Bottleneck Spanning Trees

Ahmad Traboulsi

November 17, 2014

1 Minimum Bottleneck Spanning Trees

1.1 Minimum Bottleneck Spanning Trees

Let G = (V, E) be an undirected connected graph with a cost function w mapping edges to positive real numbers. A spanning tree is a tree connecting all vertices of G. The bottleneck edge of a spanning tree is the edge with the highest cost among all edges of that tree, there might be more than one bottleneck edge in a spanning tree in which they all have the same cost. A spanning tree T is called a minimum bottleneck spanning tree (MBST) if its bottleneck edge cost is minimum among all possible spanning trees. It is easy to see that a graph may have many MBSTs (e.g. consider a graph where all edges' costs are the same, then all the spanning trees of that graph have same bottleneck edge cost and $\nexists$ spanning tree with a bottleneck edge cost lower than any other spanning tree , therefore any spanning tree of such graph is a MBST.

The well known problem Minimum Spanning Tree (MST) is related to MBST in which the Former is necessary an MBST while the opposite is not true. Therefore any algorithm that get an MST is also an algorithm to get an MBST. However we will see in the upcoming section an algorithm to get an MBST in O(E) time complexity

Most of the upcoming algorithms in this chapter try to get a MBST by achieving a unique partitioning of the Edges in the Graph G=(V,E) into two sets A* and B* such that:

- $\forall e_k \in A^*$ and $\forall e_h \in B^* \quad w(e_k) \geq w(e_h)$
- the deletion of any edge in A^* will never disconnect G. Moreover the deletion of all edges in A^* will never disconnect G.
- The Graph $G_{B^*} = (V, B^*)$ has the following properties:
  any spanning tree of G_{B^*} is an MBST of G Let K be equal to the cost of the highest edge cost in set B^*. If an edge with the cost k is deleted the graph G_{B^*} might get disconnected, but if all edges of cost k were deleted then Graph G_{B^*} will get disconnected

1.2 Camerini's algorithm for finding an MBST in a connected undirected Graph

Cameron's algorithm for finding an MBST is based on partitioning the set of edges E into two sets A and B and then checking if the current partition B of the edges in the graph G is enough to have a spanning tree of G or not, if yes then the objective now is to get the MBST of the graph G_B in which again we do the partitioning and ask the same question. If the edges in B did not form a spanning tree of G then we consider the forest formed along with the MBST of the Graph G_A collapsed into η where η here is the set of components formed by the maximal forest of G_B. We define the following functions and sets which will we see in the algorithm:

1. A= UH(E) // Function UH takes E set of edges in G and returns A $\subset$ E such that:
   (a) $|A| = \left\lfloor \frac{|E|}{2} \right\rfloor$
   (b) $\forall e_k \in A$ and $\forall e_h \in (E - A) \quad w(e_k) \geq w(e_h)$

2. Let $B = E - A$

3. F= FOREST(G_B) // where $G_B = (V, B)$ and FOREST returns F such that:
   (a) F the maximal forest of $G_B = (V, B)$

4. $\eta = \{N_1, ..., N_c\}$ where N_i is the i_th component of F

Now for the function UH we are actually computing the median and then go through the set of edges E, any edge greater than the median we put it in set A. Then if $|A| \neq \left\lfloor \frac{|E|}{2} \right\rfloor$ we keep adding the edges with the cost equal to median until $|A| = \left\lfloor \frac{|E|}{2} \right\rfloor$. Now the rest of the edges we put them in B. While the Function forest is implemented utilizing DFS.

1.2.1 Theorem 1

(a)If F is a spanning tree of G then an MBST of G is given by the MBST of G_B

(b)If F is not a spanning tree of G then an MBST of G is given by F $\cup$ any MBST of Graph G' . Where G' is Graph G_A Collapsed into η

Proof

In theorem 1 (a) it can be clearly seen that $\forall e_h \in B \leq \forall e_k \in A$ this implies that the maximum weight of edges in spanning tree of G is not less that G_B. In part (b) of the theorem, consider S an MBST of G, modify S such that if $e_h \in B - S$ creates a cycle in $S + e_k$ that contains an edge $e_k \in A$ then replace e_k by e_h, by the end of all of these substitution you will have S=F+S' where S' is the spanning tree of G'

