Algoritmo de Dijkstra vs Implementación de Dial

Autor : Steffan Widemann
Fecha : 05/06/2021

1 Detalles de la implementación

En esta sección se presentan las características principales de los métodos programados. Sea $G = (N, A)$ el grado direccional considerado, con $|N| = n$ y $|A| = m$.

a Características comunes

- Para ambos métodos se utiliza el mismo método de lectura de parámetros. Al momento de leer el archivo de arcos, se crea un diccionario V tal que $V[i] = \{j \in N | (i, j) \in A\}$. Crear este diccionario tiene no requiere más de $\mathcal{O}(m)$ en memoria, pero ayuda a los algoritmos al momento de reducir los arcos salientes de los nodos seguros agregados al árbol de la solución, en vez de revisar los m arcos existentes.

- La matriz de costos se implementó como un simple diccionario: no se implementó como un arreglo de numpy ya que no es posible asignar $\mathcal{O}(n^2)$ posiciones de memoria para las instancias mayores, pero es posible usar un diccionario que usa sólo $\mathcal{O}(m)$ de memoria debido a la baja densidad de las instancias. Por ejemplo, la instancia de 211212 nodos sólo tiene 1048553 arcos, que se pueden guardar en un diccionario de 40960 kilobytes. En caso de redes altamente densas o de tamaño aún mayor, se podría mantener la matriz en un archivo o BDD externa y al momento de elegir un nodo seguro, cargar en memoria sólo la información correspondiente a V[nodo seguro].

- Se guardan los costos y predecesores de los nodos en arreglos de numpy de tamaño $n+1$, dejando el índice 0 desocupado para que el número de los nodos coincidan con su posición.

b Dijkstra

Para optimizar la elección del nodo seguro, se mantiene la información de los nodos que aún no entran en un heap binario.

c Dial

Para disminuir el uso de memoria, se usa la implementación con $C + 1$ buckets propuesta en la sección 4.6 del libro Network Flows, de Ahuja, Magnanti y Orlin.

2 Desempeño

En esta sección se presentan la comparación en cuanto a tiempo de resolución de los métodos en diversas instancias. Los experimentos fueron realizados en un computador con procesador Intel i5-3570k con 12 gigas de RAM. Las instancias serán representados con las siguientes claves numéricas:

1. 6 nodos, 9 arcos
2. 30 nodos, 384 arcos
3. 30 nodos, 104 arcos
4. 250 nodos, 249 arcos
5. 6600 nodos, 19444 arcos
6. 53000 nodos, 563902 arcos
7. 211212 nodos, 1048553 arcos

La Tabla 1 muestra los tiempos de ejecución de las instancias. Para cada método se reporta el tiempo promedio, mínimo y máximo de ejecución tras resolver el problema de rutas mínimas para cada posible nodo fuente. Se destaca que en las primeras 3 instancias no hay diferencias significativas, pero al aumentar el tamaño de los grafos utilizados la implementación de Dial presenta tiempos de resolución menores, lo que es consistente con la motivación de su formulación: tener que elegir un único nodo seguro para entrar de Dijkstra original, aún utilizando una lista ordenada mediante heap, no es eficiente al aumentar la cantidad de nodos y arcos. Desde la instancia 6 se ve que la implementación de Dial es dramáticamente superior al algoritmo de Dijkstra.

	Dijkstra			**Dial**		
Instancia	Promedio	Min	Max	Promedio	Min	Max
1	0.0022	0.000	0.0042	0.0039	0.000	0.0156
2	0.0038	0.000	0.0156	0.0038	0.000	0.0156
3	0.0032	0.000	0.0190	0.0034	0.000	0.0156
4	0.0191	0.0156	0.031	0.0079	0.000	0.0171
5	4.8829	0.0448	5.574	0.4101	0.050	0.5680
6	1097.31	1082.65	1110.79	28.15	27.64	31.17

Table 1: Comparación tiempos de resolución. Todos los valores reportados en segundos.