DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Comparison of Two Loss-Contracting Algorithms for Network Procurement

Philipp Michael Sauter

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Comparison of Two Loss-Contracting Algorithms for Network Procurement

Vergleich zweier Loss-Contracting Algorithmen für Network Procurement

Author:	Philipp Michael Sauter
Supervisor:	Prof. Martin Bichler
Advisor:	Richard Littmann
Submission Date:	15.08.2020

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.08.2020 Philipp Michael Sauter

Abstract

In economics there is an interest to find incentive compatible and efficiently computable auction mechanisms that give good approximations for optimal social welfare. This thesis will try to find such an approximation by using algorithms for the Steiner Tree Problem on Procurement Auctions.

The environment for the Steiner Tree Problem is a weighted, undirected graph G = (V, E), where V is the set of nodes and E is the set of edges within G. There is also a subset $T \subseteq V$ called "terminals" and a cost function d: $E \rightarrow \mathbb{R}$, that returns the added path cost for a given set of edges. The Steiner Problem now asks for the minimal tree connecting all terminals. This problem is NP-complete and the algorithms presented in this paper give approximations for the solution. The first approximation algorithm we're going to look at was presented in a paper by Berman and Ramaiyer [1] and constructs Steiner minimal trees for subsets of T with at most k elements, adding them to the solution greedily and finally removing the redundant edges. It was proven that increasing the value of parameter k, improves the approximation rate, which converges to 1.746 for $k \rightarrow \infty$. The second algorithm from Hougardy and Proemel [5] improves the previous approximation rates by using a generalized version of the parameterized relative greedy heuristic (RGH) by Karpinski and Zelikovsky [8] and iteratively applying it with different parameters α_i ($i \in [1, k]$) to the previous iteration's output. They achieved the approximation ratio of 1.598 after 11 iterations and indicated the limit at 1.588 for $k \rightarrow \infty$. In a 2007 paper, Blumrosen and Nisan [2] proved that a mechanism for single-minded bidders is incentive compatible iff it satisfies monotonicity and critical payment. We will therefore assess monotonicity for both implementations and afterwards compute critical payments. To find the critical payment for edge *e* we will change the cost of *e* in the input graph and check whether or not it is still included in the resulting Steiner Tree after applying the algorithm on this changed input graph. Using binary search we are going to find the maximum prize at which *e* is still part of the tree, which is therefore the critical payment. These critical payments should vary for both algorithms as well as the total edge cost and the runtime. To conclude this thesis we will analyse these statistics and compare them between the two algorithms.

After implementing both algorithms and checking them for incentive compatibility it turned out, that Hougardy and Proemel's algorithm does not fulfill monotonicity and is therefore not strategy proof. Berman and Ramaiyer's algorithm is incentive compatible and was chosen as the best solution for our proposed setting.

Contents

1 Introduction

There are a lot of problems that modern companies face which at least partly include network procurement problems. These problems are about finding the most efficient way to form a network. The input for these types of problems usually involves a number of points that need to be connected to each other within the desired network and a way to obtain connections between two points for a cost. This cost could be valued using money, time or opportunity cost, etc. There may be other criteria that the network has to fulfill, but we are going to focus on the foundations. The goal in any case is to find the minimum cost network fulfilling the desired criteria and connecting the required points. Some real-world examples could include networks, that span populated areas to provide e.g. water access, transport, telecommunication, electricity, etc. Other examples could include optimization problems like logistics or project planning. Even assignment problems can be modelled in this way by using a flow graph and assigning a opportunity cost to every decision. We are going to propose a very general setting for these network procurement problems, which is based on the Steiner tree problem. Since this problem is NP-complete we are going to look at approximation algorithms for our setting. We are going to implement two loss-contracting algorithms originally designed for the Steiner tree problem. We are going to use examples of Steiner tree problems on SteinLib [3] as testcases and we are going to use the reported optimal solution cost there to contrast the results of the two implemented algorithms. We are also going to check the strategy proofness of our algorithms in the proposed settings, which ensures that the valuation of the connections offered to us are authentic.

2 Problem

2.1 Setting

In the following we are going to model a network auction setting, which we then use to define our goals and the steps required in order to reach these goals. The setting for our problem is a procurement auction in which the auctioneer is able to purchase connections from different sellers in order to connect a set of nodes. The true valuations of the connections are known only to their respective sellers and they may choose to prize them any way they want. We know about these sellers that they are single minded and therefore only care about increasing their personal payoff. In order to connect the required nodes the auctioneer may choose to include other nodes, since they may help connect the required nodes and therefore reduce the overall cost of the network he/she tries to procure. Within these setting we play the role of the auctioneer and our goal is to reliably procure the minimum cost network connecting our required nodes. In order to do this we need a decision algorithm that picks out the optimal connections to buy. Since the true valuations of every connection are known only to the sellers, we need our decision algorithm to be strategy proof, which makes reporting the true valuations a dominant strategy for the sellers. The problem of finding a minimum cost network connecting a set of required nodes is in essence the Steiner tree problem, which is NP-complete and has multiple greedy approximation algorithms proposed for it. We will therefore look at different approximation algorithms for the Steiner tree problem and check whether they are strategy proof via a Lemma proposed by Blumrosen and Nisan (Lemma 1.9 [2]). In order to meet the requirements for this Lemma we need to prove monotonicity for the respective approximation algorithm as well as finding critical payments for every edge included in the proposed solution of the algorithm.

