where $OPT(b)$ denotes a welfare-maximizing allocation (i.e. an optimal SMT given costs b), $c(OPT(b))$ the corresponding social welfare (i.e. cost of the Steiner tree), and $c(\mathcal{A}(b))$ the welfare achieved with the approximation algorithm $\mathcal{A}$.

Since bidders are self-interested, their reported bids b do not necessarily reflect their true costs c. Instead, bidders try to maximize their quasilinear utilities u_i, i.e., payment received minus true cost: $u_i(b) = p_i(b, f(b)) - c_i$. As a result, a strategyproof mechanism must offer bidders some incentives to reveal their true costs.

Definition 1: Strategyproofness A mechanism $\mathcal{M} = (f, p)$ is strategyproof if for all bidders $i \in E$ and all reported bid tuples $b \in B$ it holds that bidder i has a weakly higher payoff by telling the truth:

$$u_i(c_i, b_{-i}) \geq u_i(b) \tag{2}$$

We also consider the stronger criterion of weak group-strategyproofness, where it is impossible for a group of bidders to make all members of the group better off by lying.

Definition 2 : Weak Group-Strategyproofness[1] A mechanism $\mathcal{M} = (f, p)$ is weakly group-strategyproof if for every set of bidders $I \subseteq E$ and all reported bid tuples $b \in B$ it holds that at least one bidder $i \in E$ has a weakly higher payoff by telling the truth:

$$u_i(c_I, b_{-I}) \geq u_i(b) \tag{3}$$

Finally, to avoid monopoly, we restrict G to be 2-edge-connected, i.e., G remains connected even if any single edge is removed.

With this, we can now formulate the SMT problem as a mechanism design problem: Let $G = (V, E, b)$ be a 2-edge-connected graph. $|V|$ is the number of vertices, $|E|$ is the number of edges/bidders, and b is the vector of reported bid prices. Let $K \subseteq V$ be the set of terminals. Then the objective is to design a polynomial time approximation mechanism which computes an approximately efficient allocation A, and a payment scheme p which makes truthful bidding a dominant strategy, such that p and A form a strategyproof mechanism.

Definition 3: Monotonic allocation rule An allocation rule f of a mechanism $\mathcal{M} = (f, p)$ is monotonic if a bidder i who wins with bid b_i keeps winning for any lower bid $b_i' < b_i$ (for any fixed settings of the other bids).

Definition 4: Critical payment scheme A payment scheme p of a mechanism $\mathcal{M} = (f, p)$ is critical if a winning bidder i receives payment p_i^*, which is her maximum bid allowed for winning: $p_i^* := \sup\{b_i' \in B_i : i \in A(b_i', b_{-i})\}$, where $A(b_i', b_{-i})$ denotes the set of bidders that would have won if the reported bids were (b_i', b_{-i}).

[1] Note that generally, threshold or clock auctions are not strongly group-strategyproof as pointed out in [12] .

https://doi.org/10.30844/wi_2020_c4-bichler

In his seminal paper, Myerson [23] showed that an allocation rule f is implementable (i.e. there exists a payment vector p such that $\mathcal{M} = (f, p)$ is strategyproof) if and only if the allocation rule is monotonic. Moreover, if the allocation rule is monotonic and losing bidders pay 0, a critical payment scheme is the unique payment rule p such that $\mathcal{M} = (f, p)$ is strategyproof. Hence, with single-dimensional types and monotonic approximation algorithms, we can implement an outcome in dominant strategies, if we compute critical payments.

4 Approximation Mechanisms for Single-Dimensional Bidders

In this section we briefly introduce important approximation algorithms for the SMT problem and provide a corresponding critical payment scheme. Finally, we design a deferred-acceptance auction for the SMT problem.

4.1 Approximation Algorithms for the Steiner Minimum Tree

Loss-Contracting Algorithms Loss-contracting algorithms have been the most successful approach to the design of approximation algorithms for the SMT on graphs so far.

