestras propias funciones de mutación y cruce pero debido a que la librería DEAP ya implementaba muchos de los métodos narrados en este artículo decidimos usarlos.

Genetic algorithm performance with different selection strategies in solving TSP

Razali, N. M., & Geraghty, J. (2011, July). Genetic algorithm performance with different selection strategies in solving TSP. In Proceedings of the world congress on engineering (Vol. 2, No. 1, pp. 1-6). Hong Kong: International Association of Engineers.

En este artículo se puede ver como los autores utilizan el problema del viajero para aplicar algoritmos genéticos y probar distintos métodos de selección, esto con la finalidad de ver las diferencias cuando se resuelve TSP con un método a otro.
Este trabajo es muy relevante para nuestro proyecto ya que se generará algo parecido a su trabajo, comparando diferentes métodos de mutación, cruce y selección.

Descripción de la representación utilizada para el problema

Para este trabajo primero tenemos que definir lo que representa una ciudad, una ciudad esta una coordenada compuesta de X y Y.

```
firstCity = [12.2, -4.35]
secondCity = [-0.5, 8.2]
cities = [firstCity, secondCity, ... ]
```

Un camino es una lista ordenada de ciudades donde la primera posición representa la primera ciudad a visitar mientras que la última es la ciudad terminal.

```
path = [4, 5, 3, 1, 8, 0, 2]
```

Dicho camino se evaluará utilizando la distancia euclidiana entre ciudades, para el camino mostrado el resultado sería la suma de las distancias entre las ciudades adyacentes volviendo al inicio.

```
totalDistance = distance(cities[4], cities[5]) + distance(cities[5],
cities[3]) ... distance(cities[2], cities[4])
```

Todo lo necesario para ejecutar el algoritmo se encuentra definido dentro de la clase TravelingSalesmanProblem: calcular las distancias, generar soluciones, iterar, ejecutar, calcular, y graficar.

Se utilizará la librería DEAP para implementar algoritmos genéticos [3].

Descripción de los resultados obtenidos y gráficas/tablas con los resultados

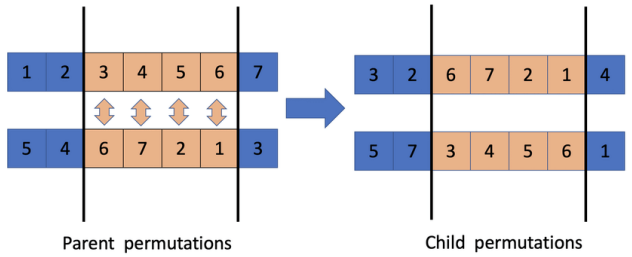
Para la demostración en este proyecto se utilizaron 3 diferentes algoritmos genéticos, comúnmente denominados “simple”, “mu+lambda” y “mu,lambda”, que difieren en la forma en la que se escoge la siguiente generación de individuos a partir de la generación actual: el primero modifica a los individuos sin importar si tuvieron una evaluación buena o mala, en el segundo los individuos de la generación pasada pueden ser mutados o “heredados” a la siguiente, y en la tercera se generan nuevos individuos a partir de los mejores padres.

También se utilizaron dos diferentes tipos de algoritmos de mutación:

- El primero, implementado por la librería deap, mutShuffleIndexes, tiene la capacidad de intercambiar partes de un individuo (en nuestro caso, caminos) para así generar caminos mutados. Esta función recibe un parámetro, indpb, que describe qué probabilidad hay de que cada parte cambie de lugar. Para este proyecto este parámetro fue establecido como 0.05
- El segundo, displacement mutation, selecciona una parte del camino aleatoriamente, lo remueve y lo inserta en un lugar aleatorio. Este algoritmo fue implementado en el archivo main.py, con el nombre de función mutDisplacement.

Por otro lado también se utilizaron dos algoritmos de crossover (a partir de dos padres, generar dos hijos con propiedades de ambos padres):

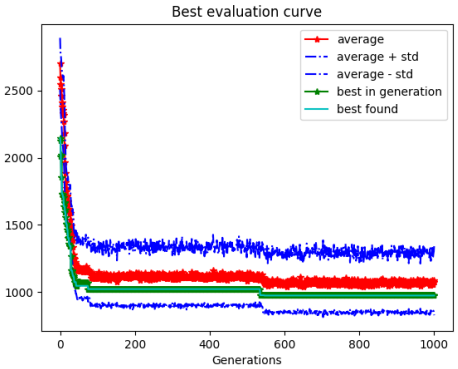
- **cxPartiallyMatched**: Este método de cruce consiste en generar dos hijos a través del intercambio de un rango de valores de los dos padres.