2.2 Steiner Tree Problem in Graphs

The Steiner problem, which was named after Jakob Steiner, has application in a lot of settings, but the most common setting is the Steiner tree problem in graphs. It takes an connected, undirected, weighted graph with non-negative edge-weights, as well as a set of vertices called "terminals" or "terminal points" and askes for the minimum weights tree connecting all terminals, called a Steiner minimum tree (SMT). Vertices that are not terminals are called Steiner points and they can be included in a Steiner tree, but their inclusion is optional as opposed to terminal points. The Steiner tree problem in graphs is one of Karp's 21 NP-complete problems [7] and we will therefore only look at approximation algorithms for it. The input graph for the Steiner tree problem is assumed to be complete and the cost of each edge between two vertices u and v is assumed to be the the length of the shortest path between u and v. This assumption is without loss of generality since if a connected graph doesn't fulfill these criteria, we can just use the metric closure of it.

2.3 Definitions

The environment for the Steiner tree problem in graphs is a weighted undirected graph G = (V, E), where V is the set of vertices in the graph and E the set of edges. The set T $\subseteq$ V marks the terminal vertices of G and V$\setminus$T is the set of Steiner points. The length of a given set of edges E or a given tree X d(E)/d(X) is the sum of all edge costs. For any set of vertices S any tree connecting S is called a Steiner tree for S. The minimum spanning tree of S is denoted by MST(S) and its length by mst(S). Similarly a Steiner minimal tree for S is denoted by SMT(S) and its length by smt(S). A Steiner tree is called full if all terminals are leaves in the tree and a k-restricted Steiner tree is a full Steiner tree that has at most k terminals. The loss of a Steiner tree X l(X) is the length of a minimum forest spanning all vertices of X, where every component of the forest contains at least one terminal node. The contraction of a set of vertices S means to set the length of edges between vertices in S to zero. The optimal solution is denoted by OPT(S) and its cost by opt(S). Finally the performance ratio of an approximation algorithm is the supremum on the length of the minimum Steiner tree found by the algorithm divided by the length of an optimal solution.

$$PerformanceRatio = sup(\frac{smt(T)}{opt(T)}) \tag{2.1}$$

2.4 Known Algorithms and their Performance Ratios

The first approximation for the Steiner tree problem is the MST-approximation algortihm by Takahashi and Hiromitsu [12], which has a performance ratio of 2. Most approximation algorithm following the MST-algorithm have improved on the performance ratio, but they still use the MST-algorithm at their core. For a full overview of the approximation algorithm leading up to Hougardy and Proemel's approximation algorithm and their respective performance ratios see Table 2.1

Table 2.1: Known approximation algorithms and their performance ratios [5]

Author(s)	Performance Ratio	Year
Takahashi, Hiromitsu	2.000	1980
Zelikovsky	1.834	1993
Berman, Ramaiyer	1.734	1994
Zelikovsky	1.694	1995
Proemel, Steger	1.667	1997
Karpinski, Zelikovsky	1.644	1997
Hougardy, Proemel	1.598	1998

3 Implementation

3.1 MST-Algorithm

To implement the MST-algorithm we first build the metric closure $\bar{G}$ of the input graph G. Then we take the subgraph $\bar{G}_T$ of this metric closure containing all required terminal nodes T. In order to compute the metric closure of our input graph we used the algorithm by Floyd Warshall [6], but any other all-pairs-shortest-path algorithm would suffice just fine. Using the subgraph as input for a minimum spanning tree algorithm, we receive a tree which now consists of only terminals and edges that represent the shortest paths between them. In this implementation we are going to use Joseph Kruskal's minimum spanning tree algorithm [10], since it is simple, intuitive and is also easily reusable for building the minimum spanning forest we need to compute the loss of a Steiner tree. Kruskal's algorithm uses a sorted list of all available edges with ascending edge costs as well as a forest structure to construct the minimum spanning tree. It initially adds a tree for each terminal to the forest and then procedes to loop through the sorted edge list and checks for every edge, whether it could connect to different trees. If it does, it is added to the forest, which reduces the number of trees in the forest by one and if it does not, the algorithm just continues through the list. If the number of components in the forest reaches one the algorithm stops, since a MST is already present. The only other way to terminate is, if the entire list of edges has been exhausted and no MST has been found yet. In this case the construction of a MST would have been impossible, since if an edge between two nodes does not exist in the metric closure of a graph they would have to be in two separate components in the original graph. In this case the desired MST cannot be created. With this MST now created the final step is for us to replace the edges of the tree by the corresponding shortest paths in G and the outcome is our Steiner tree approximation.