Any Steiner tree $S(G, K)$ of G is either a full Steiner tree, i.e., all its terminals are leaves, or can be decomposed into a forest of full Steiner subtrees (full components) by splitting all the non-leaf terminals (splitting a terminal results in two copies of the same terminal). The algorithm by Robins and Zelikovsky [21] builds an MST on the subgraph G_K induced by the terminal set K and repeatedly adds full components to improve the temporary solution. In each iteration, full components are ranked according to their gain (by how much the component improves the current temporary solution) divided by their loss (i.e., the cost committed by adding a component or more precisely its Steiner points). After a full component is added, the temporary solution is improved. This step also involves loss-contracting, a method to make components which are in conflict with added ones less appealing. By these means, the algorithm by Robins and Zelikovsky [21] achieves an approximation ratio of 1.55 if $k \to \infty$ and it is computable in $\mathcal{O}(|K|^k \cdot |V - K|^{k-2} + k \cdot |K|^{2k+1} \log(|K|))$. This is the best approximation algorithm so far, but unfortunately it is not monotonic.

Distance-Network-based Approximations Similarly to the loss-contracting approximation, the general idea of distance-network-based approximation algorithms is to build a MST on a complete subgraph G_K in the first phase. In the second phase, edges in $MST(\overline{G})$ are re-transformed back to edges in G, and an MST is computed on the resulting graph to remove possible cycles. Finally, in the third phase, non-terminal leaves are deleted. This algorithm was proposed by Kou, Markowsky and Berman [18] and runs in $\mathcal{O}(|K||V|^2)$. However, due to the cycles that can occur in the first phase, this standard variant is not monotonic. Mehlhorn [19] designed an algorithm which differs in phase 1. Here, the algorithm first partitions G into Voronoi regions,

https://doi.org/10.30844/wi_2020_c4-bichler

which are then utilized to construct a subgraph of G_K, called $\overline{G}$. It then proceeds with phase 2 and phase 3. This leads to a worst case run time of $\mathcal{O}(|V| \log |V| + |E|)$ and achieves an approximation ratio of $2(1 - 1/l)$ where l is the minimal number of leaves in any SMT (which is naturally bounded above by the number of terminals). It can be shown that the modification of phase 1 leads to a monotonic allocation.

Primal-Dual Approximation Algorithms The approximation algorithm for the SMT problem by Goemans and Williamson [20] follows a primal-dual approach which requires a runtime of $\mathcal{O}(|V|^2 \backslash log\ |V|)$, also has an approximation ratio of 2 and is monotonic.

4.2 Payment Schemes

Since the allocations of the algorithms A^{MH} by Mehlhorn [19] and the primal-dual algorithm A^{PD} by Goemans and Williamson [20] are monotonic, they can be extended to a strategyproof approximation mechanism. To achieve this, the payment scheme p needs to find the critical payment p_i^* for any winner i, such that every reported bid b_i with $b_i \leq p_i^*$ is guaranteed to win, while every reported bid b_i with $b_i > b_i^*$ is guaranteed to lose.

Gualà and Proietti [24] proposed a payment scheme which can be computed in $\mathcal{O}((|V| + |K|^2)|E| \cdot \log \alpha(|E|, |V|))$ for distance-network based approximation algorithms. In this approach the critical payment for e is calculated by adding the difference between the original cost of the shortest path including e and the minimum cost of one of the alternative shortest paths without e which can be efficiently computed using several tweaks [24-26]. Computation of an alternative path in the distance network between two different terminals v_1', v_2' can also be done efficiently by standard sensitivity analysis [27]. For the PD algorithm, a possibility of obtaining critical payments is by using binary search.

Since the payments described above are critical, they can be used in combination with their corresponding approximation algorithm to yield a strategy-proof approximation mechanism for single-dimensional bidders [23].

4.3 Deferred-Acceptance Auctions

Greedy algorithms are an important class of approximation algorithms. A greedy-in algorithm iteratively chooses the best available option based on the current state (i.e., the previous iterations) and adds it to the solution. Contrarily, in the deferred-acceptance auction (DAA), a greedy-out algorithm is used which removes the least favourable alternative from the solution in every iteration.

A greedy-in procedure which greedily accepts edges is not suitable for constructing a Steiner tree, since greedily accepting edges leads to being forced to 'correct' the structure afterwards (e.g.. assuring that Steiner points do not end up as leaves) while within a greedy-out procedure one only needs to assure that it is still possible to construct a Steiner tree based on the remaining edges (i.e. all terminals are still connected). Thus this section describes a greedy-out approximation for the Steiner

https://doi.org/10.30844/wi_2020_c4-bichler

tree problem implemented as a DAA [12]. The DAA is not only strategyproof but also weakly group-strategyproof and therefore provides a form of protection against bidder collusion.

