Universidad Nacional Autónoma De México
Facultad De Ciencias
Inteligencia Artificial, 2018-II

Proyecto Final:

Algoritmos genéticos: Agente Viajero (TSP)

Fecha de entrega:

Versión 1: 31 de Mayo del 2018
Versión 2: 3 de Junio del 2018

Integrantes:

Martínez Flores, Jorge Yael
Muñiz Patiño, Andrea Fernanda
Sánchez Morales, Rodrigo Alejandro

Introducción

Los problemas computacionales se dividen en clases según su complejidad, puede ser con respecto al espacio o al tiempo, las dos medidas de desempeño son importantes pero para fines de este reporte nos enfocaremos en el tiempo.

Como bien sabemos existen problemas que no pueden resolverse por la complejidad que tienen tanto en espacio y/o tiempo, ha estos problemas se les llama **NP-completos o NP-duros**, los problemas que están en esta clase no tienen un algoritmo que proporcione la solución en tiempo polinomial. Sin embargo se han creado estrategias para tratar de dar una solución a estos problemas por ejemplo: algoritmos de aproximación, algoritmos evolutivos, algoritmos genéticos, etc... Uno de esos problemas es el *Agente viajero* por sus siglas en inglés *TSP*.

El problema del agente viajero consiste en buscar una ruta óptima para recorrer todas las ciudades, sin repetir ninguna y con el menor costo posible, la solución es simple; teniendo la lista de ciudades por visitar, bastaría con calcular todas las combinaciones ordenadas posibles de listas de ciudades y además obtener el costo de visitar las ciudades en el orden establecido por la combinación. Devolviendo como solución la que tenga el menor costo. Sin embargo esto tiene como complejidad $O(n!)$ donde n es el número de ciudades por visitar. Ahora bien, un algoritmo genético tiene una complejidad polinomial, por ello hemos decidido usarlo para tratar de resolver este problema, haciendo énfasis en "tratar" puesto que no siempre se puede proporcionar la respuesta óptima al problema.

¡Viajemos!

Dada una lista de ciudades, las distancias y adyacencias entre cada par de ellas, *¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?*

Para nuestro proyecto utilizaremos una gráfica completa que cumple con la desigualdad del triángulo, es decir:

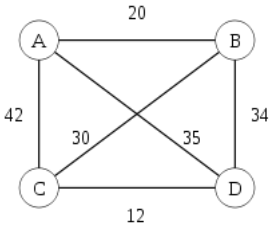


Figura 1: Gráfica G completa

Siempre será menos costoso ir de la ciudad A a la ciudad D, que ir de la ciudad A a la ciudad B y luego visitar la ciudad D.

Teniendo las consideraciones anteriores podemos plantear el problema de la siguiente manera:

Determinar un ciclo hamiltoneano en la gráfica G con el menor costo posible.

Algoritmos genéticos

Fue en 1970 cuando surgieron los algoritmos genéticos, una rama prometedora de la Inteligencia Artificial, creada por John Henry Holland.

Los algoritmos genéticos están inspirados en la evolución biológica, pues modifican alternadamente las características de los individuos. Son una estrategia de la inteligencia artificial para tratar de resolver ciertos problemas.

Las componentes esenciales de un algoritmo genético son:

- Población (Conjunto de soluciones factibles del problema).
- Proceso de selección.
- Proceso de reproducción.
- Proceso de reemplazo.

Antes de hablar de cada una de las etapas de nuestro algoritmo, estableceremos la codificación pertinente para nuestro problema. Tenemos una gráfica G completa, que cumple la desigualdad del triángulo, por ello podemos asegurar que nuestra salida es un ciclo hamiltoneano.