3.2 Berman-Ramaiyer-Algorithm

The approximation algorithm by Berman and Ramaiyer is split into two phases which both maintain a spanning tree M of T, which is initialized to the output of our MST-algorithm. The first phase is called the evaluation phase and it uses the *prepareChange* procedure, which we are going to look at shortly, to compute two sets of edges called the add-set A and the remove-set R for every j-element subset $\tau \subseteq T$, with j being increased up to a maximum size bounded by the input number k. It then uses these sets to compute a *gain* of τ, by subtracting the cost of a SMT(τ) from the cost of the remove-set. If this *gain* is greater than zero the remove-set is removed from M and every edge of the add-set has its cost reduced by *gain* and is added to M right afterwards. This marks a tentative preference to add SMT(τ). The subset τ, the Remove-Set R and the Add-Set A are added to a stack σ_j to be read in the construction phase.

```
$M = SMT(T);$
for $j = 3$ to $k$ do
    $\sigma_j = \emptyset;$
    foreach $j$-element subset $\tau \subseteq T$ do
        $[R, A] = prepareChange(M, \tau);$
        $gain = cost(R) - smt(\tau);$
        if $gain > 0$ then
            Decrease the cost of each edge $\in A$ by $gain$;
            $M = M \backslash R \cup A;$
            $\sigma_j.push(\tau, R, A);$
        end
    end
end
```
Figure 3.1: Evaluation phase from Berman, Ramaiyer (Fig.2 [1])

The construction phase works on the output of the evaluation phase and starts by initializing the second spanning tree N, which will end up being the submitted solution, with the current version of M. It moves through the stacks σ_j in opposite direction and pops entries from it until the stack is empty. Every entry's data is used to revert the changes made to M by subtracting the add-set and adding the remove-set. If all edges from the add-set are still present in N, they are removed and the SMT(τ) is added in their place. If there are only some, but not all edges from A remaining in N, each of these edges $e \subseteq (A \cap N)$ is replaced in N with the minimal cost edge in M, that connects the two components created by removing e. After these changes have been applied for every entry of every stack the approximated minimal Steiner tree is present in N, while M should have reverted to the original input of the evaluation phase, which was an MST-approximation of the minimal Steiner tree.

The procedure *prepareChange* is a recursive function, which assembles a remove-set R and an add-set A by adding the highest cost edge e that connects two sets each containing at least one terminal node, to R. Removing e creates two components,

```
$N = M$;
for $j = k$ to 3 do
    while $\sigma_j \neq \emptyset$ do
        $[\tau, R, A] = \sigma_j.pop()$;
        $M = M \backslash A \cup R$;
        if $A \subseteq N$ then
            $N = N \backslash A \cup SMT(\tau)$;
        else
            foreach $e \in A \cap N$ do
                Find $f \in M$ of minimal cost such that $N \backslash e \cup f$ connects $T$;
                $N = N \backslash e \cup f$;
            end
        end
    end
end
```

Figure 3.2: Construction Phase from Berman, Ramaiyer (Fig.3 [1])

which are reconnected by creating a substitude edge f that connects one terminal from each component. f is added to the add-set A and its cost will be changed later in the evaluation phase to mark a preference to connect these terminals using a Steiner tree of a subset including these terminals. Finally *prepareChange* will then be applied to the two components to obtain two results that will be joined and returned. The recursion stops when the number of terminals in a component reaches one. In this case the returned remove-set and add-set are both empty.

3.3 Hougardy-Proemel-Algorithm

Similar to the algorithm by Berman and Ramaiyer, the algorithm by Hougardy and Proemel [5] starts of with a Steiner tree initialized with the MST-algorithm. It iteratively applies the $RGH(\alpha)$ algortihm by Karpinski and Zelikovsky [9] with diminishing values for α in every iteration.

$$\vec{\alpha} = (\alpha_1, \ldots, \alpha_k) \text{ with } \alpha_1 \geq \cdots \geq \alpha_k = 0$$

The $RGH(\alpha)$ uses the greedy contraption framework [9] with class K including every k-restricted Steiner tree B with $k \to \infty$ and the criterion function f:

$$f(B) = \frac{d(B) + \alpha * l(B)}{smt(T) - smt(T/B)}$$

```
$prepareChange(M, \tau)$ $R = \emptyset$;
$A = \emptyset$;
if $|\tau| == 1$ then
    return [R, A];
else
    Find an edge $e$ of maximum cost such that both the connected components of
      $M \backslash e$ contain a vertex of $\tau$;
    $[M_1, M_2] = M \backslash e$;
    $[\tau_1, \tau_2]$ are the vertices of $\tau$ in $M_1, M_2$ respectively create an edge $f$ joining
      some $u \in \tau_1$ and some $v \in \tau_2$;
    $cost(f) = cost(e)$;
    $[R_1, A_1] = prepareChange(M_1, \tau_1)$;
    $[R_2, A_2] = prepareChange(M_2, \tau_2)$;
    $R = R_1 \cup R_2 \cup e$;
    $A = A_1 \cup A_2 \cup f$;
    return [R, A];
end
```

