Seminar Advanced Algorithms and Data Structures - Student Manuscript

SENSITIVITY ANALYSIS OF MINIMUM SPANNING TREES AND SHORTEST PATH TREES [1]

Sara Steiner

October 26, 2018

1 Definitions

In general I refer to the graph we are analyzing as G, with G consisting of a set of n vertices V and m edges E. The edges of G will be referred to as $e_i, i \in \{1, ..., m\}$ and the vertices of G as $v_i, i \in \{1, ..., n\}$. In addition each edge e_i has an assigned weight $c(e_i)$.
A minimum Spanning Tree (MST) (or a shortest Path Tree (SPT)) of $G = [V, E]$ will be referred to as $T_G = [V, E']$, with $E' \subset E$. When talking about a graph G and its MST T_G we will refer to non-tree edges by f.
For some algorithms we need to build a Transmuter Graph to do the computation on. I will refer to this graph as H with vertices labeled s, t or w.

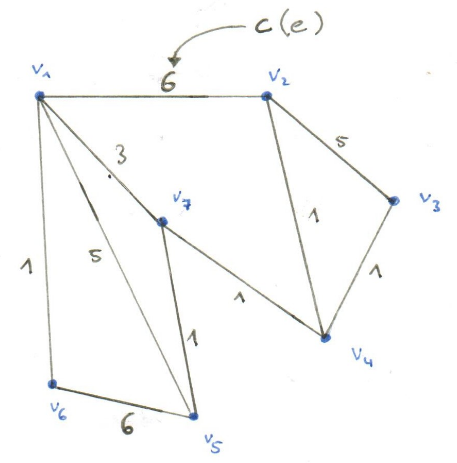

Figure 1: Graph G

2 Sensitivity Analysis of a Minimum Spanning Tree (MST)

A sensitivity analysis is helpful to find the amount an edge weight can vary before another path becomes faster (eg. IP routing) or to find the weakest point of a shortest path. This could be applied in automobile traffic or in the SBB route-planer.

2.1 Definition of MST?

A MST $T_G = [V, E']$ of a Graph $G = [V, E]$ is a connected sub-graph of G that minimizes the sum of it's edge weights. Let $w(T_G) = \sum_{e \in E'} c(e)$ be the weight of T_G.
T_G is a MST of G if T_G is a spanning tree and minimizes $w(T_G)$.

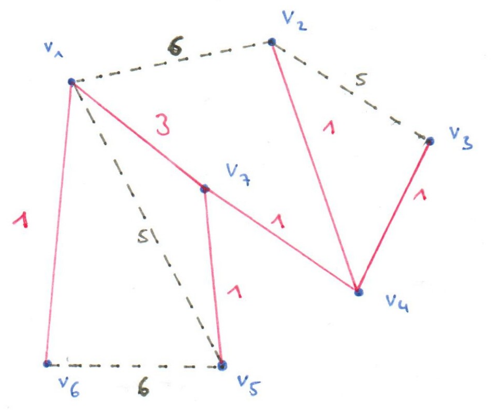

Figure 2: MST of Graph G. See figure 1

2.2 Sensitivity on a MST

The sensitivity of a MST can be understood as the amount by which each edge weight of G can vary, before the MST changes.
The algorithm we are going to look at, will show us by how much the weight of each tree edge can be increased and by how much the weight of each non-tree edge can be decreased before the MST changes.

2.3 Measure Sensitivity on a MST

In order to measure this, we will first have a look at a lemma (from [1]) and define what a simple path is.

Definition: Simple path in T_G
A simple path in T_G for a non-tree edge $f = \{x, y\}$, is referred to as $T(f)$. $T(f)$ denotes the path in T_G from vertex x to y. A simple path $T(f)$ is unique in a MST T_G.

