Diseño y Análisis de Algoritmos

Práctica 1

Algoritmos básicos en grafos:
Trivial Graph Format. Dijkstra. Librería Networkx.

Andrián Navas Ajenjo
Gloria del Valle Cano
adrian.navas@estudiante.uam.es & gloria.valle@estudiante.uam.es

Grado en Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
18-10-2019

Índice general

Índice de figuras

1. Introducción

En la primera práctica hemos implementado algoritmos básicos para conseguir los caminos mínimos de grafos ponderados, además de trabajar con la representación de grafos en diferentes formatos como `Pickle`, `JSON` o `TGF`. Para ello hemos utilizado dos estructuras de datos diferentes para la representación de los grafos, en concreto, una matriz `numpy` de adyacencia y un diccionario de diccionarios.
Asimismo, abordaremos los problemas propuestos con la representación de gráficas que nos ayuden a realizar un análisis de los tiempos de los algoritmos, tanto de la implementación de Dijkstra propuesta como de la incluida en la librería de `NetworkX` con el fin de abstraer la complejidad y la estrategia de los algoritmos.

2. Cuestiones

Cuestiones sobre guardado de grafos

Cuestión 1

Describir qué se entiende por serializar un objeto Python.
Serializar un objeto es el proceso de convertirlo a un formato específico, a través de un protocolo binario, mediante el cual pueda ser almacenado en disco de forma persistente y posteriormente cargado desde el fichero en el mismo estado en el que se guardó.

Cuestión 2

`JSON` es otro formato de serializacion de objetos. Comentar brevemente posibles diferencias entre `Pickle` y `JSON`.
La diferencia principal entre `Pickle` y `JSON` es que `Pickle` guarda los objetos en binario mientras que `JSON` guarda los objetos de forma legible mediante *atributos* $\rightarrow$ *valor*. Un ejemplo de `Pickle` para guardar la lista *[1,2,3,4]* sería:

```
]q (KKKKe
```

Y en `JSON` podría ser por ejemplo:

```
{
    lista: [1,2,3,4]
}
```

Además, `Pickle` sólo se puede usar en Python, mientras que `JSON` es un estándar de serialización por el que pueden ser accedidos todos los lenguajes de programación.
Otro aspecto a destacar entre ellos es que la implementación de `Pickle` no es tan eficiente como la de `JSON`, por lo que éste llega a ser más rápido.
Por último, `Pickle` puede ser menos seguro que `JSON`, ya que el primero puede ejecutar código abierto almacenado en memoria, mientras que `JSON` tiene mecanismos de seguridad integrados.

Cuestión 3

¿Qué ventajas e inconvenientes tendrían las funciones `Pickle` sobre las construidas mediante el formato `TGF`? Responder algo pertinente y no con lugares comunes.

- Ventajas:
  1. La escritura y lectura es más rápida ya que se escribe el objeto directamente y no es necesario procesarlo para su escritura ni lectura.
  2. La forma de escritura y lectura de `Pickle` es mucho más sencilla que la escritura-lectura en formato `TGF`, ya que para escribir y leer en este formato es necesario procesar el grafo e ir escribiendo en el formato adecuado mientras que `Pickle` vuelca el objeto directamente.
  3. El formato de `Pickle` es más seguro ya que al guardar la información en binario, no puede ser accedida por cualquier persona en cualquier momento.
- Desventajas:
  1. No es legible para personas el fichero escrito, por lo que para leerlo es necesario utilizar la librería `Pickle` junto con un script de Python.
  2. El formato `TGF` ocupa menos espacio en disco que el formato `Pickle` (para un mismo grafo pequeño de ejemplo, el objeto en formato `Pickle` ocupa 107 bytes y en formato `TGF` y comprimido, ocupa 272 bytes). Esto para grafos grandes y a nivel de rendimiento podría suponer un gran problema de almacenamiento.
  3. `Pickle`, al ser una librería de Python, hace que los grafos volcados a través de ella no sean accesibles a través de programas escritos en otros lenguajes.

Cuestiones sobre Dijkstra

Cuestión 1

¿Cuál es el coste teórico del algoritmo de Dijkstra? Justificar brevemente dicho coste.
Para grafos, la complejidad de Dijkstra es de:
Para $|E| =$ *número de aristas* y $|V| =$ *número de nodos o vértices*:

$$O(|E| + |V| \log |V|) \tag{2.1}$$

La cual sale de las operaciones básicas del algoritmo, ya que se insertará y se obtendrá cada nodo (esto tiene un coste logarítmico) E veces.
Si el grafo es denso, es decir, que el número de ramas es muy similar al número de vértices al cuadrado, $|E| = O(|V|^2)$. Por tanto la complejidad de Dijkstra es:

$$O(|V|^2 \log |V|) \tag{2.2}$$

Cuestión 2

Expresar el coste de Dijkstra en función del número de nodos y el *sparse factors* ρ del grafo en cuestión. Para un número de nodos fijo adecuado, ¿cuál es el crecimiento del coste de Dijkstra en función de ρ?
Ilustrar este crecimiento ejecutando Dijkstra sobre listas de adyacencia de grafos con un número fijo de nodos y sparse factors 0.1, 0.3, 0.5, 0.7, 0.9 y midiendo los correspondientes tiempos de ejecución.

