**Data Structure**

Catedrático: Fernandojosé Boiton Tello

Catedrático auxiliar: Luis Angel Tórtola Tejeda

Sharon Anesveth A. Maatens (20190339)

**Longest Path**

En teoría de grafos, el problema del camino más largo es, dado un grafo, encontrar un camino simple de longitud máxima. Una ruta es simple si no contiene ningún vértice más de una vez. La longitud de una ruta se puede definir como el número o el acumulado peso de sus bordes. LP es NP-completo.

1. **A Streaming Algorithm for the Undirected Longest Path Problem**

Algoritmo de transmisión para el problema de ruta más largo en grafos no dirigidos. el grafo de entrada se proporciona como una secuencia de bordes y la RAM se limita a sólo un número lineal de bordes en un tiempo (lineal en el número de vértices n).

Funciona en dos fases. En la primera fase, información global en el grafo se recopila en forma de un número constante de árboles de expansión (árboles conectados) T1,. . . , Tτ. Esto es posible en el modelo de transmisión ya que, en términos generales, para un árbol de expansión podemos ‘tomar bordes como vienen ". Se puede construir un árbol de expansión en una sola pasada; sin embargo, utiliza múltiples pases y se limita el grado máximo durante los primeros pases para favorecer las estructuras de camino y evitar grupos de bordes. Los experimentos indican claramente que esta limitación de grado es esencial para la calidad de la solución. Los árboles de expansión se ajustan a la RAM, ya que consideramos τ como constante (de hecho tendremos τ = 1 o τ = 2 en los experimentos). Después de la construcción de τ árboles, se fusionan en un grafo U tomando la unión de sus bordes. Entonces usamos algoritmos estándar para determinar un long path P en U, aislar P y finalmente agregar suficiente bordes alrededor de P para obtener un árbol T.

Luego, en la segunda fase, realizamos más pases durante los cuales probamos si el intercambio de bordes individuales de T puede mejorar el longest path en él. (Se puede encontrar un longest path en un árbol realizando DFS dos veces [10]; la longitud de un longest path en un árbol es su diámetro).

El principal desafío en la segunda fase es determinar rápidamente qué bordes deben intercambiarse. Este algoritmo muestra que esta decisión se puede tomar en tiempo lineal, por lo tanto, se obtiene un procesamiento por borde tiempo de O (n).

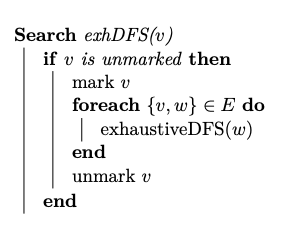
1. **Parallel Optimal Longest Path Search**

Esta sección presenta un algoritmo llamado "Ruta más larga por programación dinámica" (LPDP). El algoritmo resuelve el problema de ruta más larga (LP) para grafos ponderados no dirigidos. LPDP se basa en los principios de la programación dinámica.

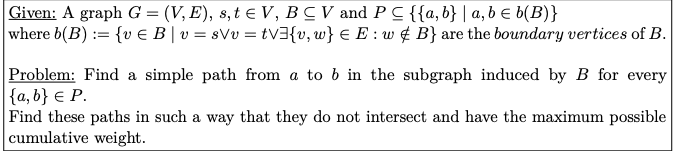
Una forma sencilla de resolver el problema del camino más largo es la búsqueda exhaustiva de profundidad primero. En la búsqueda regular de profundidad primero (DFS), un vértice tiene dos estados: marcado y sin marcar. Inicialmente, todos los vértices no están marcados. La búsqueda comienza llamando al procedimiento DFS con un determinado vértice como parámetro. Este vértice se llama raíz. El vértice actual (el parámetro de la llamada DFS actual) se marca y luego el procedimiento DFS se ejecuta recursivamente en cada vértice no marcado alcanzable por un borde desde el vértice actual. El vértice actual es llamado el padre de estos vértices. Una vez que finalizan las llamadas recursivas DFS, retrocedemos al vértice padre.