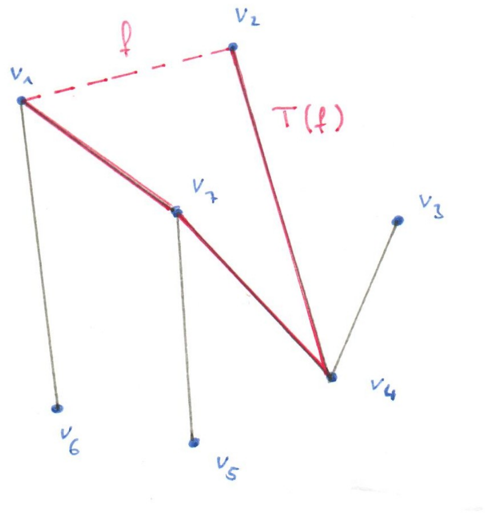

Figure 3: Simple path for f in T_G

Lemma 1 (from [1], reformulated):
$T_G = [V, E']$ is a MST of $G = [V, E]$ with $E' \subseteq E$, if and only if, T_G is a spanning tree of G and for each non-tree edge f, the weight of f is at least as big as the largest weight $c(e_i)$ on $T(f)$.

Proof of Lemma 1
Assume T_G is a spanning tree and prove its minimality.
To show:

$$T_G = [V, E'] \text{ is a MST of } G = [V, E] \Leftrightarrow$$
$$\forall f \in E \setminus E' \ . \ \forall e \in T(f) \ . \ c(f) \geq c(e)$$

First prove $\Rightarrow$ by contradiction:
Assume there exists f and e_k with $c(f) < c(e_k)$ and $e_k \in T(f)$ for a Spanning

Tree T_G.
Since $T(f) \cup f$ is a cycle we can remove one arbitrary edge from it, without disconnecting the vertices.
We build a new Spanning Tree G'_T by copying T_G, then adding f and removing e_k.
So we get:

$$w(T_G) = \sum_{e \in E'} c(e) > \sum_{e \in E' \setminus \{e_k\}} c(e) + c(f) = w(T'_G)$$

This leads to the conclusion that $w(T_G) > w(T'_G)$ and so T_G is not a MST of G.
So

$$T_G = [V, E'] \text{ is a MST of } G = [V, E] \Rightarrow$$
$$\forall f \in E \setminus E' \ . \ \forall e \in T(f) \ . \ c(f) \geq c(e)$$

Then prove $\Leftarrow$ by contradiction:
Assume T_G is not a MST of G.
Then there has to be a $T'_G = [V, E'_2]$ with

$$\sum_{e \in E'} c(e) = w(T_G) > w(T'_G) = \sum_{e \in E'_2} c(e)$$

Since T_G and T'_G are Spanning Trees they both have the same number of edges. So for every edge that is in T'_G and not in T_G there has to be an edge in T_G that is not in T'_G.
And since E_G and T'_G are both acyclic and connected it has to hold that: For every $e_k \in E'_2 \setminus E'$ there has to be an $e_p \in E' \setminus E'_2$ that is on $T(e_k)$ in T_G. Otherwise $e_k \cup T(e_k)$ would be a cycle in T'_G. From $w(T_G) > w(T'_G)$ we know that

$$\sum_{e \in E' \setminus E'_2} c(e) > \sum_{e \in E'_2 \setminus E'} c(e)$$

There has to be at least one $e_k \in E'_2 \setminus E'$ and $e_p \in E' \setminus E'_2$ with $e_p \in T(e_k)$ and $c(e_p) > c(e_k)$.

So the existence of an $e_k \in E \setminus E'$ and an $e_p \in T(e_k)$ with $c(e_p) > c(e_k)$ is proven.

$$T_G = [V, E'] \text{ is a MST of } G = [V, E] \Leftarrow$$
$$\forall f \in E \setminus E' \ . \ \forall e \in T(f) \ . \ c(f) \geq c(e)$$

Apply Lemma 1
To measure how much the weight of a non-tree edge f can be decreased, we

consider $c(f)$, then look at $T(f)$ and select edge e_i in $T(f)$ with the largest weight $c(e_i)$. If we now decrease $c(f)$ by more than $c(f) - c(e)$, f will no longer be the most expensive edge in the cycle consisting of $T(f)$ and f. Therefore T_G will no longer be a MST.