La codificación de las propuestas de solución es de la siguiente manera: Dadas las 50 ciudades, cada una será representada por un número, la correspondencia esta en el *apéndice B*. Para representar un tour tendremos un arreglo de tamaño 50, donde en cada posición tendremos un número que representa a una ciudad, la posición 0 determina de qué ciudad se parte inicialmente, y subsecuentemente en qué orden se van visitando las ciudades, por ejemplo si quisiéramos codificar un tour de 7 ciudades lo haríamos de la siguiente manera:

T1: (7, 5, 1, 2, 4, 3, 6)

Lo cual significa que debemos partir de la ciudad 7, recorrer todas las ciudades y regresar a la misma ciudad 7, en el orden que establece la codificación, es decir, primero visitar la ciudad 5, luego la ciudad 1 y así sucesivamente. El siguiente diagrama representa el orden en el que se visitan las ciudades.

$$T1 : 7 \to 5 \to 1 \to 2 \to 4 \to 3 \to 6 \to 7$$

La función de evaluación es el **fitness** que para nuestro algoritmo genético será el costo del tour, es decir, la suma del costo de visitar cada ciudad en el orden pre-establecido, el cual se obtiene de la posición correcta en la matriz *costos* que contiene los registros del costo de visitar la ciudad B a partir de la ciudad A.

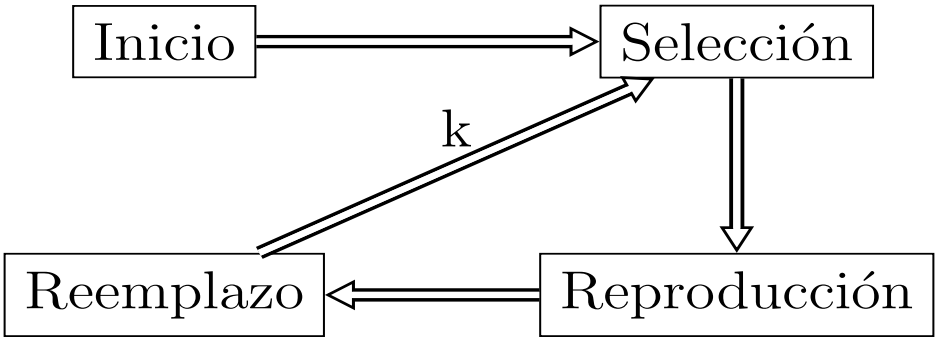
Matemáticamente hablando tendríamos que el fitness es:

$$\sum_{i=0;j=1}^{50} cost[i][j]$$

Debemos mencionar que en el caso del fitness para nuestra implementación mientras más chico sea el valor del fitness es mejor para el individuo, pues esto significa que el costo del tour es menor.

Descripción de las etapas del algoritmo genético

A continuación se presenta un diagrama para describir la forma de operar de nuestro agente.



Primero se da una población inicial, es decir, candidatos a solución, en nuestro caso son n tour codificados, donde $n = 50$, estos candidatos son tour´s aleatorios que van a determinar nuestra población inicial de tour´s.

En cada ciclo de reproducción se crea una nueva generación de tour´s. Esta se genera realizando cruces entre los elementos de la población. Para nuestro algoritmo genético hemos implementado los siguientes 3 cruces:

- Cycle crossover
- Partilly mapped crossover
- Order crossover

Se usan específicamente estos cruces debido a que estos nos garantizan que no tendremos elementos repetidos, puesto que estamos buscando un ciclo hamiltoneano y este no debe repetir vértices.

Los pares de padres se eligen de tal manera que siempre cruzaremos el mejor tour con el peor, esto gracias a que previamente se ordenaron con respecto al valor del fitness. Esto con el objetivo de no caer en óptimos locales, ya que al cruzar los dos mejores tour´s podriamos caer en óptimos locales, de manera análoga podría decirse que es como si solo buscáramos el valor máximo de una función en un intervalo especificado, el máximo puede estar en ese intervalo o no.

El cruce a utilizar se elige de manera aleatoria, así que nunca sabemos de manera concreta que cruce se utilizo para crear al hijo h de la generación.

A continuación explicaremos brevemente el cruce *Partilly Mapped Crossover:*

Este cruce consiste en hacer un mapeo de una subcadena obtenida del *padre 2*.

El cruce genera un hijo tal que cada ciudad y su posición viene de uno de los padres. Se trata de que "hereden"las posiciones de las ciudades.