The DAA greedily excludes the least desirable option from the solution until further removal would lead to an infeasible solution. To decide which option should be excluded in each iteration a scoring function is used. A scoring function assigns a value of at least 0 to an option i based only on the cost of i itself and what other options are still available (ignoring the other options' costs, however). In each iteration, an option with the highest assigned score is removed from the allocation, options that cannot be removed without making the resulting solution infeasible receive a score of 0. All remaining options with a score of 0 are accepted in the end. Hence, the algorithm always returns a feasible Steiner tree at the end.

The payment $p(e)$ for an option i is calculated the moment we cannot exclude i from the solution any more, i.e. the moment we assign a score of 0 to it. The payment is equal to the bid i could have stated such that her score would have been equal to an option that was removed in the last iteration before her score was set to 0.

In a network procurement context, the set of options is the set of edges E. An edge cannot be excluded from the solution if its removal would lead G to split into two separate connected components, each of which contains at least one terminal. To account for the specific requirements of the network procurement context, we analyze three scoring functions in our experimental analysis:

1. the weight of the edge
2. the weight of the edge divided by the number of adjacent edges
3. the weight of the edge divided by the edge betweenness centrality of the edge, where all pairs of vertices are considered

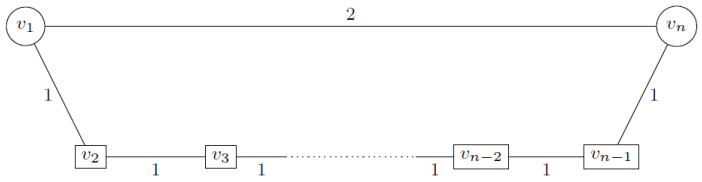
We calculate betweenness centrality by using an algorithm due to Brandes [28] on a variant G_u of G where all edges have weight 1, i.e. G_u is the unweighted version of G. This is necessary, since due to incentive reasons a scoring function may only take the respective bid and the underlying graph structure into account, not the bids of other active bidders. Since in our environment bids are the cost of edges, when calculating the score of an edge e, we must ignore the costs of all other edges $e' \neq e$ in our calculations. In the following, let DAA_w (DAA_a; DAA_c) denote the DAA with scoring by weight (divided by adjacent edges; betweenness centrality). The DAA for the SMT problem runs in $\mathcal{O}(|E|^3 + |E|^2|V| + |E|t)$ including payment calculation where t is the time necessary to update the scores in each iteration.

Greedy algorithms are usually fast, but can lead to arbitrarily bad results compared to an optimal solution for some problems. For knapsack auctions and general combinatorial auctions with single-minded bidders, both of which are maximizing social welfare, [15] prove approximation ratios for the allocation algorithms used in their DAAs. However, their techniques are not directly applicable to the network procurement setting. For instance, consider the three weight functions discussed above and a network as given in Figure 1. The network consists of n nodes $v_1, \dots, v_n$, two of which (v_1 and v_n) are terminals. The optimal Steiner Tree consists of only keeping edge (v_1, v_n), but under DAA this edge is rejected first under all weight

https://doi.org/10.30844/wi_2020_c4-bichler

functions, forcing the algorithm to accept all other $n-1$ edges in order to remain connected. This leads to an approximation ratio of $(n-1)/2$ proving the impossibility of a constant-factor approximation ratio. It remains an open question whether there exists a weight function that allows for such an approximation ratio, however this seems doubtful.

Figure 1. An example where DAA leads to an approximation ratio of (n-1)/2



5 Experimental Evaluation

We describe the data set before we discuss our results. All algorithms were implemented in Java. The experiments were executed on a laptop with Intel core i5-6600k (4 cores, 3.5 GHz) and 8GB RAM.

For the approximation mechanism based on [19] and the primal-dual algorithm [20], we computed the payments as described in Section 4.2. For the former, we employed the payment scheme for distance-network based approximation algorithms by Gualà and Proietti [24] and for the latter we calculated the payments based on binary search. For the DAA we use the threshold payments which are dynamically updated throughout the run of the algorithm as described in Section 4.3. Finally, we also included the VCG mechanism [29-31] as a baseline. We used the send-and-split method [32] as implemented by Iwata [33] to determine optimal solutions.