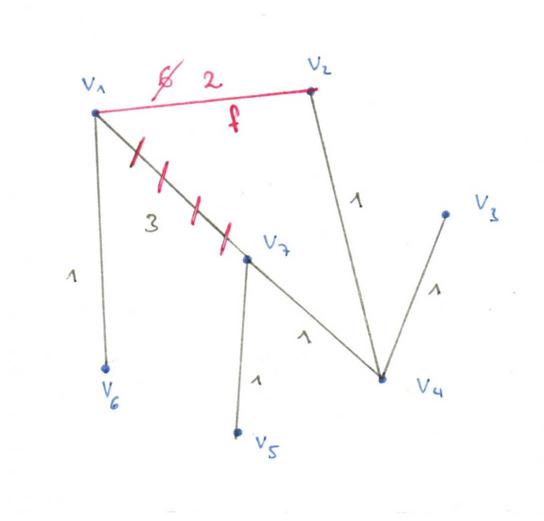

Figure 4: MST of G changes if edge f changes.

For a tree edge e, we first get $c(e)$ and then we look at all the simple paths e lies on. From those paths we only consider the one with the lowest corresponding non-tree edge weight. We refer to that path as $T(f)$ and to the corresponding non-tree edge as f. If we now increase $c(e)$ by more than $c(f) - c(e)$, e will become the most expensive edge on the cycle consisting of $T(f)$ and f.

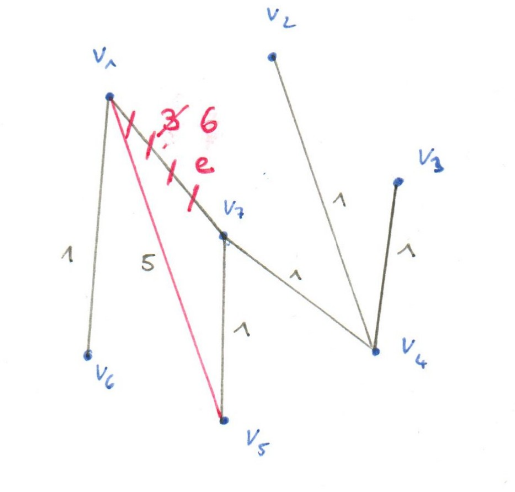

Figure 5: MST of G changes if edge e changes

2.4 Calculate Sensitivity of a MST

We build a Transmuter-Graph H that allows us to calculate the sensitivity of G for all edges at the same time. Once we got H, we will be able to get the sensitivity analysis in $O(size(H))$ time.

2.4.1 Transmuter Graph for MST

First of all, what is a Transmuter Graph?
A Transmuter Graph H for a graph G is a directed graph that contains one vertex of indegree zero $s(e)$ for each tree edge e in T_G and one vertex of outdegree zero $t(f)$ for each non-tree edge f in G. It additionally contains arbitrarily many other vertices of indegree 2 and outdegree of at least 1.
There is a path from a source $s(e)$ to a sink $t(f)$ if and only if the corresponding tree edge e is on $T(f)$.
Note: The vertices of H can be brought in to topological order. This is possible since H is acyclic.

Simple approach to building a Transmuter Graph H
Note: This is just a simple approach, used only to show that building Transmuter Graphs is possible. This algorithm does not provide the desired size of H. In order to get to a smaller H a much more complicated algorithm is needed. If you are interested in it, see [2].

Choose a root r in T_G. For every tree-edge e_i in T_G add one vertex $s(e_i)$

to H.
For each non-tree edge $f = (v_i, v_j)$ in G, find the nearest common ancestor (nca) v_a of v_i and v_j.
The nearest common ancestor of $f = (v_i, v_j)$ is the vertex at which the paths from r to v_i and from r to v_j start to differ.
From v_a follow T_G to v_i. For the first two edges traversed in T_G, e_k and e_l, add one vertex w_a to H. Add the edges $(s(e_k), w_a)$ and $(s(e_l), w_a)$ to H.
For every other edge e_o you traverse in T_G, let w_b be the last added vertex in H. Add a vertex w_{b+1} to H and the edges $(s(e_o), w_{b+1})$ and (w_b, w_{b+1}).
Once you reach v_i (assume the last added vertex of this procedure was w_p), repeat the same process for v_j (assume the last added vertex of this procedure was w_q).
Finally add a vertex $t(f)$ to H and add the edges $(w_p, t(f))$ and $(w_q, t(f))$.

