Internship report – StackMST

Olivier Marty
Tutors: Jose Correa, department of industrial engineering,
and Jose Soto, department of mathematical engineering,
university of Chile.

1 June – 15 August 2015

INGENIERIA INDUSTRIAL
UNIVERSIDAD DE CHILE

Le contexte général

Le problème StackMST (pour Stackelberg Minimum Spanning Tree) est un jeux à deux joueurs et un tour, avec deux niveaux d'optimisation. Il se déroule sur un graphe avec deux types d'arêtes : le premier joueur fixe le poids des arêtes du premier type tandis que les autres ont des poids fixés. Le deuxième joueur choisit alors un arbre couvrant de poids minimum, et le but du premier joueur est de gagner le maximum avec ses arêtes dans l'arbre couvrant de poids minimum.

Ce problème est NP-difficile et APX-difficile dans le cas général, et P avec certaines classes de graphe. Autrement, un algorithme d'approximation avec un facteur non constant est connu.

Le problème étudié

Pour certaines classes de graphe, la question de la complexité du problème est encore ouverte et, dans le cas général, la question de l'existence d'un algorithme d'approximation avec un facteur constant l'est également. Quelques papiers ont été publiés sur ce problème, mais de nombreuses questions subsistent.

La contribution proposée

Nous avons montré que certaines restrictions du problème étaient toujours NP-difficiles, et nous avons cherché un algorithme d'approximation avec un facteur constant à l'aide d'un programme linéaire, mais sans avoir abouti pour le moment. Notre programme linéaire semble meilleur qu'une autre proposition publiée, mais nous ne connaissons pas encore son *integrality gap*, et il n'est pas évident de savoir comment utiliser une solution fractionnaire pour aboutir à un algorithme d'approximation.

Les arguments en faveur de sa validité

Les instances qui mettent en évidence l'*integrality gap* du programme linéaire publié ne sont pas des solutions réalisables de notre programme linéaire, qui reste tout de même une formulation du problème.

Le bilan et les perspectives

Notre approche était liée au problème étudié, mais les techniques classiques en théorie des algorithmes d'approximation et en programmation linéaire que j'ai apprises peuvent servir à chercher et à étudier des algorithmes d'approximation pour de nombreux problèmes.

Nos résultats de complexité sont intéressants mais laissent ouverte une nouvelle question de complexité (entre deux graphes avec deux prix et deux chemins avec trois prix reste la question de la complexité du problème avec deux chemins – ou même deux arbres – et seulement deux prix).

Enfin le travail sur notre programme linéaire n'est pas abouti : par exemple la question, cruciale, de son *integrality gap* n'est pas résolue.

During this Master 1 internship of ten weeks at the *Universidad de Chile*, in Santiago, I studied with Jose Correa, Professor, and Jose Soto, Assistant Professor, the problem StackMST. It is a problem easy to state but there are a lot of questions that stay open.

We have mainly looked for a better approximation algorithm for this NP-hard problem, and this research has leaded us to some complexity results on restrictions of this problem.

1 Problem statement

Imagine a network where there is two types of links: some of them are yours, and the others belong to your competitor. Somebody wants to link all the points of the network, that is to rent links in order to construct a spanning tree. Each link has a price, so the person will try to minimize his expenses: he will rent a *minimum spanning tree* (MST). Knowing costs that your competitor practice, what are the optimum price you can choose to maximise your revenue ? That is the sum of the price of each link in the MST that belong to you. If your pricing policy is too aggressive, very few of your links will be selected: your revenue will be low. Conversely, it your policy is too few aggressive, your will not earn much for each link...

Formally, the network is a graph $G = (V, R \sqcup B)$ where the edges are partitioned into two sets: the red edges in R and the blue ones in B. Edges in R are labelled with a cost function $c : R \to \mathbb{R}$. They are the links of your opponent. Also R is supposed to contain a spanning tree, otherwise there is a cut consisting only of blue edges and the problem is unbounded.

Then the two players come into play: the leader chooses a price function $p : B \to \mathbb{R}$ and the follower takes a minimum spanning tree T of G, with regards to both c and p. The leader tries to maximise his revenue, i.e. to maximise

$$\sum_{e \in T \cap B} p(e)$$

This is a Stackelberg game: a leader and a follower, with two stages of optimisation.

We assume that during the construction of the spanning tree, the follower will prefer, at the same cost, blue edges to red edges. Otherwise we can reduce all the price by small amount and approach as close as wanted the best possible revenue.

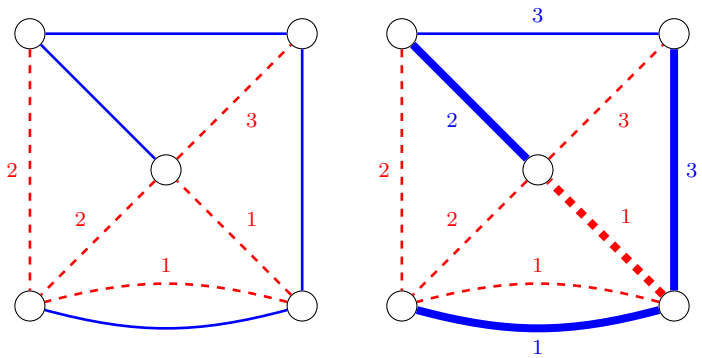

Figure 1: An example of instance, and a price function with one of its corresponding MST in bold, whose revenue is 6.