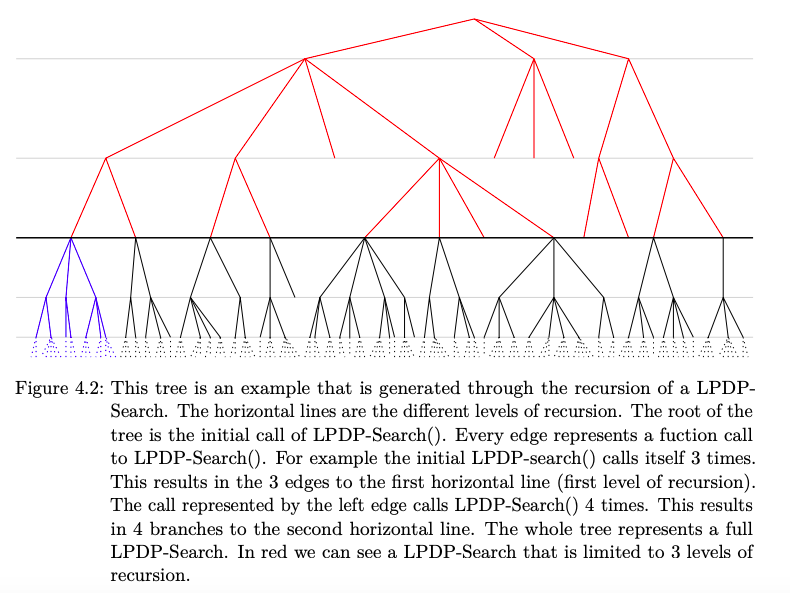
La búsqueda finaliza una vez que DFS retrocede desde el vértice raíz. De esa manera, cada ruta simple en el grafo a partir de la raíz se explora el vértice.

El problema de LP se puede resolver con DFS exhaustivo utilizando el inicio vértice como la raíz. Durante la búsqueda, la longitud de la ruta actual se almacena y compara a la mejor solución anterior cada vez que se alcanza el vértice objetivo. Si la longitud actual es mayor que la de la mejor solución, se actualiza en consecuencia. Cuando la búsqueda se ha encontrado una ruta con una longitud máxima de s a t. Si almacenamos la longitud de la ruta más larga para cada vértice (no solo el vértice objetivo), entonces las rutas largas simples desde s hasta cualquier otro vértice se pueden calcular simultáneamente.



La programación dinámica requiere que podamos dividir LP en subproblemas. A fin de que hacer esto primero generalizamos el problema:



Para hacer esto simultáneamente a través de la “paralelización” (inspirado en el "Cubo y conquista" enfoque para la solución SAT presentado por Heule et al.) la fórmula SAT se divide en muchas sub fórmulas. Estas subformulas pueden resolverse en paralelo. Podemos hacer lo mismo para LPDP dividiendo el espacio de búsqueda de LPDP-Search en muchas ramas disjuntas. Hacemos esto ejecutando LPDP-Search desde cada límite vértice con una profundidad de recursión limitada. Cada vez que la búsqueda alcanza un cierto nivel de recursividad la búsqueda almacena su contexto actual en una lista y vuelve a la recursividad anterior nivel.

**Shortest Path**

En la teoría de grafos, el problema del camino más corto es el problema que consiste en encontrar un camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima.

1. **SLF algorithm: The Small Label First algorithm (SLF)**

Mientras el algoritmo genérico se basa en la verificación iterativa de los bordes desde el vértice considerado y en la configuración de la etiqueta para vertexj, en el que un borde dado termina, a dj = di + aij, en el caso cuando dj> di + aij almacenando los vértices que deben verificarse, utiliza la lista Vis, llamada lista de candidatos. La forma en que los vértices se almacenan en esta lista, así como el método que determina la adición y la recuperación de vértices hacia y desde él, es con frecuencia el factor principal que distingue los algoritmos individuales en consideración. En el caso del algoritmo genérico, la lista de candidatos es una cola FIFO en la que se realizan operaciones de adiciones y recuperación de un vértice al final del mismo o desde su cabeza, respectivamente.

El algoritmo Small Label First (SLF) busca gestionar la lista de candidatos de tal manera que se hagan avisos con etiquetas pequeñas ubicadas lo más cerca posible del encabezado de la lista. La razón de esta operación es el hecho de que cuanto más pequeña es la etiqueta de un vértice que se recupera de la lista de candidatos, menor es la probabilidad de que este vértice se reenvíe a la lista una vez más. Este algoritmo, al igual que los dos algoritmos siguientes, intenta alcanzar la operación característica del algoritmo de Dijkstra con un desembolso computacional más bajo.

En este método, un vértice es incluido en el frente de una etiqueta si su etiqueta es más pequeña que la etiqueta en el frente de un deque. Hasta cierto punto, este método incluye el criterio principal de los métodos de establecimiento de etiquetas. Los últimos métodos siempre recuperan el elemento mínimo de la lista; el método Small Label First (SLF) pone un vértice con la etiqueta más pequeña que la etiqueta del vértice frontal en la parte superior. El enfoque puede llevar a la conclusión lógica al exigir que se incluye cada vértice en la lista según su rango para que la deque se convierta en una cola prioritaria y el método resultante se convierte en una versión de corrección de etiqueta del algoritmo de Dijkstra.