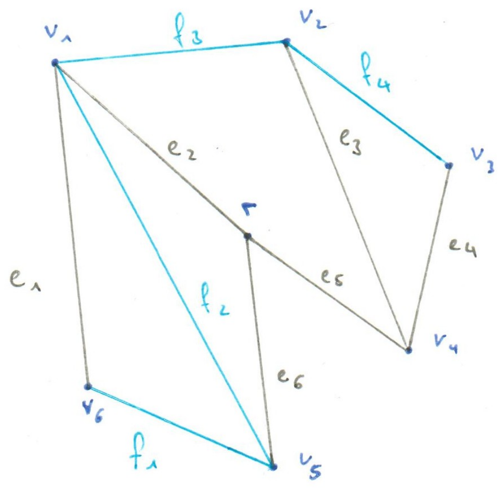

Figure 6: Graph G

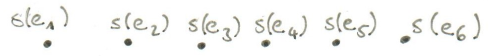

Figure 7: H after construction of $s(e_i)$ for all tree edges in T_G.

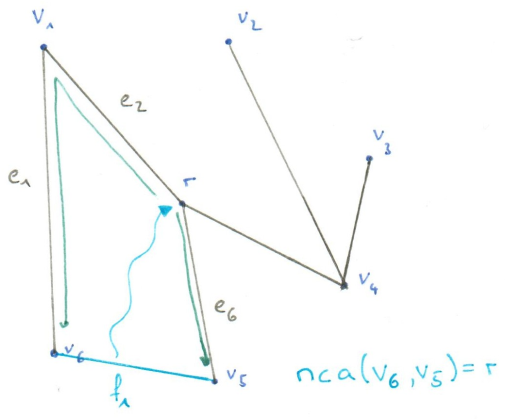

Figure 8: Graph G, finding $nca(f)$.

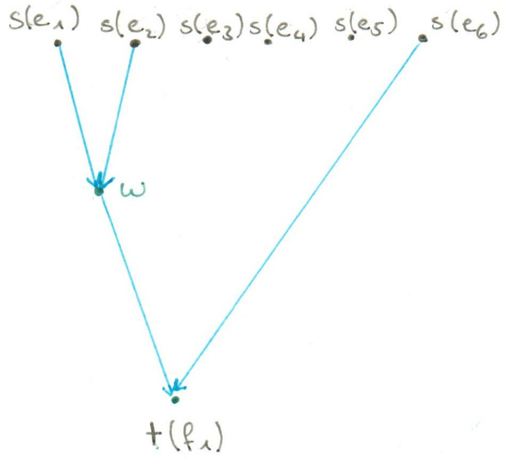

Figure 9: H after processing f in G.

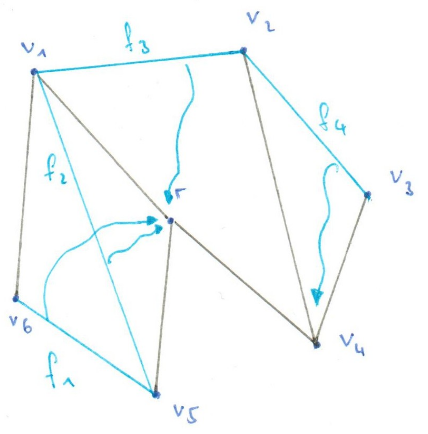

Figure 10: Graph G, with all *nca* marked.