1.1 Notations

In all this report we will use the following notations:

- $c_1 < \cdots < c_k$ are the different values taken by c. So k is the number of different values.
- For a set of edge A, and a price p, we denote by $A_{<p}$ the subset of A with edges cheaper than p, and so one with $\leq$, $>$, $\geq$, and $=$.

  For $1 \leq j \leq k$ we note $A_{\leq j}$ for $A_{\leq c_j}$ with an abuse of notation.
- For a set of vertices S, we denote by $A(S)$ the subset of A with edges that have both endpoints in S, i.e. $A(S) = A \cap (S \times S)$.

1.2 Greedy algorithms for MST

It is handful to have in mind both Kruskal's and Prim's algorithms. These greedy algorithms find an MST of a connected weighted graph.

Here is how Kruskal's operates: it sorts edges by increasing weight, then, starting with an empty tree, it considers each edge in this order and adds them to the tree as long as it does not induce a cycle. When all the edges have been considered, the tree spans the graph, and one can prove that it is an MST.

For its part, Prim's algorithm starts with a tree that contains only one node, then it finds the cheapest edge that have exactly one endpoint in this tree, and adds it to the tree. It repeats this operation until no such edge exist.

Here, in agreement with our assumption, we consider that both algorithms choose blue edges before red edges of the same price.

1.3 Some remarks

There is always an optimal price function that takes the same values than the cost function c. Indeed with any price function, we can increase other prices while maintaining the same MST until they reach values of c, and decrease prices for edges that are more expensive than the largest cost: they are not selected in any MST and if reduce their costs may add them to an MST, it will increase the revenue (see Kruskal's algorithm to be convinced).

Consequently, we will consider only price functions that takes these values.

Moreover, on the assumption that blue edges are preferred, all MST lead to the same revenue [CDF+11].

1.4 Additional assumptions

Remark 1. *We can assume that R is a tree.*

Indeed we can assume that R is reduced to one of its MST. Otherwise, because R is connected, there is a red cycle and as its heaviest edge will be never selected, and then will not interact with blue edges neither, we can omit it. By repeating this process we find an MST of R. If there are several such edges in a cycle one can choose which

one delete, see remark 3 for an argument with connected components of edges cheaper than a given price.

Remark 2. *We can assume that B contains a spanning tree.*

Otherwise, there is a cut of only red edges, and contract one of the cheapest edge of this cut will not change the revenue of any solution. Indeed it will always be selected in any MST in order to connect the two sides of the cut. Because all MST lead to the same revenue, if there are several such edges one can choose arbitrarily which one contract.

To finish a new insight:

Remark 3. *We can assume that R is a path.*

With Kruskal's algorithm the decision of adding or skipping an edge of weight p depends only on the connected components formed by edges of weight smaller than p. Hence, we can construct R' a path such that, for all weight p, the graphs $(V, R_{\leq p})$ and $(V, R'_{\leq p})$ have the same connected components. Thus for every blue price function, Kruskal's algorithm can select the same blue edges. In particular both problems have the same optimal revenue and solution.

1.5 State of the art

1.5.1 Complexity and approximation

The problem is known to be NP-hard and APX-hard even when c only takes values 1 and 2 [CDF+11]. In section 3 an idea of the reduction is depicted.

A non-constant approximation is known, along with a linear program that is a formulation of StackMST and whose the integrality gap is exactly the approximation factor. Both are explained further.

It is also known to be NP-hard even on planar graph and when c only takes values 1 and 2 [CDF+09].

However some cases are in P: series-parallel graphs and bounded-treewidth graphs via dynamic programming [CDF+09]. The case when R is a tree and B is the complete graph but R is polynomial if c take only two prices, otherwise it is approximable between $7/4 + \varepsilon$, though the question of NP-hardness is still open [BGLP10]. They also give a $2h + \varepsilon$ approximation when the radius of R is bounded by h.

A generalisation of the problem with activation costs and a budget is also studied in [BGLP10].

1.5.2 A way to solve StackMST

Let $F \subset B$ an acyclic set of blue edges. The optimal price function such that all the edges in F are selected in an MST is computable. For an edge $e \in F$, let $\mathcal{C}(F, e)$ the set of cycle in the graph $(V, F \cup R)$ that contains e. Then we set

$$p(e) = \min_{C \in \mathcal{C}(F,e)} \max_{f \in C \cap R} c(f)$$

that is for each such cycle, keep the most expensive red edge, and price e like the cheapest of them.

The proof that all F is selected and the revenue is optimal is in [CDF+11].

Since it is computable in polynomial time, a way to solve StackMST is to guess which acyclic set of blue edges has to be selected. In fact some algorithms cited in the previous section keep trace of which edges select.

2 Best-out-of-k algorithm

This is the best known approximation algorithm for StackMST in the general case. It appears in [CDF+11] with a proof of its approximation factor.

This algorithm is very simple: it tries for $1 \leq i \leq k$ to put all blue edges at price c_i, computes the revenues and keeps the best solution.