Consideremos a los siguientes padres:

$$P_1 : (1, 2, 3, 4, 5, 6, 7, 8, 9)$$
$$P_2 : (4, 5, 2, 1, 8, 7, 6, 9, 3)$$

Tomaremos la subcadena del padre 2 que va de la posición 4 a la posición 7, y con base en eso obtenemos la función de mapeo que es la siguiente:

$$1 \longleftrightarrow 4$$
$$5 \longleftrightarrow 8$$
$$6 \longleftrightarrow 7$$

Llevando acabo el cruce obtenemos lo siguiente:

$$P_1 : (1, 2, 3, 4, 5, 6, 7, 8, 9)$$
$$H_1 : (X, X, X, 1, 8, 7, 6, X, X)$$

Se añaden al hijo los valores del padre 1 que no estan en la función del mapeo

$$H_1 : (X, 2, 3, 1, 8, 7, 6, X, 9)$$

Se añaden los valores establecidos por el función mapeo

$$H_1 : (4, 2, 3, 1, 8, 7, 6, 5, 9)$$

Obteniendo así el resultado del cruce.

Ahora bien, las mutaciones deben ocurrir con una probabilidad de 0,25 ya que no debe ser muy común que estas se apliquen a los tour´s, pues como en la biología las mutaciones deben ocurrir esporadicamente.

Una vez que ya obtuvimos los hijos realizando los cruces y aplicando la mutación (en caso de ser así), es momento de de pasar a la etapa de reemplazo, la población de nuestro algoritmo genético siempre sera la misma, por lo cual tenemos que reemplazar tour´s, es decir algunos padres serán reemplazados por los hijos, el criterio de reemplazo es aleatorio, pero siempre conservando el candidato con mejor fitness.

Condición de término

Un algoritmo genético no tiene forma de percibir si ha encontrado la solución óptima, por ello se le da una condición de término, existen para TSP dos alternativas de condición, la primera es que dado un tour con costo n se detenga la ejecución del algoritmo, la segunda es que una vez creadas k generaciones, simplemente se detenga el algoritmo y devuelva el tour con el fitness mas bajo. En el caso de nuestra implementación una vez creadas 20 generaciones, detendremos el algoritmo genético y proporcionaremos el tour con menor costo.

Sobre el agente

Nuestro propósito es obtener el mejor tour posible mediante algoritmos genéticos, por lo cual nos conviene entonces usar un agente racional, ya que con la incertidumbre de los valores del fitness lo que queremos es obtener el mejor resultado.

Nuestro objeto *tour* tiene como atributo un costo, que lo determina el fitness, el fitness se puede ver como la función de utilidad, sin embargo las acciones de nuestro agente no se basan en el fitness obtenido por cada tour, es decir que el fitness en la consideración más importante de nuestro agente, pero el proceso no lo determina este valor. Por ello no utilizamos un agente basado en utilidad, ya que el fitness no nos indica el siguiente paso a realizar.

Es por ello es un agente basado en modelos, esto por el simple hecho de que tiene la capacidad de guardar los candidatos a solución.

Enfoque de nuestro agente

Usando esta estrategia para solucionar el problema de *TSP* estaríamos usando un agente que actúa racionalmente, pues lleva un proceso bien determinado y especificado, con ello podemos garantizar que nuestro agente llevara acabo un proceso de manera racional, siendo así el agente no realizara acciones sin sentido. Esto es por la capacidad que tenemos aun para tratar de resolver problemas como este, ademas de que no sería pertinente usar algún otro enfoque de la *IA*.

Agente	Rendimiento	Entorno	Actuadores	Sensores
Agente Racional	Proporcionar como solución el tour de menor costo.	Matriz de 50×50 y lista de candidatos a solución.	Cruces y mutaciones.	La función fitness.

Cuadro 1: REAS.

Entorno de trabajo

La manera en hacer los cruces es determinista, el cruce siempre se realiza de la forma determinada, sin embargo los individuos (candidatos a solución) son los que van cambiando, por lo cual no obtenemos los mismos resultados, pues ese es el propósito. Sin embargo, la manera en decidir si se hace la mutación o no, se basa en la probabilidad de obtener un número aleatorio dentro de un intervalo X, teniendo la probabilidad de 0.25 de obtener un número dentro de ese intervalo, esto se hace con una función totalmente no determinista.