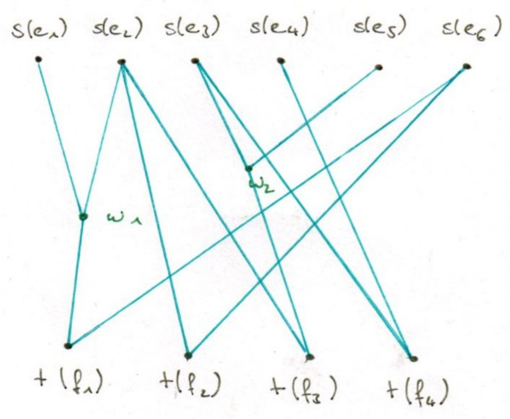

Figure 11: H fully built.

2.4.2 Algorithm for Sensitivity on MST

Overview:
We go through the Transmuter Graph twice. Once to determine the maximum decrease for each non-tree edge and once to find the maximum increase for each tree edge before T_G is no longer a MST of G.

Decrease of weight for non-tree edges in words:
We first label each source in H with the weight of the corresponding edge in T_G.
Then we label all the other vertices with the maximum value of their immediate predecessors.
The resulting label for each sink $t(f)$ is the maximum edge weight on $T(f)$. So for each sink we get the sensitivity by calculating $c(f) - t(f)$. This is how much we can decrease the non-tree edge f before T_G is no longer a MST of G.

Decrease the weight for non-tree edges in pseudo-code:

input: topologically ordered graph H with sources $s(e_i)$, sinks $t(f_i)$ and other vertices w_i, with index i corresponding to topological order

```
label all sources:
for all $s(e_i)$
$label(s(e_i)) = weight(e_i);$


label all $w_i$:
for all $w_i$ in order
$label(w_i) = \max_{(x,w_i)\in E} label(x);$


label all sinks:
for all $t(e_i)$
$label(t(e_i)) = \max_{(x,t(e_i))\in E} label(x);$


calculate the maximum decrease for non-tree edges:
for all $t(e_i)$
$maxdecrease(e_i) = c(e_i) - label(t(e_i));$
```

Increase of weight for tree edges in words:
We first label each sink $t(f)$ in H (Transmuter Graph) with the weight of the corresponding edge in G.
Then we label all the other vertices with the minimum value of their immediate successors.
The resulting label for each source $s(e)$ is the weight of the cheapest non-tree edge f that has $e \in T(f)$. So for each source we get the sensitivity by calculating $s(e) - c(e)$. This is how much we can increase e before T_G is no longer a MST of G.

Increase the weight for tree edges in pseudo-code:

input: topologically ordered graph H with sources $s(e_i)$, sinks $t(f_i)$ and other vertices w_i, with index i corresponding to topological order

```
label all sinks:
for all $t(e_i)$
$label(t(e_i)) = weight(e_i)$;


label all $w_i$:
for all $w_i$ in reverse order
$label(w_i) = \min_{(w_i,x) \in E} label(x)$;


label all sources:
for all $s(e_i)$
$label(s(e_i)) = \min_{(s(e_i),x) \in E} label(x)$;


calculate the maximum increase for tree edges:
for all $s(e_i)$
$maxincrease(e_i) = label(s(e_i)) - c(e_i)$;
```

2.4.3 Run time

The run time of this algorithm is linear in the size of H. This is easy to see since each vertex needs to be looked at twice. Also every edge is looked at at most twice. Since the number of edges is bound to two times the number of nodes in H, the total run-time is in $O(size(H))$.
So what really interests us is how fast we are able to build a Transmuter Graph H and how many edges and vertices it will have.
It is possible to build H in $O(m\alpha(m,n))$ time and space and with some tricks we can do the sensitivity analysis in $O(m)$ space. We will look at that on the next page. The algorithm we looked at was a simplified version. I will not show you the original algorithm for the Transmuter Graph, but if you are interested, I can recommend you a paper. [2]
First we will have a look at α which represents the inverse Ackermann's function.