Its approximation factor is

$$\min\left\{k, 1 + \ln\left(\frac{c_1}{c_k}\right), 1 + \ln(|B|)\right\}$$

where k is the number of different costs, c_1/c_k the ratio between the largest and the smallest red costs, and $|B|$ the number of blue edges. There exists tight instances for each of the factor.

In appendix A, we include a proof of these factors that is slightly shorter than in [CDF+11], and includes a graphical comprehension of logarithmic factors.

2.1 Best-out-of-k with optimisation

We know that one can optimise a price function in order to keep the same set of selected blue edges.

Then a legitimate idea is to execute *Best-out-of-k* and optimise the price functions for each cost. This algorithm is optimal on all tight instances of *Best-out-of-k* exhibited in [CDF+11]: the optimum use all the blue edges, which are all selected with the cheapest cost. However we do not know if it is better than best-out-of-k without optimisation.

Here is an example in which both algorithms have the same factor. This example uses two path and two prices, a situation that we do not know to be NP-hard.

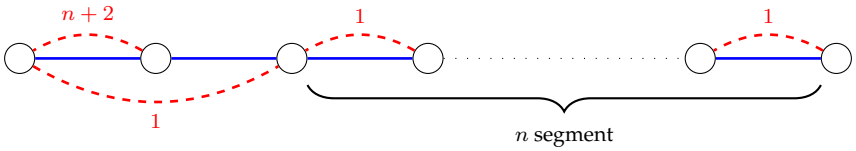

Figure 2: The factor is 2 either with or without optimisation.

In figure 2, both revenues found by best-out-of-k are $n + 2$, and in both cases the optimisation does not improve it. On the other hand the optimum is $2n + 2$ (the two leftmost blue edge have price $n + 2$, the others 1). Thus the limit of the ratio between the returned value and the optimum is $1/2$ as n approaches $+\infty$.

Though we don't know if this algorithm is better than best-out-of-k.

3 Some more NP-hardness results

3.1 NP-hardness with a fixed set of to-be-selected red edges

Let us recall the idea of the reduction in [CDF+11], which will be used to prove a stronger result.

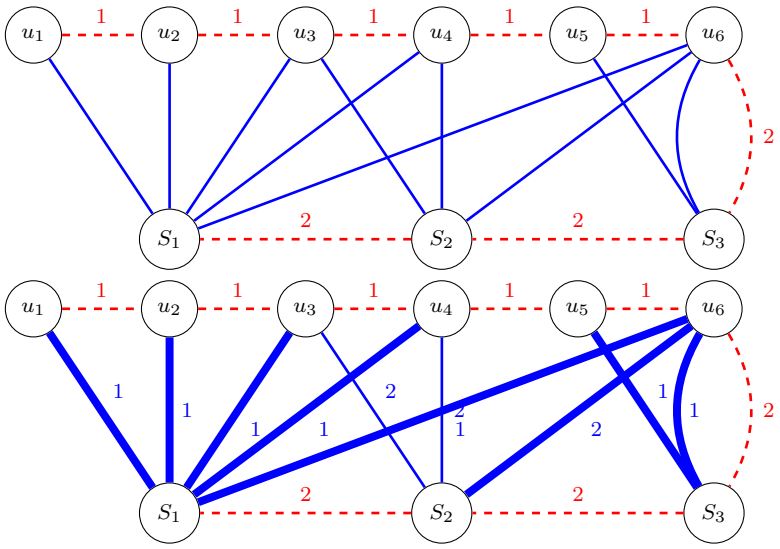

Figure 3: The constructed instance for the set cover $S_1 = \{u_1, u_2, u_3, u_4, u_6\}$, $S_2 = \{u_3, u_4, u_6\}$, and $S_3 = \{u_5, u_6\}$, and the constructed solution for the cover S_1, S_3: their edges are priced 1, the others 2. Bold edges denote an MST.

The reduction is from *SET COVER*, and here is the idea: let an instance of *SET COVER* with sets $S_1, \ldots, S_m$ and elements $u_1, \ldots, u_n$. We can suppose w.l.o.g. that u_n is in all sets. Then we construct an instance of StackMST with $n + m$ vertices, one for each element and set. A red path connects $u_1 - \cdots - u_n - S_m - \cdots - S_1$ with edges of cost 1 between vertices u_i and u_{i+1}, then edges of cost 2. A blue edge connects S_j and u_i if and only if $u_i \in S_j$. See figure 3.

Then they show that there exists a cover of size at most t if and only if there exists a price function that leads to a revenue of at least $n + 2m - t - 1$.

Let us describe our stronger result.

We know that solve StackMST is equivalent to select the best set of blue edges that must be in the MST. That is, for a given set of blue edges, we can compute in polynomial time the optimal price function that leads to an MST containing this set.

In a similar way we thought a moment to use a linear program relaxation to elect exactly which red edges must be in an MST. However, solve StackMST with a fixed such set is still NP-hard, since it is even when this set is empty.

Indeed one can remark that, in the previous reduction, if there is a cover of size t, there is a solution of revenue $n+2m-t-1$ which do not use any red edges. Therefore, with the same reduction than above, the question "is there a cover of size at most t" and "is there a price function that leads to a revenue of at least $n + 2m - t - 1$, and an MST does not contains any red edges" are equivalent. The previous construction shows the forward direction, and the converse holds because it holds without the additional restriction.