Al usar una matriz como nuestro esquema de representación, tenemos que nuestro entorno es totalmente observable, puesto que siempre podemos tener acceso a nuestra población, sin embargo en este caso no es importante poder obtener siempre todos los tour's ya que sólo nos interesa tener siempre el tour con el mejor fitness.

Retomando el diagrama de la función del agente, podemos observar que nuestro entorno es episódico, ya que cada generación nueva depende de la anterior, puesto que los padres determinan a los nuevos hijos y así sucesivamente. Es decir que la próxima generación no puede ser creada sin que los padres estén bien determinados.

Los costos de ir de una ciudad A a una ciudad B están representados en una matriz de tamaño $n \times n$, tenemos ciudades finitas y nuestra condición de término está determinado por k por lo cual nuestro entorno de trabajo es finito.
Sólo tenemos un agente haciendo mutaciones y cruces, además de que es de manera iterativa, no tenemos trabajando a la par alguna otra implementación por lo cual estamos solucionando el problema con un sólo agente.

Entorno de trabajo	Observable	Determinista	Episódico	Discreto	Agente
Generación actual	Totalmente	No	Sí	Sí	Individual

Cuadro 2: Entorno de trabajo

Análisis de complejidad

La complejidad en este caso depende absolutamente de la forma en que operemos los candidatos a solución (los tour´s) y la manera de almacenarlos.
Para el análisis de la complejidad de los métodos, los expresaremos de una forma sencilla, destacando que no es la misma forma de implementarlo en el código, sin que eso perjudique la complejidad final.

Nuestra condición de término en el algoritmo genético es realizar k generaciones sin importar qué valor tenga el fitness del mejor tour. Nuestra población inicial es de tamaño n, al realizar un cruce por cada dos padres obtendremos $\frac{n}{2}$ nuevos tour's, pero ademas debemos considerar que lo haremos k veces por lo cual generaremos un total de $\left(\frac{n}{2}\right) \cdot k$ tour's, ahora bien, si resulta que nuestra población inicial coincide con el número de generaciones por crear entonces tendríamos lo siguiente:

$$\left(\frac{n}{2}\right) \cdot k$$

Supongamos que

$$\begin{aligned} & n = k \\ & \Rightarrow \left(\frac{k}{2}\right) \cdot k \\ & \Rightarrow O\left(k^2\right) \end{aligned}$$

Entonces tenemos que la complejidad teórica es $O(k^2)$ esto sin duda es mejor que $O(n!)$ por lo cual justificamos que al usar algoritmos genéticos estamos mejorando la complejidad, dando

una cuasi óptima solución.

Complejidad en espacio

Para poder resolver nuestro problema tenemos requisitos de entrada los cuales son:

- Conjunto de ciudades.
- Lista de adyacencias, con costos.
- Ciudad inicial.

Hemos guardado las adyacencias con sus costos en una matriz que tiene la forma de ser triangular inferior de tamaño 50×50, implícitamente también están enlistadas las ciudades.

Así que si quisiéramos registrar n ciudades nuestra complejidad sería:

$$\Rightarrow O(n \times n)$$
$$\Rightarrow O(n^2)$$

Entonces la complejidad de nuestra implementación es: $O(2500)$ Como ya sabemos en cada iteración generamos $\frac{n}{2}$ hijos que sólo se almacenan temporalmente hasta pasar al proceso de reemplazo, por lo cual sólo necesitaríamos un extra de memoria para guardarlos.