Ackermanns function (as defined in [1])
For integers $i, j \geq 1$ $A(i, j)$ is defined by:

$A(1, j) = 2^j$ for$j \geq 1$
$A(i, 1) = A(i-1, 2)$ for $i \geq 2$
$A(i, j) = A(i-1, A(i, j-1))$ for $i, j \geq 2$

For integers m, n such that $m \geq n - 1 \geq 0$ we define $\alpha(m, n)$ by
$\alpha(m, n) = min\{i \geq 1 | A(i, \lfloor (m+1)/n \rfloor) > log_2 n\}$

Now a few words the reduction of the space required by the algorithm.
Let m' be the number of non-tree edges. So $m' = m - n + 1$. We split the non-tree edges into groups of size $\Theta(m'/\alpha(m, n))$. We apply the algorithm to each group, meaning that we take T and the non-tree edges of one group and do the sensitivity analysis on it. Each non-tree edge will receive its correct result and the tree edges will get $\Theta(\alpha(m, n))$ different results, of which the smallest is correct.
This leads to a space used by the algorithm per group of

$$O\biggl(n + \frac{m'}{\alpha(m,n)} \alpha\bigl(\max\bigl\{n, O(\frac{m'}{\alpha(m,n)})\bigr\}, n\bigr)\biggr) = O(m)$$

There is a max in this formula, which can have 2 outcomes. If

$$\max\bigl\{n, O(\frac{m'}{\alpha(m,n)})\bigr\} = n$$

this leads to

$$O\biggl(n + \frac{m'}{\alpha(m,n)} \alpha\bigl(n, n\bigr)\biggr) = O(n + m') = O(n + m - n + 1) = O(m)$$

.
If

$$\max\bigl\{n, O(\frac{m'}{\alpha(m,n)})\bigr\} = O(\frac{m'}{\alpha(m,n)})$$

$$O\biggl(n + \frac{m'}{\alpha(m,n)} \alpha\bigl(O(\frac{m'}{\alpha(m,n)}), n\bigr)\biggr) = O\biggl(n + \frac{m'}{\alpha(m,n)} \alpha\bigl(O(m), n\bigr)\biggr) = O(m)$$

3 Sensitivity Analysis of a Shortest Path Tree (SPT)

We will refer to the SPT with T_G. In addition we will define a root r in G from which the SPT is constructed. For all paths P_j from r to v in G, the distance $d(v) = min_j\{\sum_{e_i \in P_j} c(e_i)\}$.

3.1 Definition of SPT?

For a directed graph G and a vertex $r \in G$ a shortest path tree is a sub-graph T_G of G such that

- T_G is a connected acyclic graph containing all vertices of G
- and for all v_i : $d(v_i)$ in $G = d(v_i)$ in T_G.

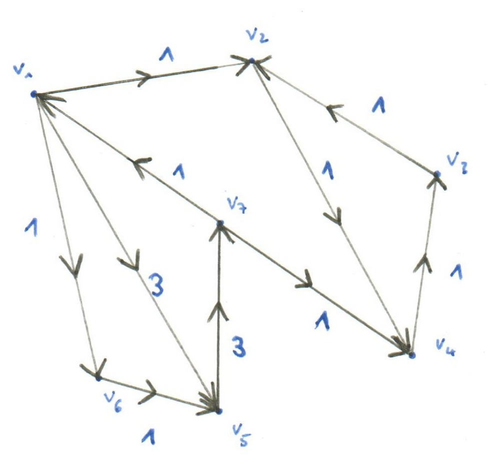

Figure 12: Directed graph G.

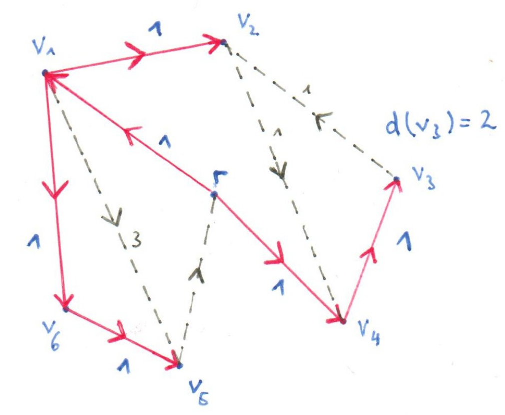

Figure 13: SPT of G with respect to r.