Theorem 1. StackMST *is* NP*-hard even when one elect exactly which red edges must go in an MST.*

3.2 NP-hardness when R and B are paths

Firstly a useful lemma:

Lemma 1. *Let p a price function on blue edges.*

Increase the price of a blue edge e will lead to a loss of revenue of at most its old price. More precisely, if F is the set of blue edges in an MST with regards to the old price function, one can construct an MST with regards to the new price function that contains $F \setminus \{e\}$.

Proof. Let $e = (a, b) \in B$ a blue edge. Let p' the price function that matches p on $B \setminus \{e\}$ and such that $p'(e) > p(e)$.

Let T an MST with regards to p. We claim that there exists $f \in R \cup B$ such that $(T \setminus \{e\}) \cup \{f\}$ is an MST with regards to p'. f is the edge that connects the two components containing the endpoints a and b of e.

Consider an execution $\mathcal{E}$ of Kruskal's algorithm that leads to T. We construct a similar execution $\mathcal{E}'$ for p' that leads to the announced tree. In a first time, $\mathcal{E}'$ will select the same edges T' than $\mathcal{E}$ except e. Then, as soon as it is one of the cheapest edges, $\mathcal{E}'$ will select an edge f that connects the two trees that contain a and b, the endpoints of e. Given that it is done as soon as possible, this shows that the first part of $\mathcal{E}'$ is correct: at each step the set of candidate edges is the same as in $\mathcal{E}$, plus the edges that connect the components of a and b, but these last edges were not the cheapest in the first phase. At this point the forest maintained by $\mathcal{E}'$ is $T' \cup \{f\}$ and we know that there is a step in which the forest maintained by $\mathcal{E}$ is $T' \cup \{e\}$, and these two forests contain exactly the same connected components of vertices. Thus $\mathcal{E}'$ can complete its tree with same edges than in $\mathcal{E}$, and the generated MST is $(T \setminus \{e\}) \cup \{f\}$ as claimed.

If $r(T)$ is the revenue given by T, the revenue become $r(T) - p(e) + p(f)$ if $f \in B$, or $r(T) - p(e)$ otherwise. Thus the loss is at most $p(e)$. □

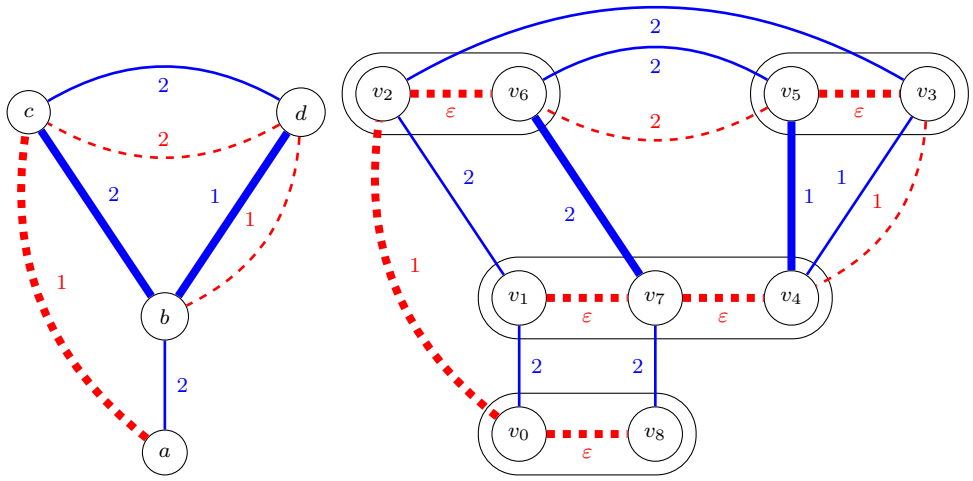

Figure 4: Vertices are copied and linked with red edges of cost ε such that B becomes a path.

Let G, c an instance of StackMST where $G = (V, R \sqcup B)$ is the graph and c the cost function.

We construct a new instance $G' = (V', R' \sqcup B')$, c' in which R' and B' are paths. See figure 4 for an example of construction.

Let $0 \leq \varepsilon < \frac{1}{2|B|}$ a real. We assume that $\varepsilon < c_1$, where c_1 is the cheapest cost of the instance (we can assume that $c_1 > 0$ since StackMST only with instances whose costs are 1 or 2 is still NP-hard).

Let $v_0 \in V$ a vertex, and $(v_0, v_1, \ldots, v_m = v_0)$ the list of vertices (with copies at each pass) met during a depth-first traversing of (V, B), starting at v_0. Let f the function that maps each element of this list to the original vertex of V to which it corresponds.

We set $V' = \{v_0, v_1, \ldots, v_m\}$, and $B' = \{(v_i, v_{i+1}), 0 \leq i < m\}$. B' is obviously a path, where each edge has been doubled.

Now we construct R'. For each $v \in V$, $f^{-1}(\{v\})$ is the set of copies of v in V'. We link these vertices together with a red path whose edges have cost ε.

