UNIVERSIDAD
DE GRANADA

Práctica 4: Programación Dinámica

TSP con Programación dinámica

José María Gómez García
Fernando Lojano Mayaguari
Valentino Lugli
Carlos Mulero Haro

Universidad de Granada
Granada
Mayo 2020

Índice

1 Introducción

A lo largo de este documento analizaremos el enfoque de programación dinámica aplicado a la resolución del problema del viajante del comercio. Si bien aplicando el enfoque voraz eramos capaces de conseguir una buena solución a este problema, no era posible obtener la solución optima. Esta vez sin embargo, haciendo uso de la programación dinámica, obtendremos la mejor solución posible para cada instancia del problema. No obstante, antes de comenzar con el análisis del problema con dicho enfoque, vamos a redactar una pequeña explicación sobre el propio enfoque de la programación dinámica con el objetivo de entender de una forma mas precisa el análisis.

En primer lugar, la programación dinámica es útil y aplicable en la búsqueda de la solución de problemas de carácter n-etápicos (Problemas de decision Markovianos) usando las ya conocidas ecuaciones de recurrencia.

Un problema n-etápico es aquel que se divide en n numero de etapas en las cuales se encuentra una solución local y cuya solución global es el conjunto completo de soluciones locales

De esta forma, entendemos que la programación dinámica consiste en obtener una solución a partir de las soluciones encontradas en los subproblemas en los que se ha dividido.

Además de ser n-etápico, otra condición necesaria que los problemas deben cumplir para poder aplicar este enfoque es el principio de optimalidad del Bellman (POB). Este principio afirma que si las decisiones tomadas anteriormente durante el problema son óptimas, la decisión que hay que tomar en la instancia actual y las que quedan después también lo serán. En otras palabras, una solución optima solo puede estar formada por sub soluciones que también sean óptimas.

El proceso a seguir para aplicar este enfoque es el siguiente:

- **Naturaleza N-etápica:** Debemos comprobar que el problema tiene una naturaleza N-etápica.
- **Cumplimiento del POB:** Se tiene que comprobar que se cumple el principio de optimalidad de Bellman. Si este se cumple, podemos aplicar el enfoque, de lo contrario, no se puede.
- **Ecuación de recurrencia:** Una vez se haya demostrado el POB, debemos plantear una ecuación de recurrencia.
- **Encontrar una solución:** Esta será una solución optimal que iremos obteniendo a partir de las soluciones obtenidas por los subproblemas.

Por último, cabe destacar que la forma de construir la solución se puede llevar a cabo mediante dos enfoques, desde el primer estado hasta el último (enfoque adelantado) o de forma invertida (enfoque atrasado).

Ahora que hemos dado una pequeña explicación acerca de la programación dinámica, pasaremos a analizar su aplicación al conocido problema denominado como Travelling Salesman Problem o TSP.

2 Travelling Salesman Problem

2.1 Descripción

El TSP o Problema del Viajante de Comercio en la formulación que se trabaja, se posee un grafo $G = (V, A)$ completo, ponderado y no dirigido, se desea recorrer todas las ciudades una sola vez partiendo de una ciudad inicial y regresando a ella con el menor coste, esto es, obtener el ciclo hamiltoniano minimal para el grafo G.

2.1.1 Problema N-etápico

Para poder aplicar Programación Dinámica, un aspecto vital a considerar es si el problema es N-etápico, esto es, que el problema se puede descomponer en subproblemas que no varíen entre si por más de una unidad o dicho de otro modo, que el problema está conformado por un número de etapas en las cuales se debe de tomar una decisión. En nuestro caso sí se cumple, ya que para obtener el ciclo hamiltoniano mínimo se debe decidir a que ciudad viajar de primero, luego a cual otra ir de segundo y así sucesivamente hasta que finalmente se hayan visitado todas las ciudades una sola vez y se complete el ciclo regresando a la ciudad de origen.

2.1.2 Verificación del Principio de Optimalidad de Bellman (POB)

El principio de optimalidad de Bellman, el cual nos dice que una solución óptima está compuesta por subsoluciones óptimas, se verifica ya que si se tiene un ciclo hamiltoniano minimal que comienza y finaliza, por ejemplo, en la ciudad 1, este ciclo consiste de un camino de la ciudad 1 a otra ciudad j junto a otro camino que parte de j visitando el resto de ciudades una sola vez y terminando en 1, si el ciclo posee el costo mínimo por lo tanto también el camino de j a 1 es de costo mínimo, ya que si existiese otro camino diferente de j a 1 que fuese de coste aún menor, este ya hubiera sido elegido en el primer lugar, por lo tanto se cumple el Principio de Optimalidad de Bellman.