The following algorithm suggested by Camerini:

Algorithm 1 Compute a MBST of Graph G

```
1: procedure MBST(G, w)
2:     Let E be the set of edges of G
3:     if |E| = 1 then
4:         Return E;
5:     else
6:         A ← UH(E, w);
7:         B ← E − A
8:         F ← FOREST(G_B)
9:         Let η = {N_1, N_2, ....., N_C} where N_i(i = 1, 2, ....., c) is the set of nodes
   of the i − th component of F;
10:        if c = 1 then
11:            Return MBST(G_B, w);
12:        else
13:            Return F ∪ MBST((G_A)_η, w);
14:        end if
15:    end if
16: end procedure
```

The time Complexity analysis of the above algorithm is as follow:

UH, FOREST, G_B, $(G_A)_\eta$ all require O($\frac{m}{2^i}$) at the $i-th$ iteration, where m is the number of edges at the first call. Since UH is similar to finding the median and then splitting the edges of E with respect to that median and finding the median can be done in O(m) time ref.[6], FOREST can be computed using DFS, and finally G_B, $(G_A)_\eta$ since it only consist of building the adjacency list which contains O($\frac{m}{2^i}$) edges at iteration i.

Therefore runs in $O(m + \frac{m}{2} + \frac{m}{4} + ... + 1) = O(m)$

Description of the illustration below showing an example of running the MBST algorithm

MBST was called for 4 times as follow:

- 1^{st} time was called for the original Graph $G \rightarrow F$ was not a spanning tree therefore return F $\cup$ MBST$(G_A)_\eta$

- 2^{nd} time was called for the Graph $(G_A)_\eta$ F' of G'_B is a spanning tree of $(G_A)_\eta$ therefore return MBST(G'_B)

- 3^{rd} time was called for the Graph $G_{B'}$
  F'' of $G_{B''}$ is not a spanning tree of G'_B therefore return $F'' \cup$ MBST$((G_{A''})_\alpha)$