Then for each $e = (a, b) \in R$, we select two copies of a and b in V', that are v_i and v_j such that $f(v_i) = a$ and $f(v_j) = b$. It is possible because B contains a spanning tree, thus the depth-first traversing visits all vertices. We link v_i and v_j by a red edge of cost $c(e)$.

Now if we quotient G' by f, that is to say identifying vertices with same value and eliminate loops and parallel blue edges, G' become isomorphic to G: all the copies of each vertex collapse, thus no edges of costs ε stay, the duplication of blue edges are deleted and then blue and red edges connect vertices as in G.

Let $K \in \mathbb{N}$ a positive integer. Then questions "is there a price function on B such that the revenue is at least K" and "is there a price function on B' such that the revenue is at least K" are equivalent.

Let $p : B \to \mathbb{R}$ a price function on B that leads to a revenue greater than K. Let $p' : B' \to \mathbb{R}$ such that for $e = (a, b) \in B$, we keep the same price on both copies of e, i.e. if $(v_i, v_j) \in B'$ such that $f(v_i) = a$, $f(v_j) = b$, then $p'((v_i, v_j)) = p(e)$. The revenue of p' is greater than K: in an execution of Kruskal's algorithm, all the red edges of cost ε are selected. Then both graphs are isomorphic if we contract these edges, and p and p' match.

Conversely, let p' a cost function on B' that leads to a revenue greater than K. We first replace value ε by an other price: let p'' such that

$$\forall e \in B', p''(e) = \begin{cases} c_1 & \text{if } p'(e) = \varepsilon \\ p'(e) & \text{otherwise} \end{cases}$$

According to lemma 1, the loss of revenue is at most $\varepsilon |p'^{-1}(\{\varepsilon\})| \leq \varepsilon |B'| \leq \varepsilon 2|B| < 1$. Then, since p' leads to a revenue greater than an integer K, the revenue of p'' is an integer greater than $K - \varepsilon|B'|$, that is also an integer greater than K. Let p defined by, for all $e = (a, b) \in B$,

$$p(e) = \min_{(u,v) \in B', f(u)=a, f(v)=b} \{p''((u, v))\}$$

the minimum price for all copies of e.

Now p and p'' lead to the same revenue. Indeed in an execution of Kruskal's algorithm for G' and p'', all the red edges of cost ε are selected, because no blue edges of this price exist. Then only the blue edge of minimum price between two ε-connected components count because if there is two such edges, they create a cycle with red edges of cost ε. Thus the situation is exactly the same as in G, and both revenue are equals.

To conclude, we know with remark 3 that G' can be transformed into an equivalent instance in which R' become a path, without transforming B'. Note that if R was a

path, we can easily construct R' in order to keep a collection of paths at each step, and a path at the end.

Theorem 2. StackMST *is* NP*-hard even when c takes three different values and R and B are paths.*

With only two different costs, the question is still open when B is a tree or a path.

4 Towards a better *IP*

Let us recall the IP (integer program) of [CDF+11].
For any family of set of vertices $C_1, \dots, C_t$ pairwise disjoint, $\delta(C_1 : \cdots : C_t)$ is the multi-cut of these sets, that is the set of edges that have one endpoint in a set, and the other endpoint in another. When $C_1 = S$ and $C_2 = V \setminus S$, we note $\delta(S) = \delta(C_1 : C_2)$.
For $e \in B$, and $1 \leq j \leq k$, the variable $x_{j,e}$ is 1 if the edge e has cost *at least* j.

There is the IP:

$$\max \sum_{\substack{e \in B \\ 1 \leq j \leq k}} (c_j - c_{j-1}) x_{j,e}$$

s.t.

$$\sum_{e \in B \cap \delta(C_1 : C_2 : \cdots : C_t)} x_{j,e} \leq t - 1 \qquad \forall j \in \{1, 2, \dots, k\}, \quad \forall C_1, \dots, C_t \text{ components of } (V, R_{\leq j-1}); \tag{1}$$

$$\sum_{e \in P \cap B} x_{1,e} + x_{j,f} \leq |P \cap B| \qquad \forall f = ab \in B, \forall j \in \{2, 3, \dots, k\}, \quad \forall Pab\text{-path in } (B \cup R_{\leq j-1}) \setminus \{f\}; \tag{2}$$

$$x_{1,e} \geq x_{2,e} \geq \cdots \geq x_{k,e} \geq 0 \qquad \forall e \in B; \tag{3}$$

$$x_{j,e} \in \{0, 1\} \qquad \forall j \in \{1, 2, \dots, k\}, \forall e \in B. \tag{4}$$

Constraints (1) hold because, in Kruskal's algorithm, all the C_i become connected components before select any edge of cost c_j. Then select more than $t - 1$ edge in the multi-cut will lead to a cycle.

Constraints (2) hold because in a such cycle, not all the blue and the red edges can be selected. Then if f is priced at least c_j, it is more expensive than all the red edges in the cycle, and at least one blue edge has to be dropped.

It is a formulation of StackMST, and the integrality gap of its LP-relaxation is the approximation factor of the algorithm *Best-out-of-k*: $\min\{k, 1 + \ln(|B|), \ln(c_1/c_k)\}$. There are tight instances for each case.