2.1.3 Ecuación de recurrencia

Para construir la ecuación primero debemos definir que las distancias entre dos nodos se encuentran en una matriz D de distancias donde $D_{[i,j]} \geq 0$ con $i \neq j$ y $D_{[i,j]} = 0$ si $i = j$ que va a ser esencial para la ecuación recurrente, adicionalmente y sin pérdida de generalidad tomaremos que el ciclo comienza y termina en el nodo 1. Ahora definimos $g(i, S)$ donde $S \subseteq N - \{1\}$ e $i \in N - S$, como el valor del camino mínimo que parte del nodo i visitando cada nodo del conjunto S una vez hasta llegar al nodo 1, con el caso particular de que $i = 1$ solo si $S = N - \{1\}$. Por el POB obtenemos la siguiente ecuación recurrente:

$$g(i, S) = \begin{cases} D_{[i,1]} & \text{si } S = \emptyset. \\ \text{Min}_{j \in S}(D_{[i,j]} + g(j, S - \{j\})) & \text{en caso contrario.} \end{cases}$$

De aquí podemos deducir que $g(1, N - \{1\}) = \text{Min}_{2 \geq j \geq n}(D_{[1,j]} + g(j, S - \{j\}))$ dará el ciclo hamiltoniano minimal. La eficiencia teórica es de $\text{O}(n^2 \cdot 2^n)$ primeramente porque se deben realizar $(n-1)$ consultas a una tabla, también $(n-1)$ sumas para calcular $g(1, N - \{1\})$ y realizar las adiciones de los posibles conjuntos $g(i, S)$, lo que supone realizar $(n-1) \cdot n2^n$ sumas.

2.2 Pseudocódigo

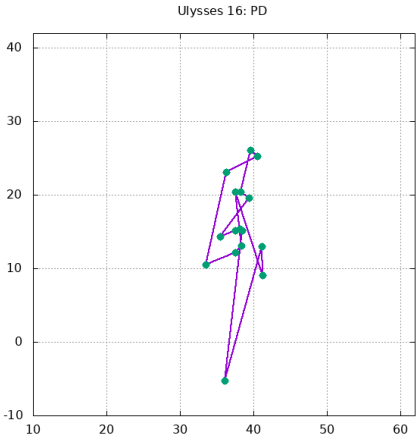
La función g devuelve la distancia del recorrido que parte de un nodo n y vuelve al mismo, pasando por todos los nodos del vector s antes de la forma más óptima posible. Para ello hacemos lo siguiente:
En caso de ser s vacío (caso base), la distancia mínima a devolver será el elemento (n,0) de la matriz de distancias. Si no es vacío, se busca el recorrido y la distancia que se pide en el mapa del objeto TSP, para así no recalcularla. En caso de no estarlo, se calcula comparando con la distancia menor, una distancia menor auxiliar (en bucle para i = 0 y mientras sea menor que el tamaño de s) cuyo valor será la suma de la posición (n, s[i]) de la matriz de distancias y el resultado de ejecutar recursivamente la función g con s[i] y un vector auxiliar que tiene los mismos elementos que el vector s, salvo el que ocupa la posición i, que el auxiliar no lo tendrá. En el caso de que la distancia mínima auxiliar sea menor que la que ya estaba, la distancia mínima auxiliar pasará a ser el valor mínimo a devolver. Después guardamos el camino óptimo en el mapa de memoria.
Finamente se devuelve el valor calculado.
Como la función g solo nos devuelve la distancia del recorrido mínimo hemos tenido que crear otra función que nos diga el orden de los nodos para hacer dicho camino: la función camino.
La función camino tiene como parámetros de entrada un nodo 'n', que es el nodo por el cual queremos empezar el camino (normalmente será siempre 0 para hacer un camino que empiece y acabe por él mismo, es decir, un circuito); y el resto de nodos por los que queremos que pase, 's'. El camino empieza por 'n' por lo que este ya puede ser añadido a la solución; a continuación consultamos las distancias (tal y como hace g: Ln,i + g(i, s-i)) con todos los posibles caminos desde 'n' hasta 0 y elegimos la menor. El nodo 'i' es añadido a la solución y sacado de 's' para consultar así la menor distancia desde i pasando por todos los posibles caminos de 's' una y otra vez hasta que 's' quede vacío y por tanto todos los nodos de 's' se hayan añadido en orden a la solución.