5.1 Data

Experiments are conducted on set I080 of the SteinLib Testdata Library [34][2]. However, instances which are not 2-edge-connected are not considered since a monopoly edge would be worth infinite amounts of money. Instances with names ending on $0x$ or $3x$ are thus not considered, which leaves us with 60 instances all of which have 80 vertices. We cluster instances with the same number of terminals and edges, this gives twelve clusters consisting of five instances each. As a summary, only the mean values for each cluster are reported in this section. The full list of results is provided in the appendix. Overall, we have 60 instances and 6 algorithms resulting in 360 experiments.

[2] http://steinlib.zib.de/showset.php?I080

https://doi.org/10.30844/wi_2020_c4-bichler

5.2 Results

Allocative Efficiency. In **Table 2** and **Table 3**, we present the mean efficiency of the five instances for each of the twelve terminal-edge combinations we considered in our experimental evaluation. While the approximation algorithm by Robins and Zelikovsky [21] is not monotonic and thus cannot be extended to an approximation mechanism, it is still interesting to compare its allocation efficiency to the other algorithms in a complete information setting. Overall, DAA_c and RZ were the best performing algorithms and the scoring function based on the betweenness centrality came out to be the best scoring function for DAAs. With a paired Wilcoxon rank sum test the differences in efficiency between DAA_a and RZ ($p = 0.88$), between DAA_c and RZ ($p = 0.0005$), and those between MH and PD ($p = 0.155$) were not significant at $p < 0.0001$, while all other pairwise comparisons were significant at this level. We also analyse differences in efficiency using a linear regression with efficiency as dependent variable, the algorithm, the number of edges and terminals as covariates. With the DAA_c as baseline, the differences to this greedy algorithm were positive and significant at the following levels: DAA_w ($p < 0.0001$), MH ($p < 0.0001$), PD ($p < 0.0001$), DAA_a ($p < 0.01$), and RZ ($p < 0.01$).

Table 2. Average efficiency

E,K	350,6	3160,6	632,6	350,8	3160,8	632,8
RZ (k=3)	1.07	1.23	1.13	1.11	1.26	1.17
MH	1.16	1.26	1.30	1.29	1.31	1.34
PD	1.15	1.26	1.28	1.31	1.31	1.33
DAA_w	2.27	2.27	2.19	2.25	2.40	2.13
DAA_a	1.21	1.47	1.27	1.17	1.35	1.19
DAA_c	1.30	1.14	1.24	1.18	1.10	1.16

Table 3. Average efficiency (continued)

E,K	350,16	3160,16	632,16	350,20	3160,20	632,20
RZ (k=3)	1.19	1.30	1.24	1.21	1.32	1.24
MH	1.25	1.39	1.30	1.30	1.40	1.33
PD	1.26	1.39	1.30	1.31	1.40	1.32
DAA_w	1.67	1.85	1.83	1.59	1.71	1.54
DAA_a	1.14	1.26	1.11	1.11	1.18	1.10
DAA_c	1.16	1.06	1.10	1.11	1.06	1.08

Let us now report averages for different subgroups of the experiments in more detail. The algorithm by Robins and Zelikovsky performs best for sparse instances, on average finding a solution only 25% worse than the optimum and even solutions as good as 1.01 times the optimum (instance $I080-015$ of SteinLib). Moreover, it performs well for complete graphs, too (1.3 approximation ratio). Mehlhorn's algorithm and the primal-dual algorithm achieve similar results ($130\% - 140\%$ of the optimum).

https://doi.org/10.30844/wi_2020_c4-bichler

The performance of the DAA heavily depends on the scoring function. Using only the weight of an edge c_e as a score, allocative efficiency is never better than 1.48 times of the optimum, usually worse than 1.6 times of the optimum and even 2.97 in one instance ($I080-022$). It seems clear, that without taking into account the structure of the graph, the greedy algorithm employed in DAA_w can not compete with more sophisticated methods. Even in later stages of the DAA, edges are only selected based on their individual cost without considering the possible paths this edge is a part on. If we use edge weight divided by number of adjacent edges as scoring function, the DAA already performs better than the primal-dual algorithm for most sets of instances and even achieves results that are better than the results of the algorithm by Robins and Zelikovsky for instances with 16 or more terminals. Results are between 1.06 and 1.6 times the optimum value with 39 out of 60 results lying between 1.1 and 1.55 times the optimum. This variant performs significantly better than the DAA_w (on a significance level of 0.1%).

Finally, the DAA_c generally provides better results on average than the algorithm by Robins and Zelikovsky for $k = 3$. On sparse instances with only 6 terminals the algorithm by Robins and Zelikovsky still performs slightly better. For these instances, the DAA_c computes solutions that are up to 1.55 times the optimum value, although most of the solutions achieve an approximation ratio of 1.3 or lower.