Complejidad en tiempo

Los métodos de mutación son una operación constante, no importa el tamaño del tour, basta con pasarle al método el tour y los indices sobre los cuales hay que mutar.

```
   public static void exchange(int arr[], int i, int j) {
      int aux = arr[i];
      arr[i] = arr[j];
      arr[j] = aux;
}
```

Otra función que hay que tomar en consideración es *fitness* que consiste en lo siguiente:

```
   //la variable individuo es un arreglo que contiene el tour.
      int k = 0;
      int l;
      int fitness = 0;
      int der = 0;
      int izq = 0;
      for (l = 1; l < 50; l++) {
         if (individuo[k] > individuo[l]) {
            der = individuo[k];
            izq = individuo[l];
         } else {
            der = individuo[l];
            izq = individuo[k];
```

```
        }
        fitness = fitness + costos[der][izq];
        k++;
      }//Acaba el for
      //Al final se calcula el costo de ir de la ultima ciudad a la primera
      int puntoDePartida = individuo[0];
      int ultimaPosicion = individuo[49];
      if (individuo[0] > individuo[49]) {
        fitness = fitness + costos[puntoDePartida][ultimaPosicion];
      } else {
        fitness = fitness + costos[ultimaPosicion][puntoDePartida];
}
```

Calcular el *fitness* cuesta tiempo lineal, pues basta con recorrer el arreglo una vez. Por ende tiene complejidad $O(n)$ donde n es el número total de ciudades, que en este caso es 50, por lo tanto la complejidad especifica es $O(50)$.

Sumarle a *fitness* el costo de ir de la última ciudad a la primera, para cerrar el ciclo hamiltoneano es constante.

Implementación

En nuestra implementación existe la posibilidad de cambiar algunos parámetros de entrada, como lo son:

- El número de generaciones que se crean.
- El tamaño de la población inicial.

Observaciones

El objetivo del agente es dar un tour cuasi optimo, para analizar de forma detallada si se alcanza el objetivo tendríamos que realizar muchas ejecuciones, nosotros analizaremos dos.

Primera ejecución

El tour de mayor costo tuvo un fitness de 70182 en la ultima generación, pero el algoritmo nos devuelve un tour cuasi optimo de 63040 lo cual no representa una mejora muy grande.

Segunda ejecución

El tour de mayor costo, al terminar la condición de terminó es de 80230 y el tour cuasi optimo que devuelve el algoritmo genético tiene un valor de fitness de 63040 así que podemos concluir que a veces se obtienen buenos resultados, alejados del mayor costo, pero a veces no. Estamos a la expectativa de la probabilidad, al parecer.

Sin embargo podemos asegurar que siempre devuelve el de menor costo.

Ventajas y Desventajas

La desventaja más concreta es que nada nos garantiza que obtengamos la solución óptima, de hecho nada nos garantiza siquiera obtener una solución cerca de la óptima, pues hay que tener cuidado de no caer en óptimos locales.

También otra desventaja a la que nos presentamos es que dependemos absolutamente del azar de los cruces, efectivamente se usan cruces convenientes para nuestro problema, pero estamos a la expectativa de que con los cruces se optimice el costo del ciclo hamiltoneano.

La ventaja de usar algoritmos genéticos es la complejidad en tiempo que tenemos para poder dar una respuesta aceptable al problema del agente viajero, obtener el tour óptimo tiene complejidad de $O(n!)$ que como se mencionó antes es un problema **NP-duro**, al usar algoritmos genéticos obtenemos una complejidad polinomial.

Otra ventaja es que implementar este tipo de agente no es complicado, lo más complicado a lo que se podría enfrentar, es al manejo de arreglos.

Conclusiones

Como sociedad siempre queremos lo mejor y de la manera más rápida que se pueda, obteniendo una respuesta cuasi óptima podemos avanzar y/o empezar a desarrollar alguna tarea, esto siempre será mucho mejor que no obtener una respuesta en un tiempo considerable, para nuestro problema podemos decir que es mejor obtener un tour de costo n, donde el usuario lo pueda aceptar, a nunca obtener la respuesta óptima.

Usar algoritmos genéticos nos ayuda a tratar de buscar la mejor solución en un tiempo polinomial, es sin duda una de las mejores opciones que tenemos actualmente para brindar una solución "optima" al problema del tour. Los algoritmos genéticos no son complicados, simplemente tenemos que simular un poco a la biología y las transformaciones que sufren los individuos.