Figure 3.3: *prepareChange* from Berman, Ramaiyer (Fig.1 [1])

The full $RGH(\alpha)$ looks like described in figure 3.4. By iteratively expanding the set of included nodes $IRGH$ manages to reduce the worst case performance and therefore the performance ratio and beat out all previously known approximation algorithms. It even includes Steiner nodes, that may increase the cost of the output tree, since their inclusion opens up more options within the next iteration. This leads to the approximation solution of this algortihm being either much better or slightly worse than its input, which is an MST-approximation. While the results of the Berman-Ramaiyer-algorithm remained very close to the input MST-approximation, with little improvements built into it, the $IRGH$ have vastly more Steiner nodes by comparison. Judging by purely its superior performance ratio (which can be referenced in table 2.1) $IRGH$ looks like the better algorithm choice. An additional thing to note for the $IRGH$-algorithm is that the runtime of it is very long compared to the other two algorithms. Since the set $\vec{\alpha}$ could have potentially infinite values α_i the runtime can be infinite as well. For this thesis we ran with the optimal α-values for $k = 3$ most of the time. The results for $k = 6$ proved to be the same for all of the relevant testcases and the computation time was greatly increased. It is important to note that the values, that appear later in this thesis could possibly be improved by simply picking a higher value for k and therefore having more iterations of RGH, but especially, since $IRGH$ already has the best performance ratio and the longest runtime it should not be necessary to increase k in order to achieve representative results for this algorihtm.

```
$S = T$;
while $smt(T) > 0$ do
    foreach k-restricted Steiner tree $B$ do
        if $B$ minimizes $f$ then
            $S = S \cup$ Steiner points in $B$;
            $T = T/B$;
        end
    end
end
return $S$;
```

Figure 3.4: $RGH(\alpha)$ by Karpinski and Zelikovsky [9]

```
$T_0 = T =$ terminals of $G$;
for $i = 1$ to $k$ do
    apply $RGH(\alpha_i)$ to $T_i - 1$ to get $S$;
    $T_i = T_i - 1 \cup \{$Steiner points of $S\}$;
end
return $SMT(T_k)$;
```

Figure 3.5: $IRGH(\vec{\alpha})$ by Hougardy, Proemel [5]

4 Comparison

In this section we are going to analyse the results of the first 15 testcases we got from the SteinLib. We are going to constrast the solutions from the respective approximation with the optimal solution cost, which is referenced in the testcase database [3]. We are also going to look at the solution tree of testcase 41. Its solution for the three algorithms differs substantially enough to nicely showcase their different approaches. The trees were output as a .dot file and visualized using the open source software *graphviz*.

4.1 MST approximation

In the table 4.1 we can see the results of MST approximation for the 15 testcases, that we are going to compare between the different algorithm. While these results are all within close proximity to the optimal solution, with the biggest difference being the cost of the approximation for testcase 45 being 441 larger than the optimal solution. Overall the average performance for MST is quite good, but its main problem is its bad worst-case-performance. Since MST only looks at the shortest paths between pairs of nodes it does not consider adding Steiner nodes, that have a largely reduced shortest cost to multiple terminals, if they do not appear in a shortest path directly. It is therefore largely dependant on the input graph and whilst it produces mostly good results it does not pick up on some potential cost reducing Steiner nodes, on which the other algorithms improve on. We see this reflected in the output tree 4.1, that consists of direct connections between terminals and just one Steiner node, which is in the shortest path between the two respective nodes.

4.2 Berman and Ramaiyer

If we look at the results for the Berman, Ramaiyer algorithm (ref. 4.2) and contrast them with the MST results (ref. 4.1) we can see that a lot of solutions have the same total treecost. This is to be expected because the Berman, Ramaiyer algorithm uses a MST-approximation as its input, which it tries to improve. So everytime it is unable to find such an improvement, its output tree is identical to MST. Since Berman, Ramaiyer looks at subsets of terminals and checks whether the solution tree of that subset and

Table 4.1: Results of MST for 15 testcases from SteinLib [3]

Graph	Opt	MST	MST-Opt
11	1479	1585	106
12	1484	1788	304
13	1381	1676	295
14	1397	1679	280
15	1495	1602	107
21	1175	1471	296
22	1178	1477	299
23	1174	1471	297
24	1161	1473	312
25	1162	1483	321
41	1276	1600	324
42	1287	1674	387
43	1295	1689	394
44	1366	1676	310
45	1310	1751	441