Allocative Efficiency depending on the number of terminals. The results show clearly that performance of all DAA variants increases as the number of terminals grows. This is to be expected since a greedy-out procedure actually solves MST optimally and the SMT problem becomes more like the MST problem for an increasing number of terminals (in the limit, when all vertices are terminal, they are identical). Contrarily, all other algorithms perform worse the more terminals are present in the graph. Moreover, it can be seen that efficiency of DAA_a and DAA_c is nearly identical for sparse graphs. In sparse graph, the number of possible paths between two nodes is very limited. Since an edge e with a lot of adjacent edges naturally allows for more paths (and hence also more shortest paths) to pass through e, the betweenness centrality of e is very dependent on its adjacent edges. Therefore, the DAAs with the corresponding scoring functions perform very similarly.

Allocative Efficiency depending on density. It can be seen that the efficiency of the DAA_w and the DAA_a decreases as instances with an increasing number of edges are considered. The algorithm by Robins and Zelikovsky experiences the same behavior. However, the opposite is true for the DAA_c.

Table 2 and **Table 3** show that the DAA_a and the DAA_c find similarly efficient solutions for sparse graphs, where no significant difference was observed. For complete graphs, every node has the same number of neighbors at the beginning. Therefore, before that crucial number of edges has been removed, the DAA_a makes the same choices the DAA_w makes. In this case, the efficiency of the DAA_a is substantially worse (up to 1.39 times the optimum). The solutions found by the DAA_c are never worse than 1.09 times the optimum and in eight out of ten instances at most 6% worse than the optimal value for complete graphs with 16 or 20 terminals.

https://doi.org/10.30844/wi_2020_c4-bichler

Runtime. In the following, we discuss the combined runtime required for the approximation mechanism to obtain both, allocation and payments (cf. **Table 4** and **Table 5**). The differences between DAA_w and DAA_c ($p = 0.9794$) as well as between DAA_a and DAA_w ($p = 0.002$) are not significant at a level of $p < 0.0001$ using a Wilcoxon rank sum test, the other pairs were significantly different. Only the mechanism based on Mehlhorn's algorithm and the primal dual approach are significantly different from VCG ($p < 0.0001$). Our experiments show that PD and MH are by far the fastest. Runtimes observed are lower than 13s on average on the set of instances with high numbers of both terminals and edges (**Table 5**). Performance for lower numbers of terminals or edges is even better. Arguably, the more advanced payment computation used within MH leads to faster completion than calculating prices for PD, although we observed higher allocation runtime of the primal dual algorithm when prices were not considered. Interestingly, Mehlhorn's algorithm with payment is faster than the primal-dual algorithm despite larger allocation time. This is evidently due to the more enhanced payment computation.

The runtime required by all DAA variants is very dependent on the density of the graph while it scales very well with the number of terminals. In our test instances, the computation time even decreases with an increasing number of terminals; in contrast to all other mechanisms. VCG in particular, is very sensitive to higher number of terminals. Calculating exact solutions for SMT problems in the case of 20 terminals proved to be computationally inefficient, requiring a factor of 40 up to 28.000 of the runtime as compared to the best performing DAA variants. Differences between the scoring functions are very small. Interestingly, the DDA_w is slower than the DAA_c on complete graphs. This might be due to the fact that edges scoring 0 early in the computation do not need to be assessed in later iterations and the DAA_c possibly finding more of these edges earlier than the DAA_w.

Table 4. Runtime combined means

E,K	350,6	3160,6	632,6	350,8	3160,8	632,8
MH	0.24	0.46	0.17	0.19	0.61	0.29
PD	0.28	3.14	0.56	0.59	4.66	0.92
DAA_w	1.37	1553.71	10.29	1.29	1678.19	10.15
DAA_a	1.39	1167.31	10.65	1.26	1392.11	10.35
DAA_c	2.34	1476.72	12.89	2.19	1596.73	12.57
VCG	2.05	1.80	1.71	2.82	2.34	2.54

https://doi.org/10.30844/wi_2020_c4-bichler

Table 5. Runtime combined means (continued)