3.2 Sensitivity of a SPT

Since a SPT T_G of G is always defined with respect to a root r. The sensitivity of a SPT can be understood as the amount by which the weight $c(e)$ of each tree edge e can be increased and the weight $c(f)$ of each non-tree edge f can be decreased before T_G is no longer a SPT of G.

3.3 Measure Sensitivity of a SPT

We will first look at Lemma 2 of [1]. (Reformulated)

Lemma 2:
The sub-graph T_G of G is a SPT if and only if the weight $c(f)$ of a non-tree edge $f = (v_i, v_j)$ is larger equal to $d(v_j) - d(v_i)$.

This leads to the conclusion that if the weight of a non-tree edge $c(f)$ is decreased by more than $c(f) - d(v_j) + d(v_i)$, T_G is not a SPT of G anymore.

If the weight $c(e)$ of a tree edge $e = (x, y)$ is increased by more than $c(f) + d(v_i) - d(v_j)$ for any $f = (v_i, v_j)$ such that v_i is not a descendant of y but v_j is a descendant of y, T_G will not be a SPT anymore.

When increasing the weight $c(e)$ of a tree edge $e = (x, y)$ in a SPT T_G by Δc the distance of y and all the nodes in the subtree S_y of y in T_G are going to increase by Δc.

Every non-tree edge $f = (v_i, v_j)$ belongs to one of four cases:

Case 1:
None of f's vertices belong to S_y. In this case it is still true that

$$c(f) \geq d(v_j) - d(v_i)$$

So edge f will still not be part of the SPT after changing $c(e)$.

Case 2:
Both vertices of f belong to S_y, so $d(v_i)$ and $d(v_j)$ both increase by Δc. Since $c(f) \geq d(v_j) - d(v_i)$ also

$$c(f) \geq \big(d(v_j) + \Delta c\big) - \big(d(v_i) + \Delta c\big)$$

So edge f will still not be part of the SPT after changing $c(e)$.

Case 3:
If $v_i \in S_y$ and $v_j \notin S_y$ the distance $d(v_i)$ changes by Δc but the distance $d(v_j)$ stays the same. Since $c(f) \geq d(v_j) - d(v_i)$ also

$$c(f) \geq d(v_j) - \big(d(v_i) + \Delta c\big)$$

So edge f will still not be part of the SPT after changing $c(e)$.

Case 4:
If $v_i \notin S_y$ and $v_j \in S_y$ the distance of $d(v_i)$ stays the same, while the distance $d(v_j)$ changes by Δc. Since $c(f) \geq d(v_j) - d(v_i)$ we can find out that

$$c(f) \geq \big(d(v_j) + \Delta c\big) - d(v_i)$$

if and only if:

$$\Delta c \leq c(f) + d(v_i) - d(v_j)$$

So edge f will only not be part of the SPT after changing $c(e)$ if Δc is small enough.

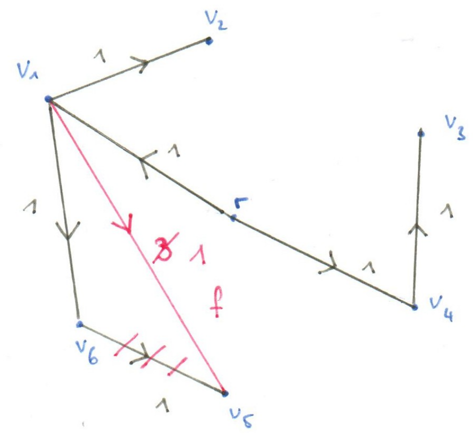

Figure 14: Resulting T_G by changing $c(f)$ in G.

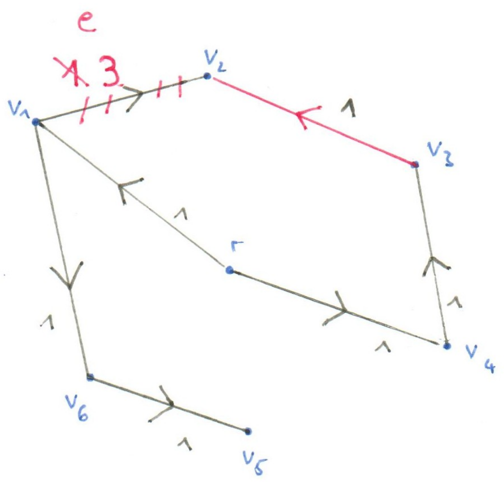

Figure 15: Resulting T_G by changing $c(e)$ in G.