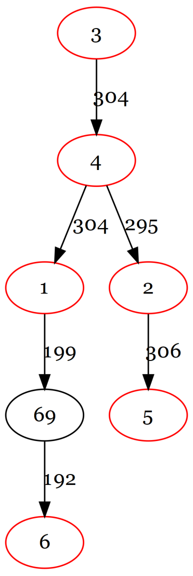

Figure 4.1: Output tree of the MST-approximation algorithm for graph 41

the subsequent removal of redundant edges would lower the cost directly, its treecost is always lower than MST. It is also more dependable when it comes to the worst case performance with a PR of 1.734, but it still misses out on Steiner nodes that efficiently connect other Steiner nodes. When we look at the output tree (fig. 4.2) we can see that it has included an additional Steiner node that connects 3 terminals and reduces the total tree cost by 22.

Table 4.2: Results of Berman, Ramaiyer for 15 testcases from SteinLib [3]

Graph	Opt	BeRa	BeRa%	BeRa-Opt
11	1479	1585	1.072	106
12	1484	1788	1.205	304
13	1381	1676	1.214	295
14	1397	1679	1.202	282
15	1495	1602	1.072	107
21	1175	1471	1.252	296
22	1178	1477	1.254	299
23	1174	1471	1.253	297
24	1161	1473	1.269	312
25	1162	1483	1.276	321
41	1276	1578	1.234	302
42	1287	1658	1.288	371
43	1295	1689	1.304	394
44	1366	1676	1.227	310
45	1310	1751	1.337	441

4.3 Hougardy and Proemel

In the results of Hougardy and Proemel's $IRGH$-algorithm (ref. 4.3) we can see that the treecosts are not strictly better than the MST-approximation. This is due to the fact, that IRGH uses a heuristic that allows even the inclusion of subtrees, that have a negative effect on the tree cost to create more opportunities in the next iteration. This leads to its results being either significantly better than Berman, Ramaiyer or even worse than MST. Its performance ratio is also the best out of the three, since it is less likely to miss good Steiner nodes to include, because upon every addition of Steiner nodes it commences to check for additional Steiner nodes to connect the recently added

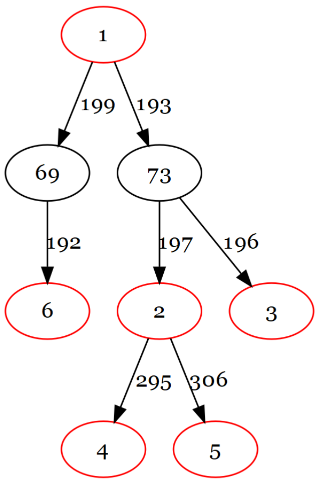

Figure 4.2: Output tree of the Berman, Ramaiyer algorithm for graph 41

ones. This strong incentive to include Steiner nodes is also visible in the output tree of testcase 41 (ref. 4.3). It also shows, that *IRGH* does not stick close to the solution of the initial MST-approximation like Berman, Ramaiyer does. There is not a single edge belonging to the initial MST-approximation left in the output tree of *IRGH*.

Table 4.3: Results of *IRGH* for 15 testcases from SteinLib [3]

Graph	Opt	HoPr	HoPr%	HoPr-Opt
11	1479	1654	1.118	175
12	1484	1759	1.185	275
13	1381	1471	1.065	90
14	1397	1848	1.323	451
15	1495	1895	1.268	400
21	1175	1471	1.252	296
22	1178	1477	1.254	299
23	1174	1471	1.253	297
24	1161	1473	1.269	312
25	1162	1483	1.276	321
41	1276	1463	1.147	187
42	1287	1631	1.267	344
43	1295	1535	1.185	240
44	1366	1546	1.132	180
45	1310	1616	1.234	306

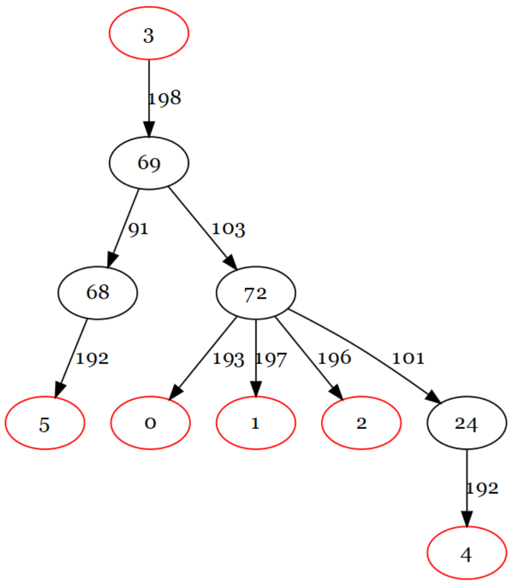

Figure 4.3: Output tree of the *IRGH*-approximation algorithm for graph 41

5 Monotonicity and Critical Payments

To be a good solution for our proposed problem setting an approximation algorithm needs to be stragtegy proof. In order to prove this, we are going to use the following Lemma by Blumrosen and Nisan [2], which we adapt to our setting in which the single minded players are not buyers, but sellers.