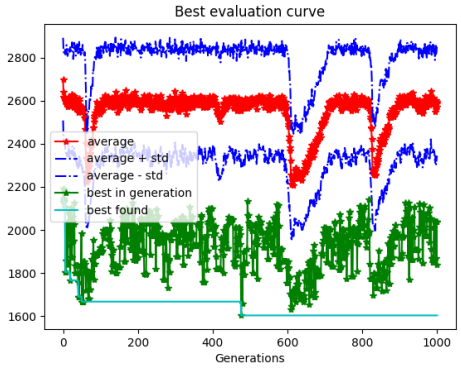
- También implementamos el algoritmo cxVotingRecombination, muy parecido al anterior pero los intercambios sólo ocurren en posiciones en donde los cromosomas tienen valores diferentes, y si un valor aleatorio entre 0 y 1 es mayor a 0.5

Ejecutamos el algoritmo por 1000 generaciones, cada una de 300 individuos, y 25 ciudades aleatorias (predefinidas e iguales para todas las ejecuciones), utilizando las 12

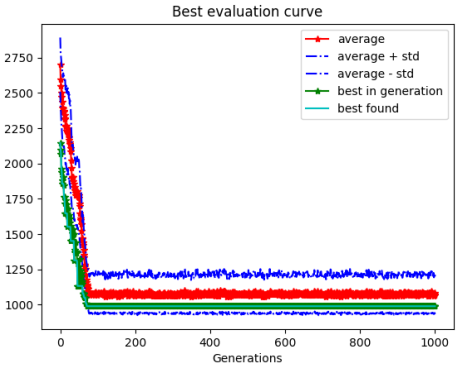
combinaciones de algoritmo, mutación y crossover, y graficamos la curva de mejor individuo encontrado para cada combinación:



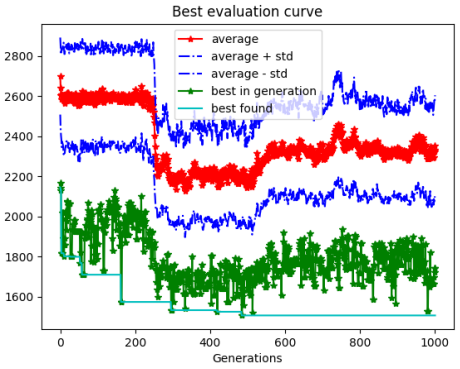
Simple, mutShuffleIndexes, cxPartialyMatched



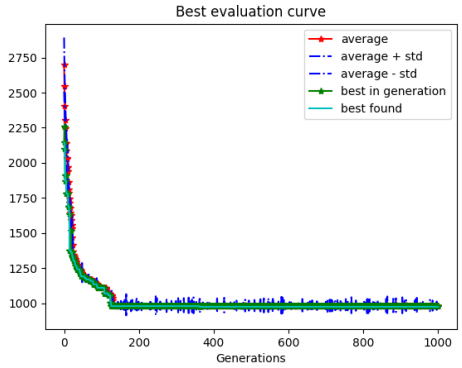
Simple, mutShuffleIndexes, cxVotingRecombination



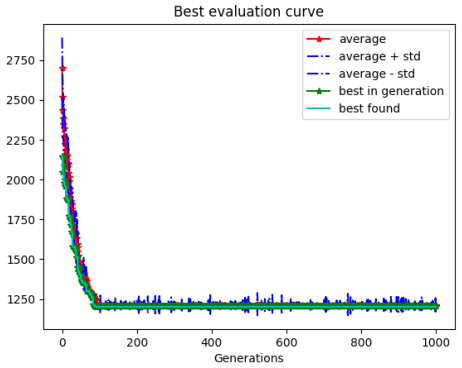
Simple, mutDisplacement, cxPartialyMatched



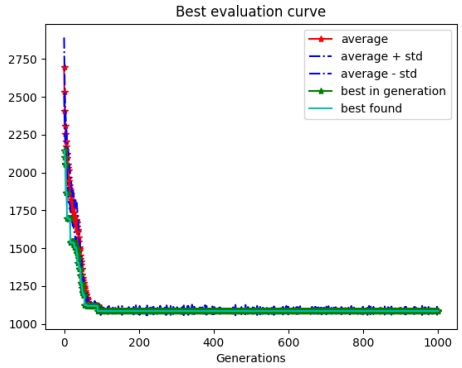
Simple, mutDisplacement, cxVotingRecombination



