Minimum bottleneck spanning tree

From Wikipedia, the free encyclopedia

In mathematics, a **minimum bottleneck spanning tree** (MBST) in an undirected graph is a spanning tree in which the most expensive edge is as cheap as possible. A bottleneck edge is the highest weighted edge in a spanning tree. A spanning tree is a minimum bottleneck spanning tree if the graph does not contain a spanning tree with a smaller bottleneck edge weight.[1] For a directed graph, a similar problem is known as **Minimum Bottleneck Spanning Arborescence** (MBSA).

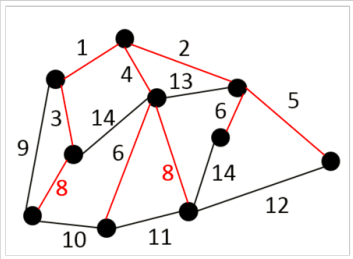
Contents

Definitions

Undirected graphs

In an undirected graph $G(V, E)$ and a function $w: E \to R$, let S be the set of all spanning trees T_i. Let $B(T_i)$ be the maximum weight edge for any spanning tree T_i. We define subset of minimum bottleneck spanning trees S such that for every $T_j \in S$ and $T_k \in S$ we have $B(T_j) \leqslant B(T_k)$ for all i and k.[2]

The graph on the right is an example of MBST, the red edges in the graph form a MBST of $G(V, E)$.

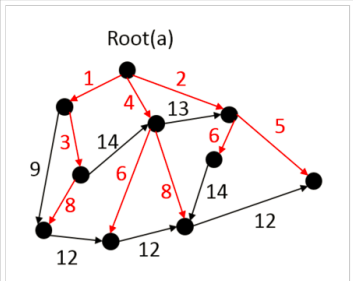


Minimal Bottleneck Spanning Tree $G(V,E)$

Directed graphs

An arborescence of graph G is a directed tree of G which contains a directed path from a specified node L to each node of a subset V' of $V \setminus \{L\}$. Node L is called the root of arborescence. An arborescence is a spanning arborescence if $V' = V \setminus \{L\}$. MBST in this case is a spanning arborescence with the minimum bottleneck edge. An MBST in this case is called a Minimum Bottleneck Spanning Arborescence (MBSA).

The graph on the right is an example of MBSA, the red edges in the graph form a MBSA of $G(V, E)$.



Minimal Bottleneck Spanning Arborescence $G(V,E)$

Properties

A MST (or minimum spanning tree) is necessarily a MBST, but a MBST is not necessarily a MST.

Camerini's algorithm for undirected graphs

Camerini proposed[3] an algorithm used to obtain a minimum bottleneck spanning tree (MBST) in a given undirected, connected, edge-weighted graph in 1978. It half divides edges into two sets. The weights of edges in one set are no more than that in the other. If a spanning tree exists in subgraph composed solely with edges in smaller edges set, it then computes a MBST in the subgraph, a MBST of the subgraph is exactly a MBST of the original graph. If a spanning tree does not exist, it combines each disconnected component into a new super vertex, then computes a MBST in the graph formed by these super vertices and edges in the larger edges set. A forest in each disconnected component is part of a MBST in original graph. Repeat this process until two (super) vertices are left in the graph and a single edge with smallest weight between them is to be added. A MBST is found consisting of all the edges found in previous steps.[4]

Pseudocode

The procedure has two input parameters. G is a graph, w is a weights array of all edges in the graph G.[5]

```
1  function MBST(graph G, weights w)
2  E ← the set of edges of G
3  if | E | = 1 then return E else
4     A ← half edges in E whose weights are no less than the median weight
5     B ← E − A
6     F ← forest of G_B
7     if F is a spanning tree then
8        return MBST(G_B, w)
9     else
10       return MBST((G_A)_η, w) ∪ F
```

In the above $(G_A)_\eta$ is the subgraph composed of super vertices (by regarding vertices in a disconnected component as one) and edges in A.