Lemma 1. *A mechanism for single-minded sellers, in which losers receive a payment of zero, is incentive compatible, iff it satisfies the following two conditions:*

i) ***Monotonicity****: A seller who wins and sells at price v_i^* keeps winning for any $v_i' < v_i^*$ (with the other sellers offers staying the same)*

ii) ***Critical Payment****: A seller who wins earns the maximum of all values v_i' such that he still wins*

5.1 Monotonicity

In order to prove that the Monotonicity criteria is fulfilled for our algorithm, we need to guarantee for every Edge e that if e is included in the output solution and we lower the cost of it ($d(e)' = d(e) - \epsilon$ with $\epsilon > 0$) it stays included in the solution tree which we obtain from running the algorithm again. To do this we are going to define t as the approximated solution of the algorithm and e, e' as two edges connecting the same nodes with cost $d(e) > d(e')$. With these definitions the monotonicity proof looks like this:

Proof 1. $\forall e, e'$: $e \in t \implies e' \in t$

5.1.1 MST-Approximation

To prove monotonicity for the MST-approximation we are going to walk through our implementation and check every step, where edge cost influences the algorithm, and check what change a reduction in said cost could induce and whether this change could cause the solution to include different edges.

The points where edge cost affects the algorithm are:

1. creation of the metric closure using Floyd-Warshall
2. creation of MST using Kruskal

If both of these algorithm implement monotonicity, we can safely deduce that the MST-approximation also implements monotonicity.

Within the creation of the metric closure we used Floyd-Warshall's all-pairs-shortest-path algorithm, which checks for every triple of nodes a, b, c, whether the sum of the shortest paths $a \to b$ and $b \to c$ is less than the cost of the current shortest path $a \to c$ and updates it accordingly. Within every shortest path that includes e and made it into the metric closure, the substitution of e by e' causes this path to be cheaper by ϵ. Since every other shortest path will have its cost either unchanged or also reduced by ϵ, if it also includes e/e', there is no way for the shortest path to change in that case. Floyd-Warshall therefore implements for our case. It's possible for a shortest path that does not include e to be replaced by a different one that includes e', but this only improves the chances for e' to appear in the solution tree.

The second point to look at is Kruskal's algorithm [10]. Replacing the edge e with e' causes every edge that corresponds to a shortest path which includes e, to be cheaper and therefore appear earlier in the sorted list. If e was included in the output tree, then there are no cheaper edges in the metric closure that connect the two components which e connects. Since $d(e') < d(e)$, there cannot be any cheaper edge than e' either and therefore Kruskal's algorithm implements monotonicity. Since both Floyd-Warshall and Kruskal are implementing monotonicity we can conclude that our MST-approximation fulfills the monotonicity requirement.

5.1.2 Berman-Ramaiyer

For the algorithm by Berman and Ramaiyer we are going to procede like we did with MST and find the points, where the changed edge cost affects the algorithm.

The point where edge cost could potetially affects the algorithm are:

1. Initialization of M to an MST-approximation
2. inclusion of e in the remove-set
3. Price of artificial edges in the add-set
4. MST-approximation of subsets τ
5. Replacement of remaining artificial edges in N

We can quickly prove the first and fourth point since we have already proven that the MST-approximation implements monotonicity. That means if the initial tree M included e, it must include e' and if it did not, then one of the $SMT(\tau)$ of subsets τ had to include e, since it has to be included in the final solution. If one of these trees $SMT(\tau)$ included e, it also has to include e'. Therefore the only thing we need to prove now is that the other three points at which the algorithm is affected don't lead to different subsets being included.

Within the *prepareChange* a lot of changes might occur. Since the *prepareChange* splits the input tree at the maximum cost edge and adds this edge to the remove-set it's possible that e could be included in a remove-set, while e' isn't. But this only changes the fact that now the cost of the remove-set may potentially assume any value within the range $d(R) \to d(R) - \epsilon$. Since e ended up in the solution tree, the set originally either was not removed or e was added in a subset tree $SMT(\tau)$. Therefore e' not being in this remove-set does not change its inclusion into the solution tree. On top of the cost of the remove-set changing, there is also the changing cost of the artificial edge f', that replaces e' if e' is part of the remove-set as well as the possibility of the *gain* changing. The latter is especially crucial, since if the *gain* for a subset τ reaches zero the subset τ is not considered in the construction phase anymore, which leads to $SMT(\tau)$ not being included in the solution. The *gain* g' of a particular subset τ can range after the replacement of e by e' from a minimum of $g - \epsilon$ to $g + \epsilon$ with g being the *gain* with unchanged edges. This is due to the scenario of the subset tree $SMT(\tau)$ not containing e, but containing e'. This would reduce the cost of the tree and therefore increase g' by ϵ. The other scenario impacting g' is with the remove-set computed by *prepareChange* including e' and not the $SMT(\tau)$. In this case the remove-set cost, as well as the *gain*, would be decreased by ϵ. The important thing to note here is that $g' - g$ is only negative, iff the subset tree $SMT(\tau)$ does not include e', therefore the only inclusions that the algorithm may omit are the ones that do not add e' to the solution tree.