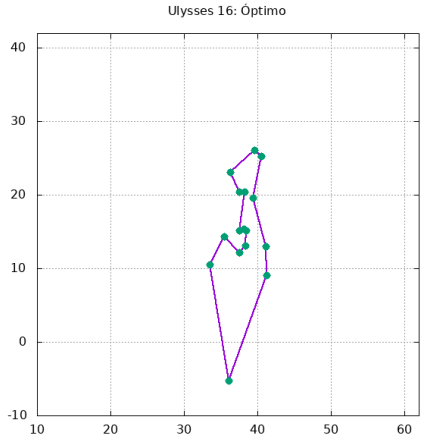
```
funcion g(n, s) //Siendo n un entero que representa un nodo y s el vector que
   contiene los demas nodos del grafo
inicio
    distancia_min := INT_MAX
    distancia_min_aux := INT_MAX
    s_aux //Vector de int auxiliar

    si(!s.vacio())
        si(calculado(g(n,s)))
            distancia_min := distancia menor del mapa
        fin
        si (!calculado(g(n,s)))
            para i:=0 mientras i< s.size()
                s_aux = s
                s_aux.borrar(i)
                distancia_min_aux := distancias[n][s[i]] + g(s[i], s_aux)

                si(distancia_min_aux < distancia_min)
                    distancia_min := distancia_min_aux
                fin
            fin
            guardar_camino(distancia_min)
        fin
    fin
    si(s.vacio())
        distancia_min := distancias[nodo][0]
    fin

    devolver distancia_min
fin
```

```
funcion camino(n, s) //Siendo n un entero que representa un nodo y s el vector
   que contiene los demas nodos del grafo
inicio
    distancia_min
    distancia_min_aux
    indice_nodo_min
    solucion
    sin_solucion := s
    s_aux

    solucion.insertar(n)

    mientras (solucion.size() <= s.size())
        distancia_min := INT_MAX
        indice_nodo_min := -1

        para cada nodo que no esta en el camino
            s_aux := sin_solucion
            s_aux.borrar(nodo_actual)

            distancia_min_aux = distancias[ultimo elemento solucion][
               sin_solucion[nodo_actual]] + g(sin_solucion[nodo_actual], s_aux)

            si(distancia_min_aux < distancia_min)
                distancia_min := distancia_min_aux
                indice_nodo_min := nodo_actual
            fin
        fin
        solucion.insertar(sin_solucion[indice_nodo_min])
        sin_solucion.borrar(nodo_actual)
    fin
    devolver solucion
fin
```

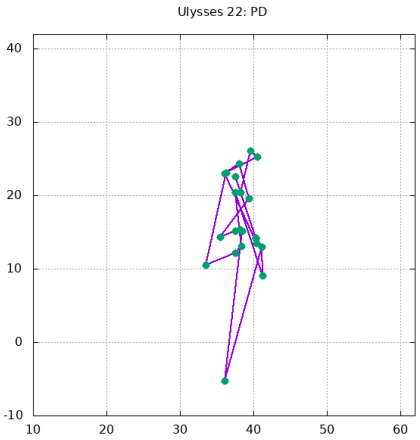
2.3 Escenarios de ejecución

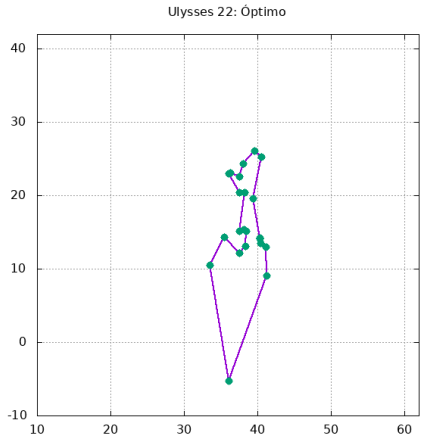
Una vez comprobado que la programación dinámica es aplicable a este problema y de haber expuesto el pseudocódigo del algoritmo que hemos implementado, pasaremos a mostrar los escenarios de ejecución sobre los cuales se ha ejecutado el algoritmo.