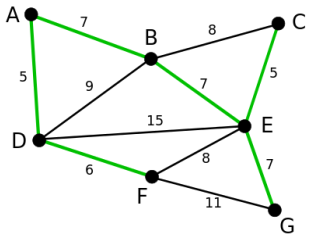
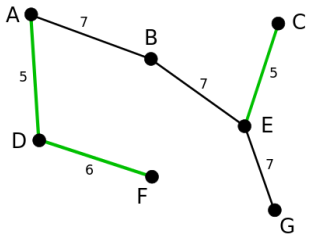
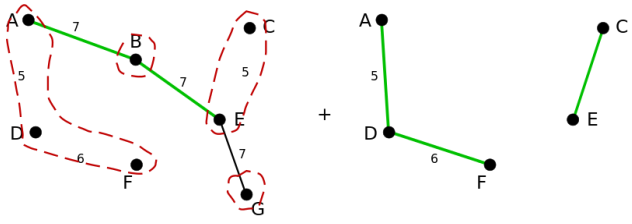
Running time

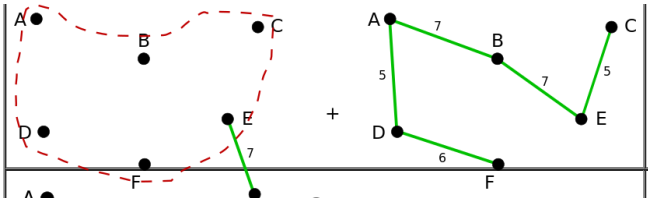
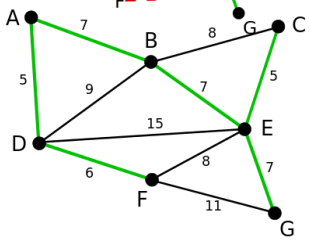
The algorithm is running in $O(E)$ time, where E is the number of edges. This bound is achieved as follows:

- dividing into two sets with median-finding algorithms in $O(E)$
- finding a forest in $O(E)$
- considering half edges in E in each iteration $O(E + E/2 + E/4 + \cdots + 1) = O(E)$

Example

In the following example green edges are used to form a MBST and dashed red areas indicate super vertices formed during the algorithm steps.

	The algorithm half divides edges in two sets with respect to weights. Green edges are those edges whose weights are as small as possible.
	Since there is a spanning tree in the subgraph formed solely with edges in the smaller edges set. Repeat finding a MBST in this subgraph.
	Since there is not a spanning tree in current subgraph formed with edges in the current smaller edges set. Combine the vertices of a disconnected component to a super vertex (denoted by a dashed red area) and then find a MBST in the subgraph formed with super vertices and edges in larger edges set. A forest formed within each disconnected component will be part of a MBST in the original graph.

	Repeat similar steps by combing more vertices into a super vertex.
	The algorithm finally obtains a MBST by using edges it found during the algorithm.

MBSA algorithms for directed graphs

There are two algorithms available for directed graph: Camerini's algorithm for finding MBSA and another from Gabow and Tarjan.[4]

Camerini's algorithm for MBSA

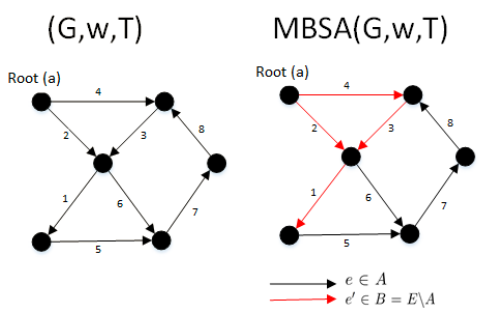
For a directed graph, Camerini's algorithm focuses on finding the set of edges that would have its maximum cost as the bottleneck cost of the MBSA. This is done by partitioning the set of edges E into two sets A and B and maintaining the set T that is the set in which it is known that G_T does not have a spanning arborescence, increasing T by B whenever the maximal arborescence of $G(B \cup T)$ is not a spanning arborescence of G, otherwise we decrease E by A. Total total time complexity is $O(E \log E)$.[3][4]

Pseudocode

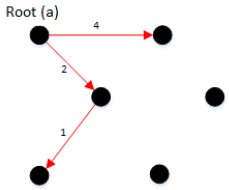
```
1  function MBSA(G, w, T)
2  E ← the set of edges of G;
3  if | E - T | > 1 then
4     A ← UH(E-T);
5     B ← (E - T) - A;
6     F ← BUSH(G_BUT);
7     if F is a spanning arborescence of G then
8        S ← F; MBSA((G_BUT),w,T);
9     else
10       MBSA(G,w,TUB);
```