- 4^{th} time was called for the Graph $(G_{A''})_\alpha$

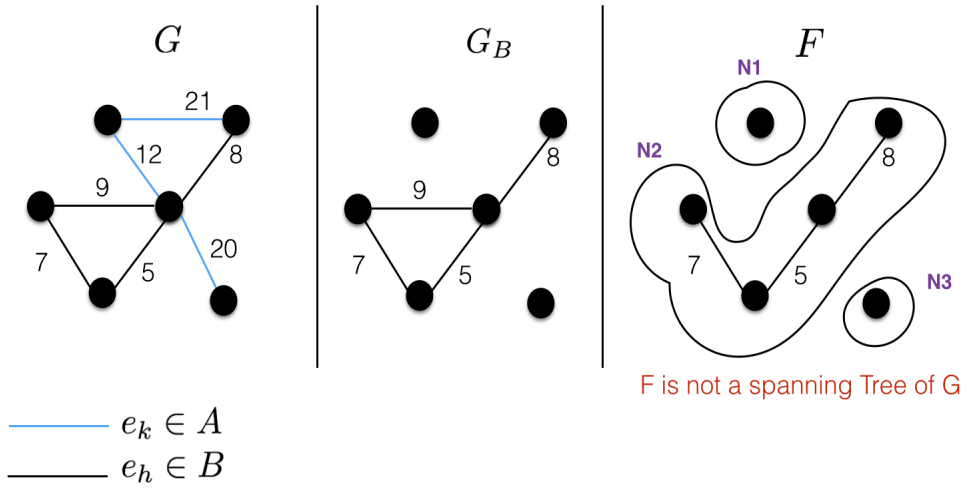

Figure 1: Disconnected Components in F

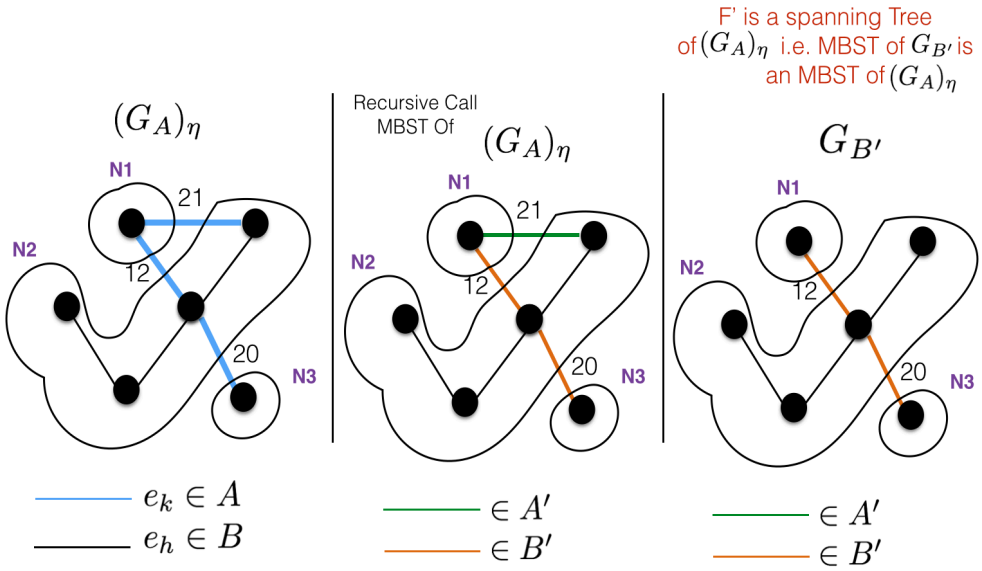

Figure 2: Recursive Call for MBST on $(G_A)_\eta$

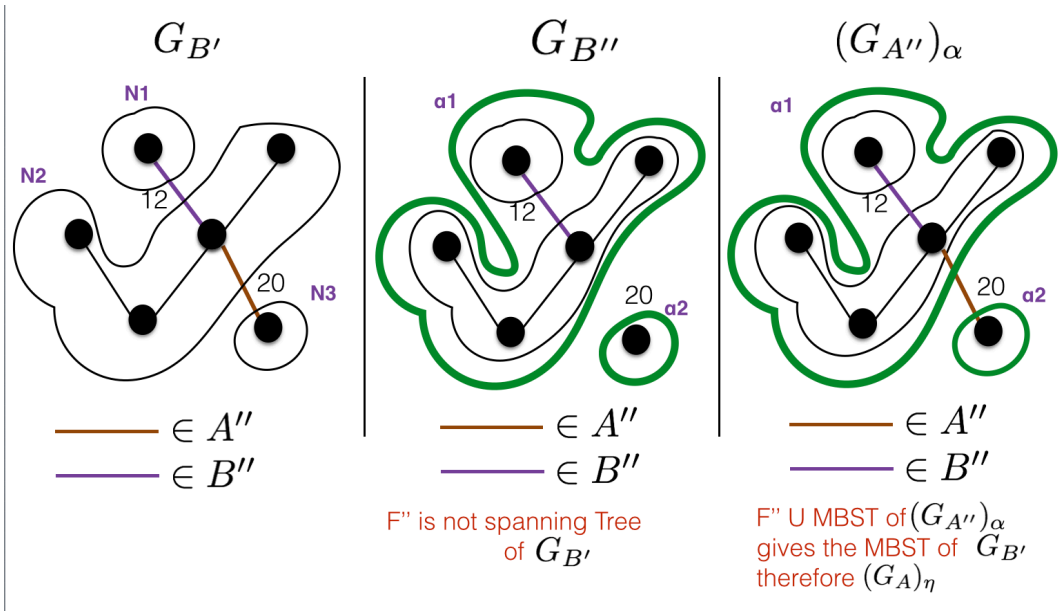

Figure 3: Recursive Call for MBST on $G_{B'}$ and then $(G_{A''})_\alpha$

MBST OF $G = F \cup$ MBST of $(G_A)_\eta$

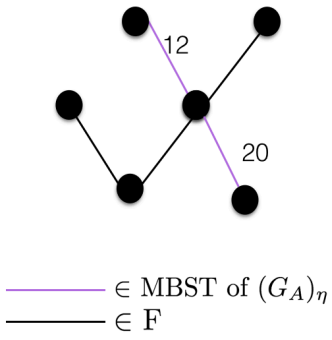

Figure 4: Finally MBST of G

1.3 MBST in Directed Graphs (MBSA)

Given a directed graph $G = (V, E)$ with a cost function w mapping edges to positive real numbers. An arborescence of G is a directed tree of G which contains a directed path from a specified, say node L, to each node of a subset V' of $V - \{L\}$. Node L is called the root of arborescence.
An arborescence is a spanning arborescence if $V' = V - \{L\}$. An MBST in this case is called a Minimum Bottleneck Spanning Arborescence (MBSA).