3.4 How to Calculate Sensitivity on a SPT

3.4.1 Transmuter Graph for SPT

This time we have to manipulate G a little before we build a Transmuter Graph. More specifically we build a graph G' consisting of T_G and some other edges.

For every non-tree edge $f = (x, y)$ we get the nearest common ancestor of x and y (referred to as $nca(f)$) and then construct an edge $f' = (nca(f), y)$ in G'. We then build the Transmuter Graph H on G' with T_G as the spanning tree. Note: These extra edges $f' = (x, y)$ have a path P from x to y in G'. If the weight of any edge $e \in P$ is changed the edge f in G that corresponds to f' in G' is in Case 3 of the case distinction made beneath Lemma 2.

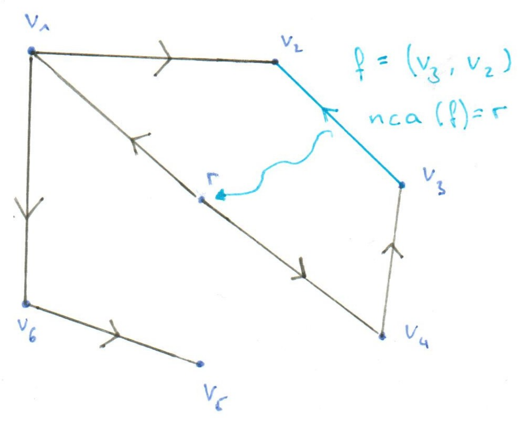

Figure 16: Finding $nca(f)$ in T_G.

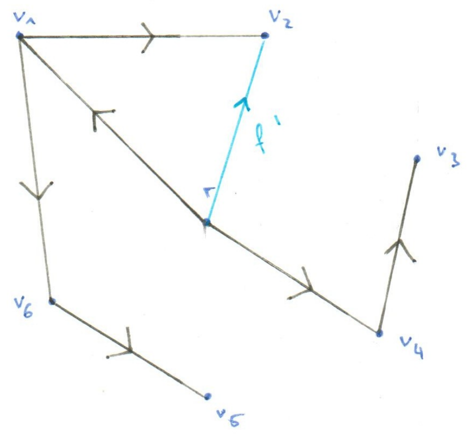

Figure 17: Adding extra edge f' to G'.

3.4.2 Algorithm to calculate Sensitivity on SPT

Decrease weight for non-tree edges in words:
In this part we do not need the Transmuter Graph.
As a first step we compute $d(v)$ for every vertex in T. Then for every non-tree edge $f = (v_i, v_j)$ we get its maximum decrease by calculating $\Delta(f) = c(f) + d(v_i) - d(v_j)$.

Increase weight of tree edges in words:
Here we need H.
The process is analogous to the backwards labeling of the MST. The only difference is that we label the sinks $t(f')$ with $\Delta(f)$ instead of $c(f)$.
Also the last step of the algorithm is not required since the sources $s(e_i)$ are directly labeled with their maximum increase value.

Increase weight for tree edges in pseudo-code:

```
input: topologically ordered graph with sources $s(e_i)$, sinks $t(f')$ and other vertices $w_i$

label all sinks:
for all $t(f')$
label $(t(f')) = delta(f)$;


label all $w_i$:
for all $w_i$ in order
$label(w_i) = min_{(w_i,x)\in E} label(x)$;


label all sources :
for all $s(e_i)$
$label(s(e_i)) = min_{(s(e_i),x)\in E} label(x)$;
```

4 References

[1] Robert Endre Tarjan, Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees(1981)

[2] Robert Endre Tarjan, Applications of path compression on balanced trees, J.ACM 26(1979) 711-715