Now there is our new proposal:

We consider k copies of each blue edge: one for each possible price. In this way, we work with the weighted graph $(V, R \sqcup B_1 \sqcup \cdots \sqcup B_k)$ where each B_j is a copy of B in which edges have weight c_j.
We denote by E all the edges, i.e. $E = R \cup B_1 \cup \cdots \cup B_k$. For any edge e in E, we denote by p_e its weight.
For a set S of vertices, we denote by $\bar{\delta}(S)$, the set $\{e \in \delta(S) | p_e \leq \min_{f \in \delta(S) \cap R} p_f\}$. This is the set of edges cheaper that the cheapest red edge of the cut.
The variables are x_e, for all $e \in E$. Their interpretation is that the set of edges whose

variable equal 1 are an MST for some price function. For any set of edge A, we use the shorthand $x(A)$ for $\sum_{e\in A} x_e$.
And for $e \in B$, we denote when necessary e_j the copy of e in B_j, whose price is c_j.

Then here is the IP:

$$\max \sum_{e\in B_1\cup\cdots\cup B_k} x_e p_e$$

s.t.

$$x(E) = |V| - 1; \tag{5}$$
$$x(\bar{\delta}(S)) \geq 1 \qquad \forall S \subsetneq V, S \neq \emptyset; \tag{6}$$
$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1 \qquad \forall j \in \{1,\dots,k\}, \forall S \subset V, S \neq \emptyset; \tag{7}$$
$$x_e \in \{0,1\} \qquad \forall e \in E. \tag{8}$$

4.1 Formulation

Firstly, let us give an intuition on constraints.

The constraint (5) ensures that the variables denote a set of edges that have as many edges as in a spanning tree.

Constraints (6) ensure that we select one of the cheaper candidate edges for any cut: we have to select at least one edge cheaper than the cheapest red edge of the cut. This is the idea behind Prim's algorithm.

Constraints (7) is an improved version of the cycle-elimination constraint $x(E(S)) \leq |S| - 1$ (with $j = 1$). During an execution of Kruskal's algorithm, when it considers edges of price c_j, the endpoints of $R_{\leq j-1}(S)$ are already connected together, it implies that not a lot of edges in $x(E_{\geq j}(S))$ can be selected.

Also note that for all edge $e = (a, b) \in B$, this constraint with $S = \{a, b\}$ and $j = 1$ implies $\sum_{1\leq j\leq k} x_{e_j} \leq 1$ (uniqueness of variable valued 1 for each original edge).

Proposition 1. *This IP is a formulation of* StackMST.

Proof. Let p a price function, and T an MST of G weighted by both c and p.
For all $e \in R$, we set $x_e = 1$ iff $e \in T$, and for all $e \in B_1 \cup \cdots B_k$, let e' its original copy in B, and we set

$$x_e = \begin{cases} 1 & \text{if } e' \in T \text{ and } p(e') = p_e \\ 0 & \text{otherwise} \end{cases}$$

that is x_e is 1 iff e appears in T with the price p_e.

Then constraints (5) and (8) holds.

Constraints (6): with the proof of Prim's algorithm in mind, we know that for all cut $\delta(S)$, any MST must contains one of the cheapest edge of the cut. Well, $\bar{\delta}(S)$ contains all the cheapest edges of the cut.

Constraints (7): let $S \subset V$, $S \neq \emptyset$, and $1 \leq j \leq k$. Let $e \in R_{\leq j-1}(S)$. In an execution of Kruskal's algorithm, its endpoints has to be connected before consider any edge of cost c_j. Thus when the algorithm will consider edges of costs at least c_j, it has to connect at most $|S| - |R_{\leq j-1}(S)|$ connected components, and this is done with at most $|S| - |R_{\leq j-1}(S)| - 1$ edges. This shows the constraint.

Conversely, let x a feasible solution of the IP.
For e in E, if there exists j such that $x_{e_j} = 1$ we set $p(e) = c_j$, otherwise we set $p(e) =$

$+\infty$.
Let $T = \{e \in E | x_e = 1\}$.
We show that T (considered with original copies of blue edges) is an MST for G with price function p. With (5) and (7) with $j = 1$, we know that T is a tree.

Let $S \subset V$, $S \neq V$ and $S \neq \emptyset$. We show that one of the cheapest edge of G in the cut $\delta(S)$ is in T.
With (6) we have $x(\bar{\delta}(S)) \geq 1$, then $\bar{\delta}(S) \cap T$ contains at least one edge, including one of the cheapest edge of the whole cut. Indeed if there is a cheapest edge that is blue, is it in T, otherwise its price was $+\infty$, and in $\bar{\delta}(S)$, otherwise a red edge of the cut is cheaper. If there are no cheapest blue edges, it is because they are more expensive than the cheapest red edge of the cut, then they do not appear in $\bar{\delta}(S)$, or are not selected and are priced $+\infty$. Thus the only edges of $\bar{\delta}(S)$ that can be valued 1 in the IP are the red ones, that are by construction of $\bar{\delta}$ the cheapest of the cut.

Given that this holds for any S, this shows that T is a minimum spanning tree as in Prim's algorithm, and the objective value is actually the revenue.