1.3.1 Camerini's algorithm for finding MBSA

For a directed graph Camerini's algorithm focus 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. Below we define the functions and sets required in the algorithm of camerini for MBSA. In the directed graph G=(V,E), it is assumed to have a spanning arborescence rooted at node L. We define the follow

1. A= UH(E-T) // Function UH takes (E-T) set of edges in G and returns A $\subset$ (E-T) such that:
   (a) $|A| = \left\lfloor \frac{(|E-T|)}{2} \right\rfloor$
   (b) $\forall e_k \in A$ and $\forall e_h \in ((E - T) - A) \quad w(e_k) \geq w(e_h)$

2. T a subset of E for which it is know that G_T does not contain any spanning arborescence rooted at node L. Initially T=$\emptyset$

3. Let $B = (E - T) - A$

4. F= BUSH($G_{B \cup T}$) // where $G_{B \cup T} = (V, B \cup T)$ and BUSH returns F such that:
   (a) F the maximal arborescence of $G_{B \cup T}$ rooted at L.

The function UH is same us the one utilized in the previous algorithm while BUSH is also similar to FOREST in which it is implemented using DFS.
There are two cases in which algorithm 3 is build upon. The cases are

(a) If F was a spanning arborescence of G then MBSA of G is given by MBSA of $G_(B \cup T)$

(b) If F was not a spanning arborescence of G then T is increased by B then compute A,B and F again.

Algorithm 2 Main call

```
1: procedure MBSA-MAIN(G, w, T)
2:     S ← BUSH(G);
3:     T=∅;
4:     MBSA(G,w,T);
5:     Return S;
6: end procedure
```

Algorithm 3 Compute a MBSA of A Directed Graph G

```
1: procedure MBSA(G, w, T)
2:     Let E be 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_{B∪T});
7:         if F is a spanning arborescence of G then S ← F; MBSA((G_{B∪T}, w, T);
8:         else
9:             MBSA(G, w, T ∪ B);
10:        end if
11:    end if
12: end procedure
```

Fig 5 shows how the search space of the MBSA is changing during a run of the algorithm.

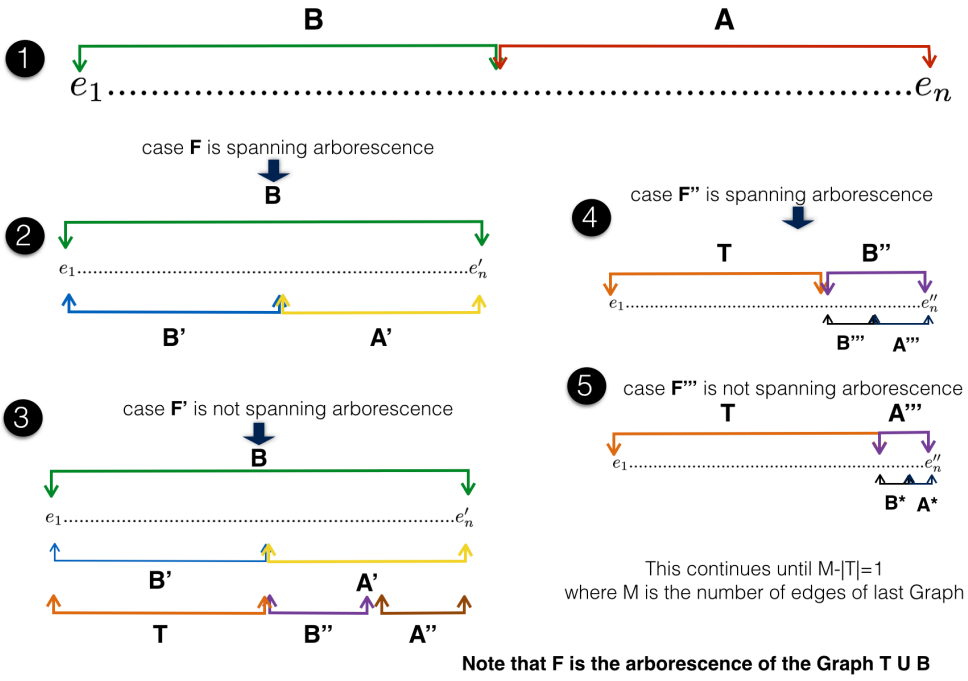

Figure 5: Search Space

The Time Complexity analysis of algorithm 3 is as follow:

- UH requires O(m)
- BUSH requires $O(m)$ at each execution , implemented using DFS.
- The number of these executions is $O(logm)$ since $|E - T|$ is being halved at each call of MBSA

Total total time complexity is $O(mlogm)$

1.3.2 Gabow and Tarjan algorithm for MBSA

Gabow and Tarjan noticed that if Dijkstras single-source shortest path algorithm ref.[2] was modified slightly it will produce an MBSA. Their modified algorithm is shown below:
In the directed graph G=(V,E) let c(u,v) denote the cost of the edge (u,v). s is a distinguished root vertex that has a path to all nodes in G. Let $p(v)$ denote the parent of v in the tree. F contains a collection of vertices along with their corresponding inclusion cost $c(v)$. Initially F contains node s the distinguished root vertex with $c(s) = -\infty$

Algorithm 4

```
1: procedure MBSA-GT(G, w, T)
2:     for |V| times do
3:         Select v with minimum c(v) from F
4:         Delete it from F
5:         for ∀edge(v, w) do
6:             if w ∉F or ∉ Tree then
7:                 add w to F
8:                 c(w)=c(v, w)
9:                 p(w)=v
10:            else
11:                if w ∈ F AND c(w) > c(v, w) then
12:                    c(w) = c(v, w)
13:                    p(w) = v
14:                end if
15:            end if
16:        end for
17:    end for
18: end procedure
```

The above algorithm runs in $O(|V|log|V|+|E|)$ time if Fibonacci heap ref.[4] was used to implement F.
Below a table showing the operations on fibonacci heap time complexity taken from `http://en.wikipedia.org/wiki/Fibonacci_heap`

Operation	Fibonacci[1]
find-min	$\Theta(1)$
delete-min	$O(\log n)$[b]
insert	$\Theta(1)$
decrease-key	$\Theta(1)$[b]
merge	$\Theta(1)$

Figure 6: Operations Time Complexity

The Total edges visited are $|E|$ and we are iterating in the outer for loop for $|V|$ times the operation done inside the inner loop are in $O(log|V|)$, therefore time complexity is $O(|V|log|V|+|E|)$

1.3.3 Gabow and Tarjan Finding λ^*

Another approach proposed by Tarjan and Gabow with bound of $O(mlog*n)$ 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 another words the iteration number, and K(i) an increasing function that denotes the number of partitioned sets that we 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(λ^*) is an MBSA in which G(λ^*) is the Graph where all its edge's costs are $\leq \lambda^*$. The algorithm to find λ^* is presented below.

Algorithm 5

procedure MBSA-KPARTITIONING(G, w)
  λ_1= $c(u,v)$ where $c(u,v) \leq c(x,y)\ \forall(x,y) \in E$ // λ_1 be the minimum edge cost in E
  λ_2= $c(u,v)$ where $c(u,v) \geq c(x,y)\ \forall(x,y) \in E$ // λ_2 be the maximum edge cost in E
  i=0;
  Step 1:
    i $\leftarrow i+1$;
    Let $S_0 \subset E$ such that $\forall e_k \in S_0\ c(e_k) \leq \lambda_1$
    Let $E_1 \subset E$ such that $\forall e_h \in E_1\ \lambda_1 < c(e_h) \leq \lambda_2$
  Step 2:
    Partition E_1 into k(i) subsets such that $S_1, S_2, S_3, ...,S_{(k(i))}$
    and $\forall e_r \in S_i$ and $\forall e_p \in S_{i+1}\ e_r \leq e_p$
    and the size of each subset is $\lceil \frac{|E_1|}{K(i)} \rceil$ or $\lfloor \frac{|E_1|}{K(i)} \rfloor$
  Step 3:
    Find minimum j such that $G(j) = (V, S_0 \cup S_1 \cup S_2 \cdots \cup S_j)$
    and in G(j) $\forall v \in V \exists$ path from s to v where s is the distinguished root
  Step 4:
    if (j=0)
      λ^*=λ_1 Terminate;
    Else
      λ_1= $c(u,v)$ where $c(u,v) \leq c(x,y)\ \forall(x,y) \in S_j$
      // λ_1 be the minimum edge cost in S_j
      λ_2= $c(u,v)$ where $c(u,v) \geq c(x,y)\ \forall(x,y) \in S_j$
      // λ_2 be the maximum edge cost in S_j
      Go to Step 1
end procedure