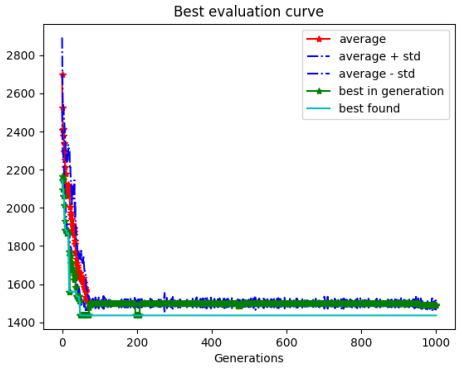
Plus, mutShuffleIndexes, cxPartialyMatched



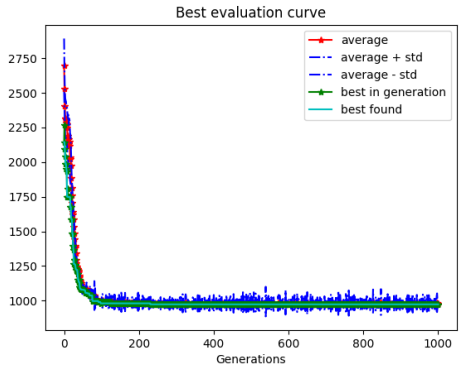
Plus, mutShuffleIndexes, cxVotingRecombination



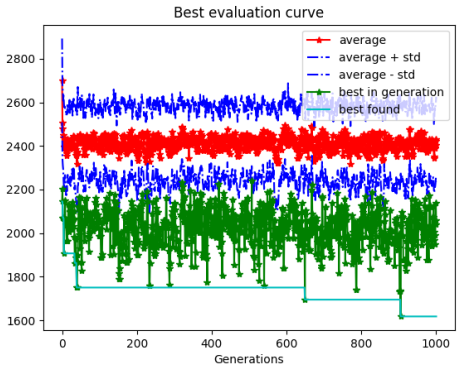
Plus, mutDisplacement, cxPartialyMatched



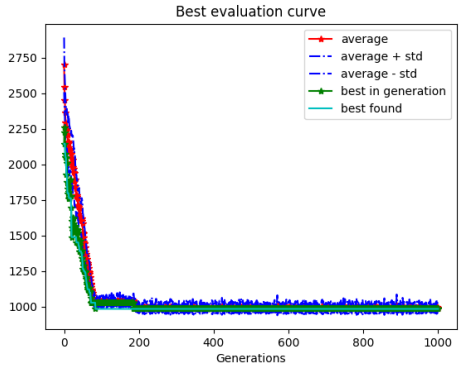
Plus, mutDisplacement, cxVotingRecombination



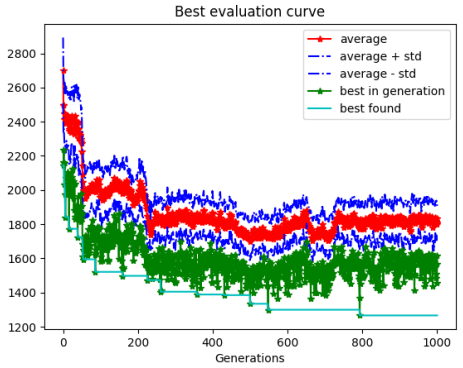
Comma, mutShuffleIndexes, cxPartialyMatched



Comma, mutShuffleIndexes, cxVotingRecombination



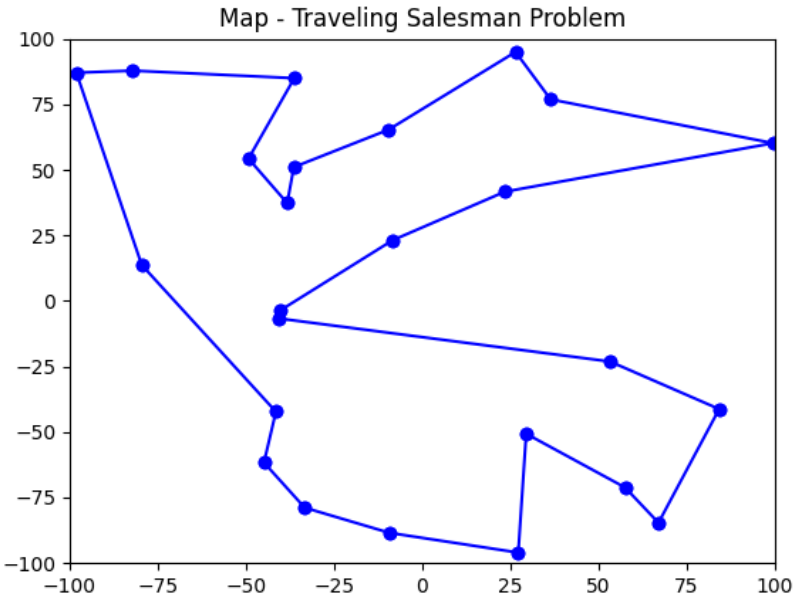
Comma, mutDisplacement, cxPartialyMatched



Comma, mutDisplacement, cxVotingRecombination

Como es posible observar, se obtuvieron los mejores resultados usando la cruza de partially matched, usualmente con el algoritmo de mutación de shuffle indexes, y el algoritmo genético en realidad provocó pocas variaciones en el resultado.