Constraints 7 with $j > 1$ are there for improving the integrality gap of the LP-relaxation of this IP (at least they break some bad instances of the first LP-relaxation). □

4.2 Separation

The aim of this IP is to solve its LP-relaxation (by removing constraints (8)) and to use a fractional solution in order to find a good solution of the instance.

If the exponential number of constraints may be an issue to solve the LP-relaxation in polynomial time with regards to the size of the instance, it is known with the ellipsoid method that if one is able to solve the separation problem in polynomial time, then it is possible to solve the optimisation problem as well.

Let us recall that the separation problem for an LP is the following: given a vector x, say if x is a feasible solution of the LP, and if not, give a violated constraint.

Proposition 2. *This LP-relaxation is separable in polynomial time.*

Proof. Constraints (7): we use the fact that one can find the maximum of a supermodular function in polynomial time [Cun85]. Nevertheless, we ask here for $S \neq 0$ whereas this result ask for no constraints, such as minimal cardinality. Therefore we use the following set of constraints that is equivalent to constraints (7):

$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq \max\{|S| - 1, 0\} \qquad \forall j \in \{1, \dots, k\}, \forall S \subset V \qquad (9)$$

Now, for a fixed j, the function

$$S \mapsto |R_{\leq j-1}(S)| + x(E_{\geq j}(S)) - \max\{|S| - 1, 0\}$$

is supermodular as it is a sum of simple supermodular functions. Then it is possible to find its maximum and check that it is negative. Otherwise it provides a violated constraint in (9), and it can not be for $S = \emptyset$ so it provides also a violated constraint in (7).

Constraints (6): for each red edge $e = (a, b)$, find the min-cut in $(V, E_{\leq p_e})$ that separates a and b. Then check that it is greater than 1.

To finish, there is a polynomial number of constraints (5) and (8), thus it is sufficient to check them one by one. □

4.3 Stronger than the other LP-relaxation ?

The bad instances for the first LP-relaxation do not hold with our LP-relaxation. However we do not now the integrality gap of ours, which can be also bad.

We tried to show that our LP is at least as good as the first LP.

A solution x of our LP can easily be adapted into a solution $\tilde{x}$ of the first LP, by setting

$$\forall e \in B, \forall 1 \leq j \leq k, \tilde{x}_{j,e} = \sum_{l=j}^{k} x_{e_l}$$

Then constraints (3) holds, and constraints (8) imply constraints (4) in the integral case.

Constraints (7) are stronger than constraints (1): let $j \in \{1, 2, \ldots, k\}$ and $C_1, \ldots, C_t$ components of $(V, R_{\leq j-1})$. We set $S = C_1 \cup \cdots \cup C_t$, then (7) gives

$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) \leq |S| - 1$$

Moreover, since there are at most t connected components in $(S, R_{\leq j-1})$,

$$|R_{\leq j-1}(S)| \geq |S| - t$$

then by subtracting both

$$x(E_{\geq j}(S)) \leq t - 1$$

and the set $E_{\geq j}(S)$ contains all edges considered in (1), along with much more.

This constraint is thus an asset to improve the integrality gap of the LP-relaxation. However we do not know if constraints (2) are implied by our LP, but an LP with constraints (7) and (2) is, for sure, stronger.

To finish, we considered the set of constraints

$$|R_{\leq j-1}(S)| + x(E_{\geq j}(S)) + x(F_{\leq j-1}) \leq |S| - 1 \quad \forall j \in \{1, \ldots, k\}, \forall S \subset V, S \neq \emptyset, \quad \forall F \subset E(S) \text{ s.t. } R_{\leq j-1}(S) \cup F \text{ is acyclic.} \tag{10}$$

where two copies of the same blue edges are not considered to induce a cycle. These constraints imply both constraints (1) and (2) of the first LP, but do not seem to be separable.

4.4 No algorithm stands out

It is unclear how to use a fractional solution to guess a good solution.

Plus the selection of blue edges have to be done carefully: if forget an edge lead, at worst, only to its loss, select one more edge can wipe out the revenue.

Conclusion

Our new results of NP-hardness were welcome and have given us some insight in this problem. Nevertheless, they leave a thin subproblem between graphs with only two different prices and paths with three prices: paths with two prices, for which the NP-hardness stays an open question.

On the other hand, our work on the LP is not finished. Since this LP seems to be stronger that the last LP, it may be a progression towards a better approximation algorithm for StackMST than best-out-of-k. However, several questions stay open, such that its integrality gap.

To conclude I would thank Jose Correa for having received me in this country *muy distinto*, and Jose Soto who have worked with us.

References

[BGLP10] Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Specializations and generalizations of the stackelberg minimum spanning tree game. In Amin Saberi, editor, *Internet and Network Economics*, volume 6484 of *Lecture Notes in Computer Science*, pages 75–86. Springer Berlin Heidelberg, 2010.

[CDF+09] Jean Cardinal, ErikD. Demaine, Samuel Fiorini, Gwenaël Joret, Ilan Newman, and Oren Weimann. The stackelberg minimum spanning tree game on planar and bounded-treewidth graphs. In Stefano Leonardi, editor, *Internet and Network Economics*, volume 5929 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2009.

[CDF+11] Jean Cardinal, ErikD. Demaine, Samuel Fiorini, Gwenaël Joret, Stefan Langerman, Ilan Newman, and Oren Weimann. The stackelberg minimum spanning tree game. *Algorithmica*, 59(2):129–144, 2011.