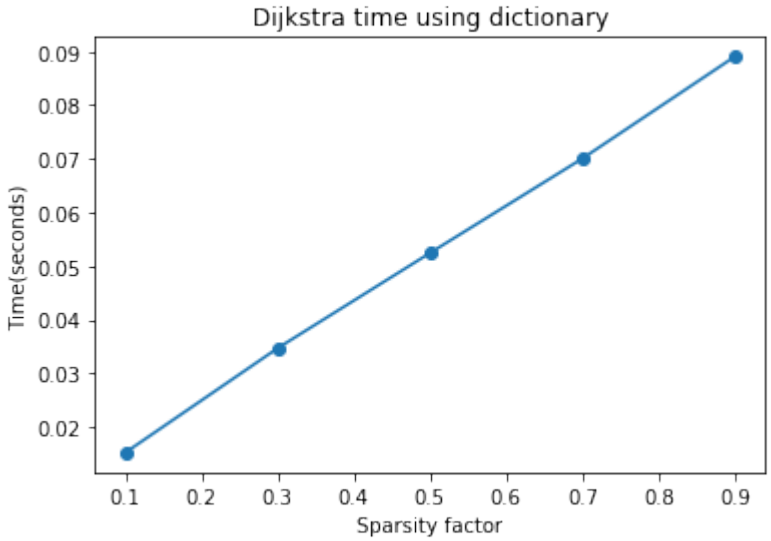

Figura 2.1: Cuestión 2 sobre Dijkstra

Como podemos observar en la gráfica, el tiempo de ejecutar Dijkstra sobre todos los nodos sobre listas de adyacencia variando el *sparse factor* es lineal en función a ρ. En dicha gráfica hemos mostrado los puntos del

tiempo en segundos que tarda Dijkstra en ejecutar en todos los nodos de un grafo (siempre de 100 nodos y con el tiempo medido promediando el tiempo de 10 grafos), y además hemos mostrado la línea que une dichos puntos para observar la trayectoria del tiempo y como hemos indicado anteriormente, se puede comprobar que el crecimiento es claramente lineal en función a ρ.

Cuestión 3

¿Cuál es el coste teórico del algoritmo de Dijkstra iterado para encontrar las distancias mínimas entre todos los vértices de un grafo?
Contando con que el coste de Dijkstra para encontrar los caminos más cortos de un nodo es de $O(|V|^2 \log|V|)$, el coste de ejecutar Dijkstra para todos los $|V|$ nodos que hay en el grafo sería $O(|V|^3 \log|V|)$.
Esto es debido a que queremos calcular todas las distancias, por tanto el número de veces crece exponencialmente.

Cuestión 4

¿Cómo se podrían recuperar los caminos mínimos si se utiliza Dijkstra iterado?
Para recuperar los caminos mínimos una vez obtenidos los caminos mínimos de cada nodo aplicando Dijkstra a cada uno de ellos, debemos utilizar las listas de padres guardadas en cada una de las iteraciones anteriores, siguiendo la lista de padres de cada uno de ellos podemos obtener el camino mínimo desde cada nodo hasta los demás.

Cuestiones sobre Networkx

Cuestión 1

Muestra gráficamente el crecimiento de los tiempos de ejecución del algoritmo de Dijkstra como una función que muestra el número de nodos en función del *sparse factor* usando la librería `NetworkX`.

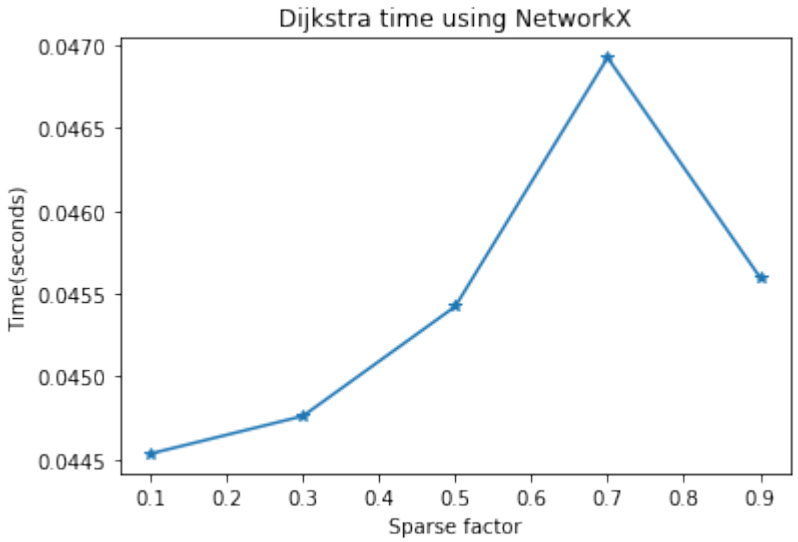

Figura 2.2: Cuestión 1 sobre NetworkX

Gráfica de tiempo de ejecución de Dijkstra sobre grafos utilizando `NetworkX` con grafos de 100 nodos. Cómo podemos observar, el crecimiento es también lineal en función del *sparse factor* a excepción del *sparse factor* 0.9.
Creemos que esto guarda una íntima relación con el principio de cercanía de un grafo. Entendemos que los nodos con un mayor factor están conectados a muchos nodos, que a su vez están mucho más conectados, por lo que pueden ser más accesibles. En un caso real podrían ser buenos para extender información. Esto tal vez podría justificar de alguna manera que pudiese tardar cada vez menos a partir del punto 0.7, simulando un comportamiento al de su predecesor 0.5.