1. T represents a subset of E for which it is known that G_T does not contain any spanning arborescence rooted at node "a". Initially T is empty
2. UH takes (E-T) set of edges in G and returns A ⊂ (E-T) such that:
   1. $|A| = \left\lfloor \frac{(|E-T|)}{2} \right\rfloor$
   2. $W_a >= W_b$, for a ∈ A and b ∈ B
3. BUSH(G) returns a maximal arborescence of G rooted at node "a"
4. The final result will be S

Example

	After running the first iteration of this algorithm, we get the F and F is not a spanning arborescence of G, So we run

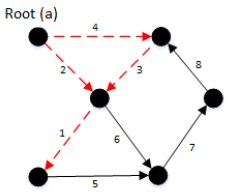
F



F is not a spanning arborescence of G

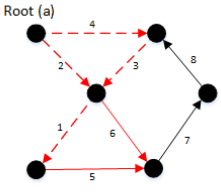
the code $MBSA(G, w, T \cup B)$.

(G,w,T'=(T U B))



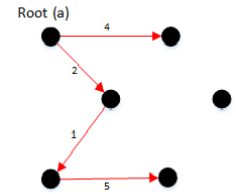
$e \in A$
e' ∈T U B

MBSA(G,w,T' = (T U B))



e ∈A'
e'' ∈T U B = T'
e' ∈B'

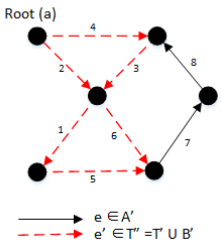
F'



F' is not a spanning arborescence of G

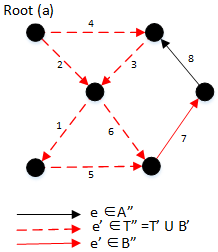
In the second iteration, we get the F' and F' is also not a spanning arborescence of G, So we run the code $MBSA(G, w, T' \cup B')$

(G,w,T'' = T' U B')



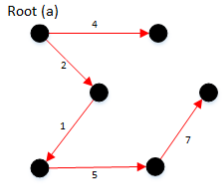
e ∈A'
e' ∈T'' =T' U B'

MBSA(G,w,T'' = T' U B')



e ∈A''
e' ∈T'' =T' U B'
e' ∈B''

F''

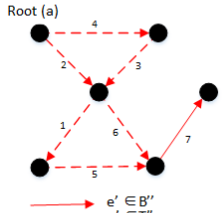


e' ∈T'' U B''

S=F''

In the third iteration, we get the F'' and F'' is a spanning arborescence of G, So we run the code $MBSA(G_{T'' \cup B''}, w, T'')$

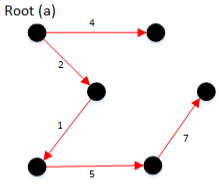
MBSA($(G_{B''UT''}$,w,T''))



e' ∈B''
e' ∈T''

|E-T|=1, return

F''



when we run the $MBSA(G_{T'' \cup B''}, w, T'')$, the $|E - T|$ is equal to 1, which is not larger than 1, so the algorithm return and the final result is $S = F''$.

Gabow and Tarjan algorithm for MBSA

Gabow and Tarjan provided a modification of Dijkstra's algorithm for single-source shortest path that produces an MBSA. Their algorithm runs in $O(E + V \log V)$ time if Fibonacci heap used.[6]

Pseudocode

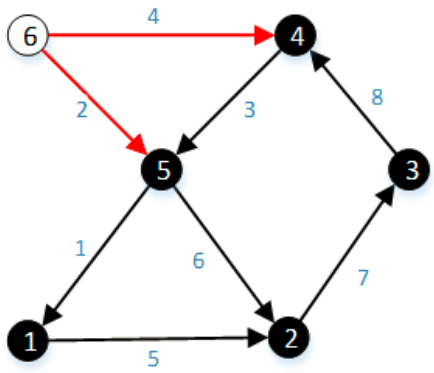
```
For a graph G(V,E), F is a collection of vertices in V.
Initially, F = s where s is the starting point of the graph G and c(s) = ∞

1  function MBSA-GT(G, w, T)
2    Select v with minimum c(v) from F;
3    Delete it from the F;
4    for ∀ edge(v,w) do
5      if w ∉ F or ∉ Tree then
6        add w to F;
7        c(w) = c(v,w);
8        p(w) = v;
9      else
10       if w  F and c(w) > c(v,w) then
11         c(w) = c(v,w);
12         p(w) = v;
```

Example

The following example shows that how the algorithm works.