[Cun85] WilliamH. Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.

Appendices

A Proof of the factor of best-out-of-k

We denote by $c_1 < \cdots < c_k$ the different values taken by c, and by $n_i = |\{e \in R \mid c(e) = c_i\}|$ the number of red edges labelled with c_i. We also denote by $N_i = \sum_{j=i}^{k} n_j$ the number of red edges of cost at least c_i.

Because it is the cost of the red spanning tree, we have

$$OPT \leq \sum_{i=1}^{k} c_i n_i$$

Lemma 2. *Let $i \in \{1, \ldots, k\}$.*
Suppose that for all e in B, $p(e) = c_i$. Then N_i blue edges are selected in any MST.

Proof. Let us analyse the construction of an MST by Kruskal's algorithm. Since R is a tree it does not contain any cycles, and then all the edges of cost $c_1, \ldots, c_{i-1}$ are selected. If K is their number, at this step there is a lack of $|R| - K = N_i$ edges that will be blue because B contains a spanning tree. □

k-approximation According to lemma 2, we have

$$SOL = \max_{i \in \{1,\ldots,k\}} \{c_i N_i\}$$

Then

$$\begin{array}{rcl} SOL & \geq & \frac{\sum_{i=1}^{k} c_i N_i}{k} \\ & \geq & \frac{\sum_{i=1}^{k} c_i n_i}{k} \\ & \geq & \frac{OPT}{k} \end{array}$$

shows the first approximation factor.

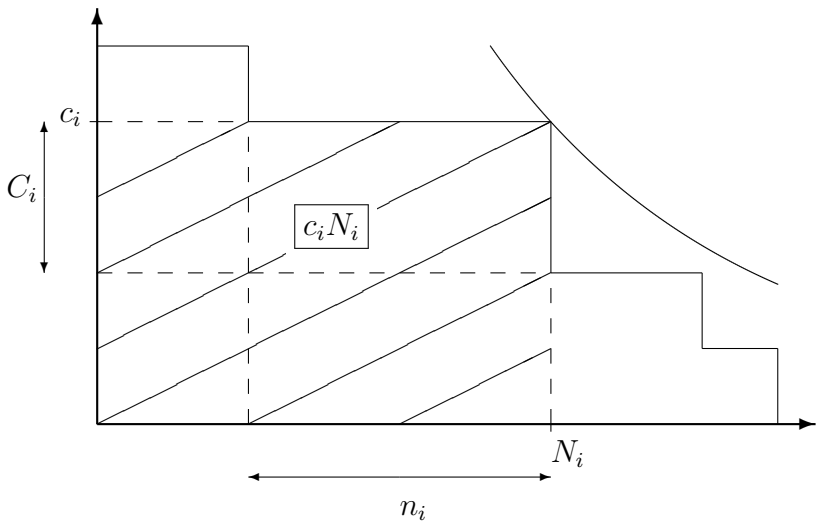

Figure 5: Example of costs distribution

$(1+\ln(|B|))$-approximation In figure 5 we show an example of red costs distribution. We put each edge on the x-axis by decreasing costs, represented on the y-axis. Thus a

vertical rectangle have as height a costs c_i and as width the number of red edges that have this cost.

Then take a rectangle like the hatched one on the figure. Its area $c_i N_i$ is the revenue by putting all prices to c_i, so the algorithm returns the area of the biggest such rectangle.

Note that the total area under the curve is $\sum_{i=1}^{k} c_i n_i$, our upper-bound for OPT.

Now we wonder what if the biggest possible total area knowing that SOL is the revenue returned by the algorithm, that is the one that maximise $c_i N_i$. All the other rectangles have smaller area, then their top-right corner are under the curve of equation $y = SOL/x$. The total area is thus smaller that the area under this curve, which leads to the two logarithmic bounds.

Now comes a formal proof.

Given that for all i in $\{1, \ldots, k\}$, $SOL \geq c_i N_i$ we have $c_i \leq \frac{SOL}{N_i}$.
Then

$$\begin{array}{rcl} OPT & \leq & \sum_{i=1}^{k} c_i n_i \\ & \leq & SOL \sum_{i=1}^{k} \frac{n_i}{N_i} \end{array}$$

and

$$\begin{array}{rcl} \sum_{i=1}^{k} \frac{n_i}{N_i} & \leq & 1 + \ln(N_1/N_k) \\ & \leq & 1 + \ln(|B|) \end{array}$$

$(1 + \ln(c_k/c_1))$**-approximation** As previously, for all i in $\{1, \ldots, k\}$ we have $N_i \leq \frac{SOL}{c_i}$. Let $C_i = c_i - c_{i-1}$, agreeing that $c_0 = 0$. Since $\sum_{i=1}^{k} c_i n_i = \sum_{i=1}^{k} C_i N_i$ we have

$$\begin{array}{rcl} OPT & \leq & \sum_{i=1}^{k} C_i N_i \\ & \leq & SOL \sum_{i=1}^{k} \frac{C_i}{c_i} \\ & \leq & (1 + \ln(c_k/c_1)) SOL \end{array}$$

This completes the proof. □