The last point where the replacement of e by e' comes into play is in the construction phase, where the artificial edges of the add-sets are replaced by the minimal cost edge in M that connects the created components. The fact that the minimal cost edge is the one that will be included makes it very easy to see that if e was included here e' will be too.

With all these critical points examined we can deduce that Berman and Ramaiyer's algorithm satisfies our monotonicity condition, which would make it a better choice than the MST-approximation, because of its better performance ratio.

5.1.3 Hougardy-Proemel

For the *IRGH* algorithm by Hougardy and Proemel, we are not going to try and prove it, instead we are going to show an example, where the monotonicity condition is violated. A fellow student at the Technical University of Munich has found a case [4], where the *RGH* algorithm by Zelikovsky [14] violates our monotonicity condition. If we use our *IRGH* and choose $\vec{\alpha} = \{0\}$ we end up with one iteration of *RGH* and the heuristic changes like this:

$$f(B) = \frac{d(B) + \alpha * l(B)}{smt(T) - smt(T/B)} \Rightarrow f(B) = \frac{d(B)}{smt(T) - smt(T/B)}$$

Since the second heuristic is the one that is used in the *RGH*-implementation in question, we can use this case to also violate monotonicity for the *IRGH* by Hougardy and Proemel. Because it doesn't implement monotonicity this algorithm isn't strategy proof and therefore can't be reliably used for our problem setting.

5.2 Critical Payments

After trying to prove monotonicity for all three algorithms, we are going to compute critical payments for every edge in the solution tree. To find these critical payments we are going to use binary search. We are going to use the actual cost of the edge in question as our lower bound L for the binary search and we are going to use exponential search to compute the upper bound R. To find this upper bound we double the cost of the edge in question e and check whether it is still included in the solution tree. We repeat this process until it is not included anymore and we return this cost as our upper bound. With this upper bound we iteratively check whether e is still part of the solution tree after changing its cost to $d(e) = m = \lfloor (L + R)/2 \rfloor$. If it is, we put $L = m + 1$ and otherwise we put $R = m$. Since we already know that *IRGH* does not implement monotonicity and that the performance ratio of the MST-approximation is inferior we are only going to compute the critical payments for the algorithm by Berman and Ramaiyer. The following figure 5.1 shows on the left the output of the Berman, Ramaiyer algorithm with unchanged edge costs and on the right it shows the same tree, but with every edge substituted by the critical payment cost of that edge. It is important to keep in mind, that each of these critical payments was computed with the rest of the tree unchanged, but since it would be unreasonable to print the same tree, with just one edge changed, once for every edge, we condensed it all into one tree.

We can clearly see that most of these critical payments are just slightly higher, than the original cost. This is because there are a lot of other viable options in the graph we used, which was testcase 41. If there is an edge in the graph, that is irreplaceable e.g. if

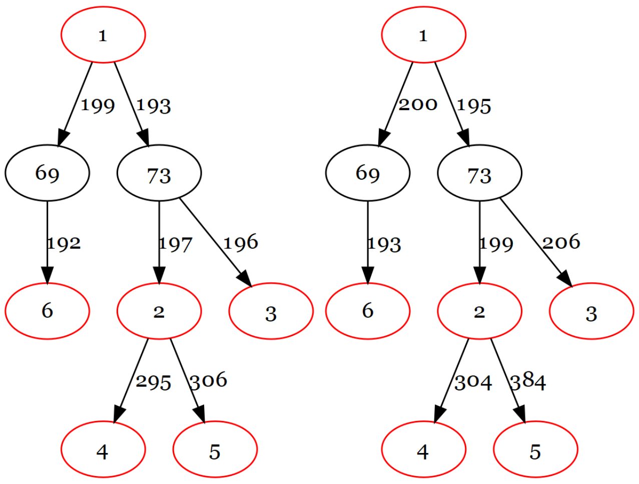

Figure 5.1: Output tree of testcase 41(left) and critical payments(right)

there is a terminal that has a degree of one within the graph, then its critical payment would be ∞, since it would always be included in the solution tree, no matter how high its cost. Now knowing the critical payments we can easily adjust our implementation of the Berman, Ramaiyer algorithm to always pay out the critical payment to the winning seller, which makes out algorithm implement critical payments.