Complexity analysis of the above algorithm

Step 3 takes $O(|E|)$ using incremental search ref.[3]. Step 1 and step 4 take $O(|E|)$ each. And Step 2 is similar to finding the median however you have to find the median $logk(i)$ times takes $O(|E_1|logK(i))$ but since $K(i) = 2^{K(i-1)}$ and at the i-th iteration $|E_1|$=$O(\frac{|E|}{k(i-1)})$ therefore it runs in $O(|E|)$
The total time per iteration is $O(|E|)$ and the number of iterations is $O(log^*K)$ giving an overall time $O(|E|log^*K)$

The below figures illustrate the partitioning of the Gabow and Tarjan second algorithm.

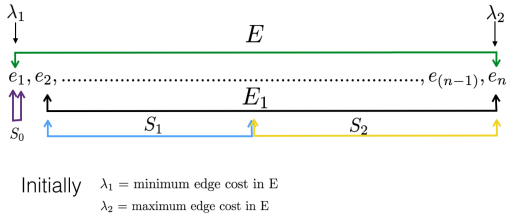

Figure 7:

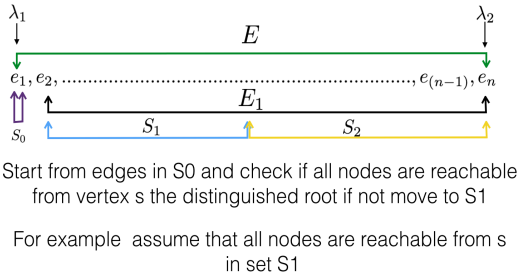

Figure 8:

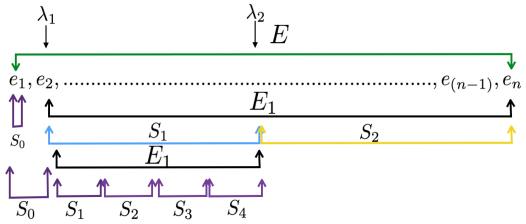

Figure 9:

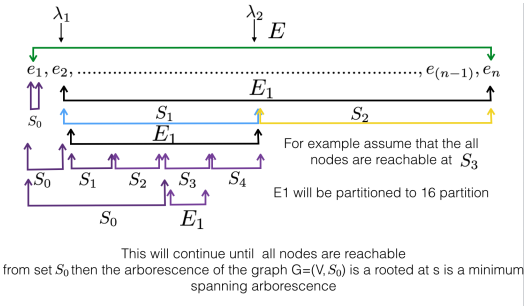

Figure 10:

Exercises

1. Give a linear time algorithm to determine if a graph $G = (V, E)$ contains a MBST with its maximum edge $\leq b$, where b is a given constant.

2. Show that a MST is always a MBST while MBST is not always an MST.

References

[1]P.M. Camerini, The Min-Max Spanning Tree Problem And Some Ex- tensions, Information Processing Letters, Vol. 7, Num. 1, January 1978

[2]E. W. Dijkstra, A note on two problems in conexxion with graphs, Numer. Math. 1 (1959), 269-271.

[3]H.N. Gabow, R.E. Tarjan, Algorithms for Two Bottleneck Optimization Problems, Journal Of Algorithms 9 (1988) 411-417

[4]M. L. Fredman, R. E. Tarjan, Fibonacci heaps and their uses in network optimization, J. Assoc. Comput. Mach. 34 (1987), 596-615.

[5]Minimax and applications by Du, Dingzhu; Pardalos, P. M Nonconvex optimization and its applications, 1995

[6]Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. (August 1973). "Time bounds for selection". Journal of Computer and System Sciences 7 (4): 448461

[7] http://flashing-thoughts.blogspot.ru/2010/06/everything-about-bottleneck-spanning.html