Cuestión 2

Mide y muestra gráficamente los tiempos de ejecución de Dijkstra iterado para encontrar la distancia mínima entre todos los vértices de un grafo utilizando la librería de `NetworkX` y trabajando en grafos con un número fijo de 25 nodos y *sparse factors* de *0.1, 0.3, 0.5, 0.9.*

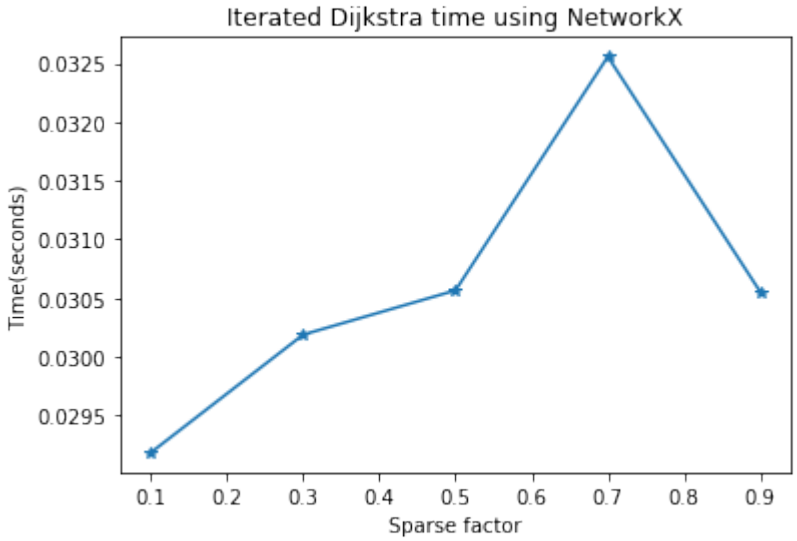

Figura 2.3: Cuestión 2 sobre NetworkX

Cómo podemos observar, el tiempo en segundos de ejecución de Dijkstra iterado tiene un crecimiento lineal en función del *sparse factor* indicado a excepción del *sparse factor* 0.9, justificando con el mismo argumento mencionado en la cuestión previa.

3. Conclusión

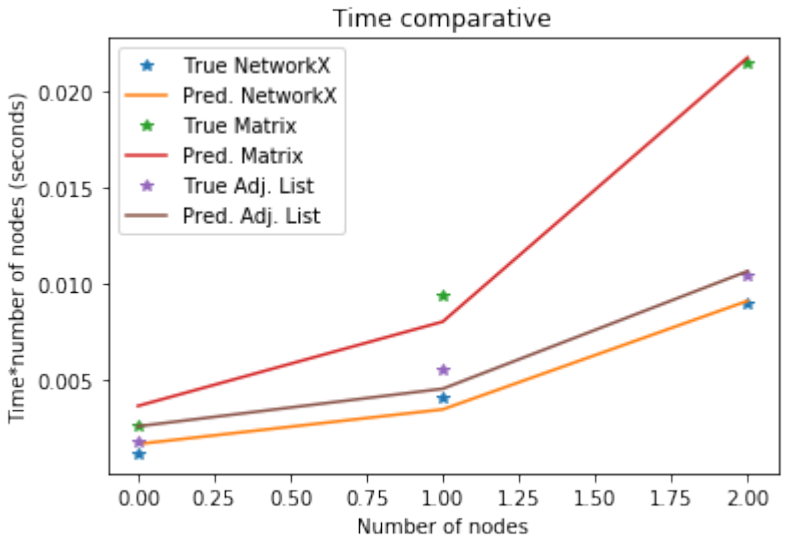

Figura 3.1: Comparación de tiempos de Dijkstra

Finalmente, mostramos una comparativa de los tiempos de ejecución de los algoritmos propuestos, mostrando cada uno de ellos la predicción que obtenemos del entrenamiento frente a los tiempos reales obtenidos. Vemos que la predicción se acerca bastante a los datos, y que además crece de manera exponencial a medida que aumentamos el número de nodos. También podemos comprobar que el caso más eficiente podría ser utilizar el algoritmo de Dijkstra implementado en la librería de `NetworkX` frente a los implementados por nosotros. Sí que cabe destacar que la alternativa que a nivel computacional puede resultar mejor sería la ofrecida utilizando la lista de adyacencia en lugar de la matriz. Por tanto, entendemos que la librería de `NetworkX` está fuertemente capacitada y optimizada para manipular grafos y que la implementación aportada por nosotros de Dijkstra nos ha servido para comprender el algoritmo y realizar un análisis del mismo tanto de funcionamiento como de rendimiento.