La mejor solución que encontramos, con una versión modificada de nuestro algoritmo utilizando la variante simple y 2,000 iteraciones, fue la siguiente:



Retos encontrados durante la elaboración

Existieron principalmente tres retos durante la elaboración del proyecto:

Mutación, cruce y algoritmos

Cuando se comenzó el desarrollo del proyecto nos dimos cuenta que DEAP tenía ya en su librería múltiples maneras de hacer mutación, cruce y algoritmos, tuvimos que leer

documentación para entender cómo funcionaban internamente y así poder interpretar los resultados arrojados. Muchas de estas funciones parecían empeorar el rendimiento del proyecto, por lo que se tuvo que determinar cuáles eran los que mejor se adaptan al vendedor viajero.

Probar diferentes parámetros

Una vez encontrados los mejores métodos de mutación, cruce y algoritmos el reto fue encontrar los mejores parámetros para cada uno de estos, nos dimos cuenta que no necesariamente añadir más iteraciones mejoraba el rendimiento, si no que el tamaño de la población, probabilidad de mutación y cruce son fundamentales para el buen rendimiento en la búsqueda de soluciones óptimas de un algoritmo genético.

Interpretación de los resultados

Fue difícil determinar qué método es mejor que otro, ya que dependía mucho de la parametrización utilizada para cada uno, al final se optó por usar la misma parametrización en cada ejecución y medir su eficiencia según el número de iteraciones que tomó el algoritmo para llegar a una solución aceptable. Para futuros trabajos pensamos que es importante saber cómo determinar los atributos para hacer más eficiente un algoritmo.

Conclusiones

El problema del vendedor viajero es un lo de los muchos problemas ***NP-completo*** que existen, encontrar una solución eficiente es muy tardado para las computadoras resolviéndolo con fuerza bruta, por lo que es importante encontrar maneras eficientes de solucionarlo ya que en la vida rea encontrar caminos más cortos y soluciones más eficientes sin esperar tanto tiempo es necesario.
El uso de las metaheurísticas para un problema nos ayuda a encontrar soluciones de una manera diferente, en este caso, los algoritmos genéticos se basan en la selección natural y en métodos de mutación, crossover y selección. El uso de estos es un gran recurso de optimización de problemas, en este proyecto nos pareció muy interesante como nos aproximamos a una solución decente sin haber programado un solución de fuerza bruta y que esta tardara muchos menos.

Probar diferentes métodos de mutación, crossover y algoritmos nos ayudó a ver las diferencias entre cada uno y observar su eficiencia para esta representación del TSP utilizada, si bien no existió una que sobresaliera en todos los aspectos, si que la mayoría se aproxima muy bien a un resultado decente.

Como conclusión se puede decir que no existe una selección universal de parámetros o métodos para todos los problemas, se deben de estudiar las diferentes posibilidades y representaciones antes de escoger métodos de mutación o crossover para sacarle provecho.

Aunque se presentaron muchos retos durante la elaboración de este proyecto pudimos fortalecer nuestro conocimiento en el ámbito de inteligencia computacional y observar como el objetivo se cumplía.

Por último, muchas veces se trata de encontrar la mejor solución a un problema sin que importe nada más, pero cuando están en juego otros factores como tiempo, dinero, etc. Es necesario utilizar métodos que nos ayuden a hacer este tipo de tareas de una manera más fácil.

Referencias

1. Karla L Hoffman, Manfred Padberg, Giovanni Rinaldi, et al. 2013. Traveling salesman problem. *Encyclopedia of operations research and management science 1* (2013), 1573–1578
2. Razali, N. M., & Geraghty, J. (2011, July). Genetic algorithm performance with different selection strategies in solving TSP. In Proceedings of the world congress on engineering (Vol. 2, No. 1, pp. 1-6). Hong Kong: International Association of Engineers.
3. Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. Journal of Machine Learning Research 13 (jul 2012), 2171–2175.