Ulysses 16





Ulysses 22

Viendo la comparativa de las gráficas obtenidas mediante Programación dinámica y las otorgadas para el análisis como resultados óptimos, se presenta la necesidad de dar una explicación, pues la trayectoria del camino en ambos casos, Ulysses16 y Ulysses22, son poco similares a las trayectorias presentadas por sus respectivos óptimos.
El motivo de esta anomalía se debe a que, si nos fijamos en la documentación ofrecida en `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/` la cual aparece referenciada en el guión de la práctica 3 como soporte para poder entender mejor el funcionamiento de la librería TSPLIB, podemos ver que el cálculo de las distancias en ambos casos de Ulysses se ha hecho con un método diferente al que se nos ha propuesto para estas prácticas (distancia Euclídea).
De esta forma, al tomar cada nodo valores de distancia diferentes a los que se obtendrían con el método de cálculo realizado en los casos ofrecidos como óptimos, en cada momento del algoritmo, este tomará un par de nodos diferente, dando como resultado un camino diferente.

No obstante, si bien parece que los resultados son muy diferentes analizando solamente las gráficas, lo cierto es que no existe mucha diferencia entre las distancias obtenidas y las óptimas ofrecidas. Vamos a reflejar los resultados en una tabla para poder contrastarlos mejor.

TSP	Óptimo ofrecido	Programación dinámica
Ulysses16	73	71
Ulysses22	74	72

Como podemos ver, únicamente hay dos unidades de distancia entre los resultados obtenidos y los que se nos ofrecen. No obstante, esa pequeña diferencia nos sirve para llegar a la conclusión de que mediante el cálculo de la distancia Euclídea unida a la programación dinámica podemos obtener un resultado mejor en cuanto a estos dos casos, convirtiéndolos en óptimos.
Otro punto que cabe destacar es que, se nos proporcionaron 2 archivos TSP más a parte de los ya expuestos anteriormente para analizar. Uno de ellos cuenta con 48 nodos y el último con 280.
Sin embargo, debido al orden de eficiencia de este algoritmo, al intentar llevar a cabo su ejecución, este tardaría demasiado en resolver el problema con 48 nodos. Por ende, podemos asumir que si intentáramos ejecutarlo para 280 nodos, no llegaríamos a ver la solución en esta vida.

2.4 Comparación Greedy

Para comparar el problema TSP en Programación Dinámica y en Greedy es necesario medir el tiempo de ejecución que estos tardan en resolver el problema:

Ulysses16	Tiempo de Ejecución (s)	Camino
Programación Dinámica	6.312	71
Cercanía	0	103
Inserción	0	105
Aristas	0	90

Ulysses22	Tiempo de Ejecución (s)	Camino
Programación Dinámica	1029.45	72
Cercanía	0	93
Inserción	0	107
Aristas	0	91

Los tiempos han sido tomados en un HP Pavilion con procesador Intel Core i5-1035G1 con 8 GB de RAM. Podemos observar como el tiempo de ejecución para obtener la solución Greedy es muy cercano a 0; sin embargo la Programación Dinámica si nos muestra tiempos más elevados, llegando a obtener 17 minutos aproximadamente para resolver ulysses22 en comparación a los 0 s. de los algoritmos Greedy. Es aquí cuando debemos preguntarnos qué preferimos, si una solución óptima pero lenta, o una que se aproxime al óptimo pero rápida.

Caminos Mínimos

TSP	PD	Cercanía		Inserción		Aristas	
		Camino	Diferencia	Camino	Diferencia	Camino	Diferencia
Ulysses 16	71	103	45,07%	105	47,88%	90	26,76%
Ulysses 22	72	93	29,17%	107	48,61%	91	26,38%

Si en nuestro caso queremos un camino óptimo estricto vamos a elegir la Programación Dinámica como solución; sin embargo, si queremos una solución rápida que se aproxime al óptimo podemos elegir resolver el problema por Aristas, que calcula el camino con una diferencia menor al 27% en ambos casos, siendo el algoritmo Greedy que mejor resultado nos ha dado.

2.5 Conclusión

Considerando los resultados obtenidos, podemos concluir que los algoritmos implementados con programación dinámica resultan una opción más que interesante para resolver problemas de optimización n-etápicos, ya que en estos casos, dichos algoritmos no solo tratan de reducir el tiempo de ejecución, sino que también obtienen el mejor resultado para estos problemas.
Sin embargo para nuestro problema particular, el problema del viajante de comercio, si bien obtenemos el resultado óptimo para resolver el problema, debemos destacar el elevado tiempo de ejecución que el algoritmo tiene.
En resumen, al tratarse el problema TSP de un problema NP completo no resulta conveniente su resolución mediante programación dinámica. Es cierto que de esta forma obtenemos el resultado óptimo para el problema, sin embargo el uso de algoritmos heurísticos en este caso nos da soluciones que son próximas a la óptima sin tomar tanto tiempo de ejecución.