5.3 Incentive Compatibility

Using the Lemma by Blumrosen and Nisan [2] we can conclude that MST-approximation algorithm, as well as the algorithm by Berman and Ramaiyer, are incentive compatible or strategy proof after modifying them so that critical payments are payed out to the winning seller, regardless of the price offered to us. Hougardy and Proemel's *IRGH* algorithm does not implement monotonicity so we can use the aforementioned Lemma to disprove incentive compatibility for *IRGH*. Since we have proven strategy proofness for Berman, Ramaiyer's algorithm, we can assume that the valuations that the sellers give us reflect their true valuations of the edges which they are selling to us.

6 Summary

We started by defining our problem setting, which was a procurement auction with single minded sellers and ourself in the position of an auctioneer trying to procure the minimum cost network that spans a set of required nodes. We then implemented two loss contraction algorithms for this problem setting, which are based on Steiner tree approximation algorithms. Since they both relied on a different algorithm for an initial Steiner tree we provided this algorithm as well. We used the Steiner tree problems at SteinLib [3] as testcases for our algorithms. The simplest one was the MST-approximation algorithm, which was used as the initial working tree for the other two algorithms. While its solution trees for our testcases all had decent average total tree costs, it was bested by both other algorithms when we compare the average total tree cost and the performance ratio. We then looked at the approximation algorithm presented by Berman and Ramaiyer [1], which provided slightly better average tree costs and a significantly better performance ratio, by including Steiner nodes, that appeared in subsets of terminals, into the final solution tree. The last algorithm we looked at was the $IRGH$-algorithm by Hougardy and Proemel [5], which improved the performance ratio once again. Its average tree cost was also better compared to the other two algorithms, but not consistently. In most cases it managed to outperform the other algorithms with its heuristic that helped include more Steiner nodes into the tree, allowing it to find more complex solutions, but in some cases this aggressive inclusion of Steiner nodes led to it increasing upon the cost of the initial MST-approximation. We then proceeded to discern whether the algorithms in question would implement strategy proofness or incentive compatibility. This incentive compatibility is required to guarantee that the dominant strategy for our single-minded sellers is to report their true valuations for the edges they're selling to us. In order for the algorithms to be strategy proof they had to implement monotonicity and had to pay out critical payments to the winning seller. This is due to a Lemma proposed by Blumrosen and Nisan [2]. We were able to prove monotonicity for MST and Berman, Ramaiyer, while we disproved it for Hougardy, Proemel. Since the average tree cost and performance ratio of Berman, Ramaiyer are strictly better than for the MST-approximation we proceeded to only look at Berman, Ramaiyer for the critical payments. We computed these using exponential search and binary search successively. We can now safely conclude that within the algorithms we looked at in this thesis the best algorithm for our proposed setting is the approximation algorithm by Berman and Ramaiyer, since it is strategy proof and has the superior performance ratio compared to the MST-approximation. The full adaptation of this algorithm to our initially proposed setting would include us as the auctioneer, comparing the offered edges using the regular Berman, Ramaiyer algorithm, but changing the payments, that the sellers of the respective edges receive, to the critical payments that we computed.

List of Figures

List of Tables

Bibliography

[1] P. Berman and V. Ramaiyer. "Improved Approximations for the Steiner Tree Problem." In: *Journal of Algorithms 17* (1994), pp. 381–408.

[2] Liad Blumrosen and Noam Nisan. "Combinatorial auctions." In: *Algorithmic game theory* 267 (2007), p. 300.

[3] C. Duin. "Steiner Problems in Graphs." PhD thesis. University of Amsterdam, 1993.

[4] Andreas Guggenbichler. "Loss Contraction Mechanisms for Steiner Trees." Technical Univerity Munich, 2020.

[5] S. Hougardy and H. Prömel. "A 1.598 Approximation Algorithm for the Steiner Problem in Graphs." In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (1999), pp. 448–453.

[6] Stefan Hougardy. "The Floyd–Warshall algorithm on graphs with negative cycles." In: *Information Processing Letters* 110.8-9 (2010), pp. 279–281.

[7] Richard M Karp. "Reducibility among combinatorial problems." In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[8] M. Karpinski and A. Zelikovsky. "New approximation algorithms for the Steiner problems." In: *Journal of Combinatorial Optimization 1* (1997), pp. 47–65.

[9] Marek Karpinski and Alexander Zelikovsky. "New approximation algorithms for the Steiner tree problems." In: *Journal of Combinatorial Optimization* 1.1 (1997), pp. 47–65.

[10] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem." In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[11] Chin Lung Lu, Chuan Yi Tang, and Richard Chia-Tung Lee. "The full Steiner tree problem." In: *Theoretical Computer Science* 306.1-3 (2003), pp. 55–67.

[12] Hiromitsu Takahashi et al. "An approximate solution for the Steiner problem in graphs." In: (1980).

[13] Florian Walch. URL: `https://github.com/fwalch/tum-thesis-latex`.

[14] Alexander Zelikovsky. "Better approximation bounds for the network and Euclidean Steiner tree problems." In: *University of Virginia, Charlottesville, VA* (1996).