E,K	350,16	3160,16	632,16	350,20	3160,20	632,20
MH	0.52	1.01	0.33	0.33	1.26	0.49
PD	1.01	9.91	1.44	1.93	12.97	1.90
DAA_w	1.35	1668.95	10.38	1.39	1478.79	9.84
DAA_a	1.33	1368.65	10.28	1.39	1230.18	9.89
DAA_c	2.22	1547.89	12.91	2.32	1377.29	12.23
VCG	638.87	532.76	646.89	67544.54	58556.20	67595.38

6 Conclusions

The design of electronic markets has a number of challenges due to the fact that multiple self-interested parties participate. Incentive-compatibility is an important property in order to select the efficient allocation in the presence of strategic bidders. Unfortunately, the well-known VCG mechanism requires exact solutions of the allocation problem, which is often intractable. The SMT is a case in point. It is NP-hard and one cannot expect to solve this problem exactly for realistic problem sizes.

There is an established literature of approximation algorithms for SMT problem. However, not all approximation algorithms can be extended to strategyproof approximation mechanisms. The best known algorithm by Robins and Zelikovsky violates monotonicity, a necessary condition for strategyproofness. However, the algorithms by Mehlhorn [19] and the primal-dual algorithm by Goemans and Williamson [20] are monotonic. Adding a critical payment rule to these algorithms leads to polynomial-time and strategyproof 2-approximation mechanisms.

We designed a deferred-acceptance auction for the SMT problem and analyzed several scoring functions. Overall, the results show that DAAs (foremost DAA_c) yield very good results (efficiency and cost) at low computation times. The DAA combines high average efficiency (as compared to MH, PD, and even RZ) with a low runtime compared to VCG. Moreover, the DAA is the only mechanism that offers weak group-strategyproofness. This property makes the DAA an appealing choice for network procurement applications, where collusion is an issue. While the worst-case approximation ratio can be quite low, the average-case solution quality is remarkably high, as shown in our numerical experiments based on the SteinLib.

While the VCG mechanism is the approach an auctioneer should follow to minimize payments it experiences a steep increase in runtime as the number of terminals grows. For these instances, the MH mechanism exhibits low payments and runtime, but it is less efficient compared to others.

Finally, it is important to note that the performance of the DAA is dependent on the scoring function. On average, the efficiency of algorithms with a scoring function based on betweenness centrality is very good and competitive with the best approximation algorithms for the SMT problem.

https://doi.org/10.30844/wi_2020_c4-bichler

7 Acknowledgements

The work of Richard Littmann was funded by Deutsche Forschungsgemeinschaft (DFG), GRK 2201.

References

1. Adomavicius, G., Gupta, A.: Toward comprehensive real-time bidder support in iterative combinatorial auctions. Information Systems Research 16, 169-185 (2005)
2. Adomavicius, G., Gupta, A., Sanyal, P.: Effect of information feedback on the outcomes and dynamics of multisourcing multiattribute procurement auctions. Journal of Management Information Systems 28, 199-230 (2012)
3. Bichler, M., Gupta, A., Ketter, W.: Research commentary—designing smart markets. Information Systems Research 21, 688-699 (2010)
4. Goetzendorff, A., Bichler, M., Shabalin, P., Day, R.W.: Compact bid languages and core pricing in large multi-item auctions. Management Science 61, 1684-1703 (2015)
5. Bichler, M., Shabalin, P., Pikovsky, A.: A computational analysis of linear price iterative combinatorial auction formats. Information Systems Research 20, 33-59 (2009)
6. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games and Economic behavior 35, 166-196 (2001)
7. Vazirani, V.V.: Approximation algorithms. Springer Science & Business Media (2013)
8. Karp, R.M.: Reducibility among combinatorial problems. Complexity of computer computations, pp. 85-103. Springer (1972)
9. Öncan, T., Cordeau, J.-F., Laporte, G.: A tabu search heuristic for the generalized minimum spanning tree problem. European Journal of Operational Research 191, 306-319 (2008)
10. Xu, J., Chiu, S.Y., Glover, F.: Tabu search heuristics for designing a Steiner tree based digital line network. University of Colorado (1995)
11. Contreras, I., Fernández, E.: General network design: A unified view of combined location and network design problems. European Journal of Operational Research 219, 680-697 (2012)
12. Milgrom, P., Segal, I.: Clock Auctions and Radio Spectrum Reallocation. (2018)
13. Leyton-Brown, K., Milgrom, P., Segal, I.: Economics and computer science of a radio spectrum reallocation. Proceedings of the National Academy of Sciences 114, 7202-7209 (2017)
14. Li, S.: Obviously strategy-proof mechanisms. American Economic Review 107, 3257-3287 (2017)
15. Dütting, P., Gkatzelis, V., Roughgarden, T.: The performance of deferred-acceptance auctions. Mathematics of Operations Research 42, 897-914 (2017)
16. Newman, N., Leyton-Brown, K., Milgrom, P., Segal, I.: Assessing economic outcomes in simulated reverse clock auctions for radio spectrum. arXiv preprint arXiv:1706.04324 (2017)
17. Takahashi, H.M., A: An approximate solution for the Steiner problem in graphs. Math. Japonica. 6, 573-577 (1990)