**VENTAJA:** Menor gasto computacional respecto al algoritmo genérico y el más veloz de los algoritmos tradicionales de Shortest Path.

**APLICACIONES:** Social Network Analysis: calcular centralidad entre otros.

**PSEUDOCODIGO:**

procedure Shortest-Path-Faster-Algorithm(*G*, *s*)

1 for each vertex *v* ≠ *s* in *V*(*G*)

2 d(*v*) := ∞

3 d(*s*) := 0

4 offer *s* into *Q*

5 while *Q* is not empty do

6 *u* := poll *Q*

7 for each edge (*u*, *v*) in *E*(*G*) do

8 if d(*u*) + w(*u*, *v*) < d(*v*) then

9 d(*v*) := d(*u*) + w(*u*, *v*)

10 if *v* is not in *Q* then

11 procedure Small-Label-First(*G*, *Q*)

12 if d(back(*Q*)) < d(front(*Q*)) then

13 u := pop back of *Q*

14 push u into front of *Q*

1. **A Generalized Threshold Algorithm for the Shortest Path Problem with Time Windows**

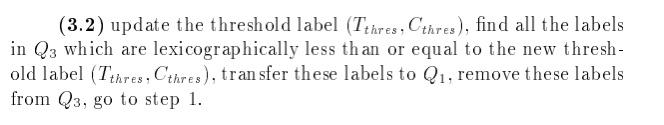
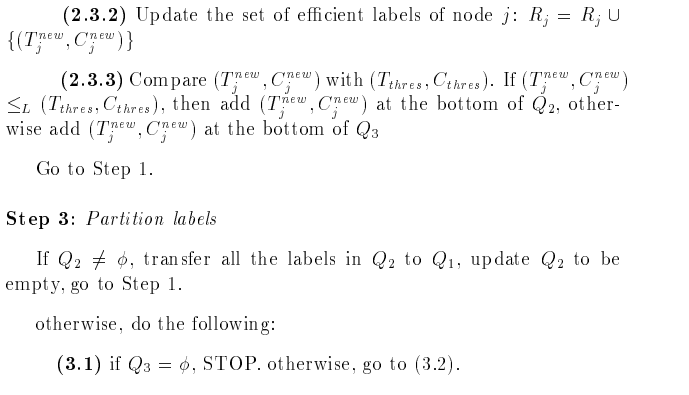
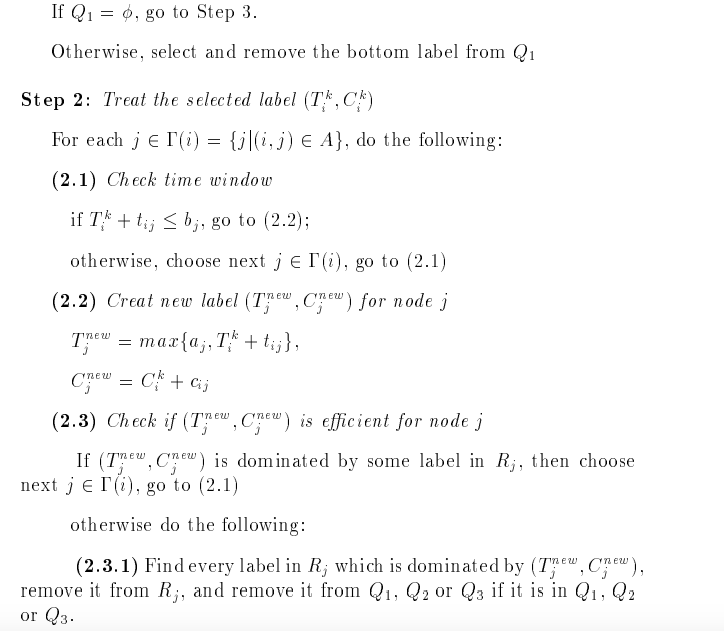
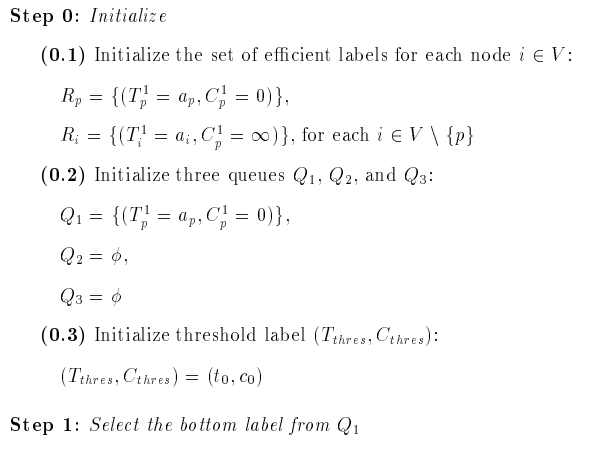
Algoritmo de etiquetado para el problema de ruta más corta con ventanas de tiempo (SPPTW). Se generaliza desde el algoritmo Threshold para el problema del camino más corto sin restricciones.