F

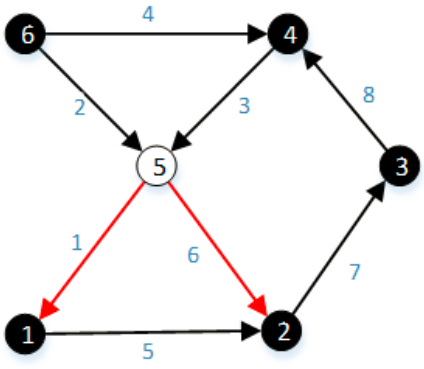
Vertex X	C(v)	P(v) [parent of v]
6	-∞	-
5	2	6
4	4	6



At the first step of the algorithm, we select the root s from the graph G, in the above figure, vertex 6 is the root s. Then we found all the edge(6,w) ∈ E and their cost c(6,w), where w ∈ V.

F

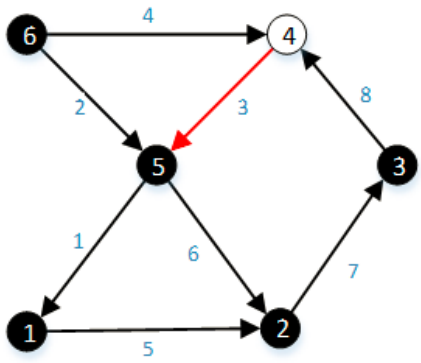
Vertex X	C(v)	P(v) [parent of v]
~~6~~	-∞	-
5	2	6
4	4	6
1	1	5
2	6	5



Next we move to the vertex 5 in the graph G, we found all the edge(5,w) ∈ E and their cost c(5,w), where w ∈ V.

F

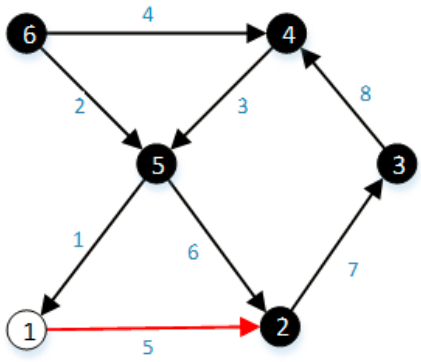
Vertex X	C(v)	P(v) [parent of v]
~~6~~	-∞	-
~~5~~	2	6
4	4	6
1	1	5
2	6	5
remove-> 5	3	4



Next we move to the vertex 4 in the graph G, we found all the edge(4,w) ∈ E and their cost c(4,w), where w ∈ V. We find that the edge(4,5) > edge(6.5), so we keep edge(6,5) and remove the edge(4,5).

F

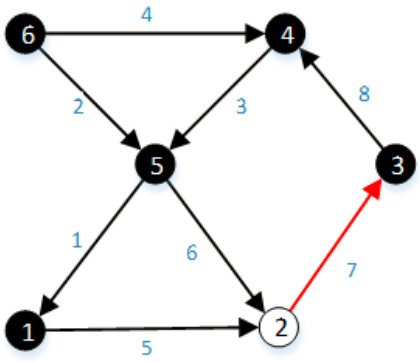
Vertex X	C(v)	P(v) [parent of v]
~~6~~	-∞	-
~~5~~	2	6
~~4~~	4	6
1	1	5
replace-> 2	6	5
2	5	1



Next we move to the vertex 1 in the graph G, we found all the edge(1,w) ∈ E and their cost c(1,w), where w ∈ V. We find that the edge(5,2) > edge(1,2), so we remove edge(5,2) and keep the edge(1,2).

