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 c: 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 Prömel [3] improves the previous approximation rates by using a generalized version of the parameterized relative greedy heuristic (RGH) by Karpinski and Zelikovsky [6] 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'll therefore assess monotonicity for both implementations and afterwards compute critical payments. To find the critical payment for Edge e we'll 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're 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.

Contents

1 Introduction

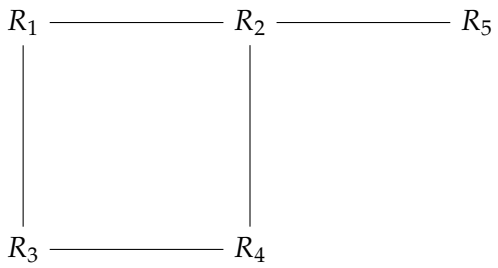

Figure 1.1: An example for a simple drawing.

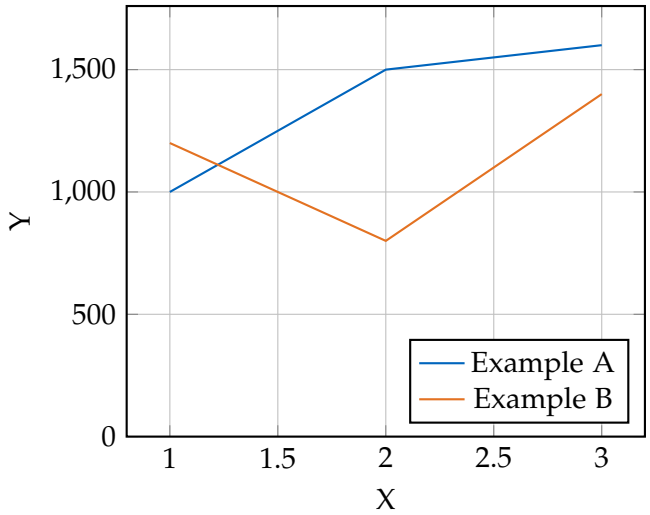

Figure 1.2: An example for a simple plot.

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 be looking 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 aren't 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 [5] 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$\backslash$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, which reached a performance ratio of 2 is the MST-Approximation-Algortihm by Takahashi and Hiromitsu [10]. Since all other algorithms have built upon the existing Foundation and therefore every other approximation algortihm is built around this MST-Approximation. 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 [3]

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 [4], 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 [8], since it's 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 1 and if it doesn't the algorithm just continues through the list. If the number of components in the forest reaches 1 the algorithm stops as 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 doesn't 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 can't 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**
  $\sigma_j = \emptyset$;
end
for $j = 3$ **to** k **do**
  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 entries 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

```
N = M;
for j = k to 3 do
    while σ_j ≠ ∅ do
        [τ, R, A] = σ_j.pop();
        M = M\A ∪ R;
        if A ⊆ N then
            N = N\A ∪ SMT(τ);
        else
            foreach e ∈ A ∩ N do
                Find f ∈ M of minimal cost such that N\e ∪ f connects T;
                N = N\e ∪ f;
            end
        end
    end
end
```

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

R and an Add-Set A by adding the highest cost edge e, that connects two sets, which each contain at least one terminal node, to R. It creates a substitude edge f, which connects the components created by removing e by connecting 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 and the two results will be joined and returned. The recursion stops when the number of terminals in a component reaches one. In this case an 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 [3] starts of with an Steiner tree initialized with the MST-algorithm. It iteratively applies the $RGH(\alpha)$ algortihm by Karpinski and Zelikovsky [7] with diminishing values for α in every iteration.

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

```
prepareChange(M, τ)
if |τ| == 1 then
    R = ∅;
    A = ∅;
    return [R, A];
else
    Find an edge e of maximum cost such that both the connected components of
     M\e contain a vertex of τ;
    [M1, M2] = M\e;
    [τ1, τ2] are the vertices of τ in M1, M2 respectively create an edge f joining
     some u ∈ τ1 and some v ∈ τ2;
    cost(f) = cost(e);
    [R1, A1] =prepareChange(M1, τ1);
    [R2, A2] =prepareChange(M2, τ2);
    R = R1 ∪ R2 ∪ e;
    A = A1 ∪ A2 ∪ f;
    return [R, A];
end
```

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

The $RGH(\alpha)$ uses the greedy contraption framework [7] with class K including every k-restricted Steiner tree B with $k \to \infty$ and the criterion function f:

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

The full $RGH(\alpha)$ the looks like described in 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 to the point. Judging by purely its superior performance ratio 2.1 $IRGH$ looks like the better algorithm choice.

$S = T$;
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 [7]

$T_0 = T =$ 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 [3]

4 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 aren't buyers, but sellers.

Lemma 1. *A mechanism for single-minded sellers in which losers get 0 is incentive compatible if and only if 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*

4.1 Monotonicity

In order to prove that the Monotonicity criterium is fulfilled for our algorithm we need to guarantee for every Edge e, if e is included in the output solution and we lower the cost of e ($d(e)' = d(e) - \epsilon$ with $\epsilon > 0$) it stays included in the solution tree we obtain from running the algorithm again. To do this we are going to define t as the approximated solution of the algorithm in question 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. $\forall e, e'$: $e \in t \implies e' \in t$ □

4.1.1 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 point 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 doesn't 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 [8]. 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, than there aren't any cheaper edges in the metric closure, that connect the two components that e connects. Since $d(e') < d(e)$ there can't be any cheaper edge than e' either and therefore Kruskal's algorithm implements monotonicity and since Floyd-Warshall does so too we can conclude that the MST-approximation fulfills monotonicity.

4.1.2 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 already proved that the MST-approximation implements monotonicity. That means if the initial tree M included

e it must include e' and if it didn't 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 $d(R) \rightarrow d(R) - \epsilon$. Since e ended up in the solution tree, the set originally either wasn't removed or e was added in a subset tree $SMT(\tau)$. Therefore e' not being in this remove-set doesn't change its inclusion into the solution tree.

4.1.3 Hougardy-Proemel

4.2 Critical Payments

4.3 Incentive Compatibility

5 Comparison

5.1 Average Treecost

5.2 Included Steiner Nodes

5.3 Runtime

6 Conclusion

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] 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.

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

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

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

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

[8] 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.

[9] 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.

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

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