El método divide el conjunto de nodos etiquetados en dos subconjuntos, NOW y NEXT, que se mantienen como colas FIFO. Al comienzo de cada iteración del algoritmo, NOW está vacío. El método también mantiene un parámetro de umbral t que se establece en un promedio ponderado de las etiquetas de distancia mínima y promedio de los nodos en NEXT. Durante una iteración, el algoritmo transfiere los nodos v con d (v) 5 t de NEXT a NOW y escanea los nodos en NOW. Los nodos que se etiquetan durante la iteración se agregan a NEXT. El algoritmo termina cuando NEXT está vacío al final de una iteración. El código THRESH implementa el algoritmo de Threshold con los valores de parámetro MINWT y WTCNG. Si la función de longitud no es negativa, THRESH se ejecuta en tiempo O (nm).

**VENTAJA:** Supera en un 40% la velocidad de los algoritmos que usan labels/etiquetas.

**APLICACIONES:** Shortest path con Time Windows, servidores horarios.

**PSEUDOCODIGO:**



1. **Dijkstra’s Algorithm with Arc-Flags (Fast Point-to-Point Shortest Path Computations with Arc-Flags)**

En el problema de la ruta más corta punto a punto (P2P) se tiene que encontrar una ruta más corta entre dos nodos especificados en un grafo de entrada. Un algoritmo estándar para este problema es el desarrollado por Dijkstra (1959) que se ejecuta en tiempo O (m + n log n) (Fredman y Tarjan, 1987). Sin embargo,en la práctica, las técnicas de aceleración pueden reducir el tiempo de ejecución y a menudo resultan en una ejecución sublineal respecto al tiempo. Dependen crucialmente del hecho de que el algoritmo de Dijkstra establece etiquetas y que puede ser terminado cuando se liquida el nodo de destino. Por lo tanto, el algoritmo no necesariamente busca todo el grafo.

Si permitimos un paso de preprocesamiento, el tiempo de ejecución se puede reducir aún más con el siguiente insight: considere, para cada arco a, el conjunto Sa de nodos a los que se puede llegar por una ruta más corta que comienza con a. Es fácil verificar que el algoritmo de Dijkstra se puede restringir al subgrafo con esos arcos para cuál es el nodo de cola t en Sa. Sin embargo, almacenar todos los conjuntos Sa requiere O (n^2) espacio, lo cual es prohibitivo para grafos grandes (n >> 1M). Por lo tanto, se utiliza una partición del conjunto de nodos V en p (: = | R |) regiones para una aproximación del conjunto Sa. Formalmente, usaremos una función r : V → {1,. . . , p} que asigna a cada nodo el número de su región. Ahora usaremos un vector de bandera fa: {1,. . . , p} →{true, false} con p entradas, cada una de las cuales corresponde a una región. Para cada arco a, establecemos la entrada fa (i) a true, si a es el comienzo de cualquier ruta más corta a al menos un nodo en la región i ∈ {1,. . . , p}. Además, para cada arco (v, w) con v, w ∈ V configuramos la entrada de bandera f (v, w) (rw) a true.

Para una consulta de ruta más corta específica de sa t, el algoritmo de Dijkstra puede restringirse al subgrafo Gt inducido por aquellos arcos donde la entrada de la bandera corresponde a la región objetivo (la región donde t pertenece a) es true.

**Bibliografía**

* https://drops.dagstuhl.de/opus/volltexte/2016/6398/pdf/LIPIcs-ESA-2016-56.pdf
* https://publikationen.bibliothek.kit.edu/1000104171/51132444
* <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.2020&rep=rep1&type=pdf>
* <http://users.diag.uniroma1.it/challenge9/papers/kohler.pdf>
* <https://app.slack.com/client/TRX5ZFTRQ/CRX5ZFXA6?cdn_fallback=1>
* <https://dl.acm.org/doi/pdf/10.5555/314464.314638>
* <https://www.researchgate.net/publication/242013156_Efficiency_Evaluation_of_Shortest_Path_Algorithms>
* <https://www.researchgate.net/publication/242013156_Efficiency_Evaluation_of_Shortest_Path_Algorithms>