F

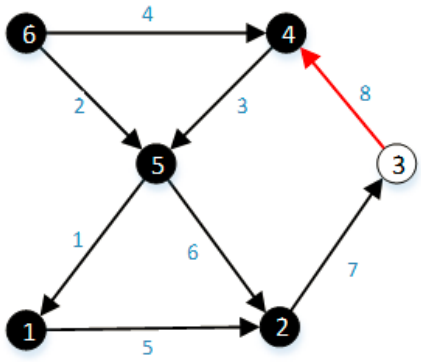
Vertex X	C(v)	P(v) [parent of v]
~~6~~	-∞	-
~~5~~	2	6
~~4~~	4	6
~~1~~	1	5
2	5	1
3	7	2



Next we move to the vertex 2 in the graph G, we found all the edge(2,w) ∈ E and their cost c(2,w), where w ∈ V.

F

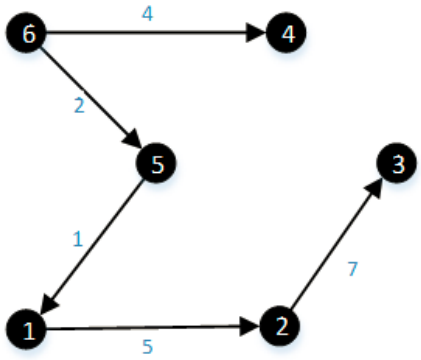
	Vertex X	C(v)	P(v) [parent of v]
	~~6~~	-∞	-
	~~5~~	2	6
	~~4~~	4	6
	~~1~~	1	5
	~~2~~	5	1
	3	7	2
remove->	4	8	3



Next we move to the vertex 3 in the graph G, we found all the edge(3,w) ∈ E and their cost c(3,w), where w ∈ V. We find that the edge(3,4) > edge(6,4), so we remove the edge(3,4) and keep the edge(6,4).

F

Vertex X	C(v)	P(v) [parent of v]
~~6~~	-∞	-
~~5~~	2	6
~~4~~	4	6
~~1~~	1	5
~~2~~	5	1
~~3~~	7	2
End		



After we loop through all the vertices in the graph G, the algorithm has finished.

Another approach proposed by Tarjan and Gabow with bound of $O(E \log^* V)$ for sparse graphs, in which it is very similar to Camerini's algorithm for MBSA, but rather than partitioning the set of edges into two sets per each iteration, $K(i)$ was introduced in which i is the number of splits that has taken place or in other words the iteration number, and $K(i)$ is an increasing function that denotes the number of partitioned sets that one should have per iteration. $K(i) = 2^{k(i-1)}$ with $k(1) = 2$. The algorithm finds λ^* in which it is the value of the bottleneck edge in any MBSA. After λ^* is found any spanning arborescence in $G(\lambda^*)$ is an MBSA in which $G(\lambda^*)$ is the graph where all its edge's costs are $\leq \lambda^*$.[4][6]

References

1. Everything about Bottleneck Spanning Tree (http://flashing-thoughts.blogspot.ru/2010/06/everything-about-bottleneck-spanning.html)
2. Murali, T. M. (2009), *Applications of Minimum Spanning Trees* (http://courses.cs.vt.edu/~cs5114/spring2009/lectures/lecture08-mst-applications.pdf) (PDF)
3. Camerini, P.M. (1978), "The min-max spanning tree problem and some extensions", *Information Processing Letters*, 7 (1): 10, doi:10.1016/0020-0190(78)90030-3 (https://doi.org/10.1016%2F0020-0190%2878%2990030-3)
4. Traboulsi, Ahmad (2014), *Bottleneck Spanning Trees* (http://people.scs.carleton.ca/~maheshwa/courses/5703COMP/14Seminars/BST-Report.pdf) (PDF)
5. Cui, Yuxiang (2013), *Minimum Bottleneck Spanning Tree* (http://people.scs.carleton.ca/~maheshwa/courses/5703COMP/Talks/MBST/slides.pdf) (PDF)
6. Gabow, Harold N; Tarjan, Robert E (1988). "Algorithms for two bottleneck optimization problems" (http://www.cs.princeton.edu/research/techreps/TR-104-87). *Journal of Algorithms*. 9 (3): 411 – 417. doi:10.1016/0196-6774(88)90031-4 (https://doi.org/10.1016%2F0196-6774%2888%2990031-4). ISSN 0196-6774 (https://www.worldcat.org/issn/0196-6774).

Categories: Graph algorithms | Spanning tree