Bibliografía

- Russell, S. & Norman, P.. (2009). Artificial Intelligence: A Modern Approach. Upper Saddle River: Prentice Hall.
- Desconocido. (2000). 1940salesman.jpg [Ilustración]. Recuperado de `https://ese.wustl.edu/ContentFiles/Research/UndergraduateResearch/CompletedProjects/WebPages/sp12/KevinTeng/1940salesman.jpg`
- Gasca Soto, María de la luz. (2003). Introducción a los problemas NP-completos. Nota 19° de la Facultad de Ciencias.
- Guia Roji (2004)

Apéndice A

A continuación se muestra la matriz de adyacencias y costos de nuestra gráfica G a utilizar.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2	908									
3	1513	1668								
4	2007	2162	494							
5	1193	1782	721	1215						
6	2258	1389	3020	3512	3132					
7	1117	515	1578	2072	1666	1493				
8	678	448	1899	2393	2013	1780	886			
9	306	602	1207	1701	1293	1952	810	833		
10	1651	970	2417	2911	2531	1466	1373	922	1351	

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	1703	1858	424	388	839	3208	1768	2045	1397	2607
12	117	791	1396	1890	1310	2141	999	804	189	1540
13	1882	1013	2642	3136	2756	376	1117	1404	1576	1191
14	1310	441	2070	2564	2184	1038	844	832	1004	529
15	3350	2709	4116	4610	4230	1425	2813	2621	3050	1699
16	889	246	1697	2191	1811	1578	684	202	631	720
17	760	181	1520	2014	1634	1570	559	479	454	997
18	2348	1707	3114	3608	3228	769	1811	1619	2048	697
19	4701	4060	5467	5961	5581	2776	4164	3972	4401	3050
20	782	127	1542	2036	1656	1516	541	425	476	943

	11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
11										
12	1586									
13	2832	1765								
14	2260	1193	662							
15	4306	3239	1696	2228						
16	1887	820	1202	630	2419					
17	1710	643	1194	622	2696	277				
18	3304	2237	694	1226	1002	1417	1694			
19	5657	4590	3047	3579	1351	3770	4047	2353		
20	1732	665	1140	568	2642	223	54	1640	3993	

Apéndice B

A continuación se enlistan las ciudades de México que son representadas con los números naturales con el intervalo de $1 \leq i \leq 50$

1. Acapulco (Guerrero)
2. Agusacalientes (Aguascalientes)
3. Campeche (Campeche)
4. Cancún (Quintana Roo)
5. Ciudad Cuauthemoc (Chiapas)
6. Ciudad Juarez (Chihuahua)
7. Colima (Colima)
8. Cuernavaca (Morelos)
9. Culiacán (Sinalona)
10. Chetumal (Quintanna Roo)
11. Chilpancingo (Guerrero)
12. Chihuahua (Chihuahua)
13. Durango (Durango)
14. Ensenada (Baja California)
15. Guadalajara (Jalisco)
16. Guanajuato (Guanajuato)
17. Hermosillo (Sonora)
18. La Paz (Baja california)
19. León (Guanajuato)
20. Manzanillo (Colima)
21. Matamoros (Tamaulipas)
22. Mazatlán (Sinaloa)
23. Mérida (Yucatán)
24. Mexicali (Baja California)
25. México (Ciudad de México)
26. Morelia (Michoacán)
27. Monterrey (Nuevo León)
28. Nogales (Sonora)
29. Nuevo Laredo (Tamaulipas)
30. Oaxaca (Oaxaca)
31. Pachuca (Hidalgo)
32. Piedras Negras (Coahuila)
33. Puebla (Puebla)
34. Querétaro (Querétaro)
35. Reynosa (Tamaulipas)
36. Salinas Cruz (Oaxaca)
37. Saltillo (Coahuila)
38. San Luis Potosí (San Luis Potosí)
39. Tampico (Tamaulipas)
40. Tapachula (Chiapas)
41. Tepic (Nayarit)
42. Tijuana (Baja California)
43. Tlaxcala (Tlaxcala)
44. Toluca (Estado de México)
45. Torreón (Coahuila)
46. Tuxtla Gutiérrez (Chiapas)
47. Veracruz (Veracruz)
48. Villahermosa (Tabasco)
49. Xalapa (Veracruz)
50. Zacatecas (Zacatecas)