https://doi.org/10.30844/wi_2020_c4-bichler

18. Kou, L., Markowsky, G., Berman, L.: A fast algorithm for Steiner trees. Acta informatica 15, 141-145 (1981)
19. Mehlhorn, K.: A faster approximation algorithm for the Steiner problem in graphs. Information Processing Letters 27, 125-128 (1988)
20. Goemans, M.X., Williamson, D.P.: The primal-dual method for approximation algorithms and its application to network design problems. Approximation algorithms for NP-hard problems 144-191 (1997)
21. Robins, G., Zelikovsky, A.: Tighter bounds for graph Steiner tree approximation. SIAM Journal on Discrete Mathematics 19, 122-134 (2005)
22. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved LP-based approximation for Steiner tree. In: Proceedings of the forty-second ACM symposium on Theory of computing, pp. 583-592. ACM, (Year)
23. Myerson, R.B.: Optimal auction design. Mathematics of operations research 6, 58-73 (1981)
24. Gualà, L., Proietti, G.: A truthful (2–2/k)-approximation mechanism for the Steiner tree problem with k terminals. In: International Computing and Combinatorics Conference, pp. 390-400. Springer, (Year)
25. Buchsbaum, A.L., Kaplan, H., Rogers, A., Westbrook, J.R.: Linear-time pointer-machine algorithms for least common ancestors, mst verification, and dominators. arXiv preprint cs/0207061 (2008)
26. Pettie, S., Ramachandran, V.: Computing shortest paths with comparisons and additions. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 267-276. Society for Industrial and Applied Mathematics, (Year)
27. Tarjan, R.E.: Sensitivity analysis of minimum spanning trees and shortest path trees. Information Processing Letters 14, 30-33 (1982)
28. Brandes, U.: A faster algorithm for betweenness centrality. Journal of mathematical sociology 25, 163-177 (2001)
29. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. The Journal of finance 16, 8-37 (1961)
30. Clarke, E.H.: Multipart pricing of public goods. Public choice 11, 17-33 (1971)
31. Groves, T.: Incentives in teams. Econometrica 41, 617-631 (1973)
32. Erickson, R.E., Monma, C.L., Veinott Jr, A.F.: Send-and-split method for minimum-concave-cost network flows. Mathematics of Operations Research 12, 634-664 (1987)
33. Iwata, Y.S., Takuto: Send-and-split method for minimum-concave-cost network flows. (2018)
34. Koch, T.M., Alexander, Voß, S.: SteinLib: An Updated Library on Steiner Tree Problems in Graphs. (2000)

https://doi.org/10.30844/wi_2020_c4-bichler

8 Appendix: Approximation Algorithms for the minimum Steiner tree

In 8.1 we discuss loss-contraction algorithms on the basis of the algorithm by Robins and Zelikovsky [21], in Section 8.2 we consider distance-network based approaches, in Section 8.3 the primal-dual approximation, and in Section 8.4. the DAA.

8.1 Loss-Contracting Approximation: The Algorithm by Robins and Zelikovsky

Any Steiner tree $S(G,K)$ of G is either a full Steiner tree, i.e., all its terminals are leaves, or can be decomposed into a forest of full Steiner subtrees (full components). A k-restricted full component F is a full component with $k \geq 3$ terminals. By $C_l[F]$ we denote the loss-contracted full component of F. We define the gain and loss of a full component F formally and then describe the execution of Algorithm 1 below.

Definition 5 (Gain and Loss of a Full Component (Robins and Zelikovsky, 2005))
Let T be a tree spanning K and F be an arbitrary full component of G given K. Let $T[F]$ be a minimum cost graph in $F \cup T$ which contains F completely and spans all terminals in K. This means $T[F]$ is the result of replacing a part of the tree T with the full component F.
Then the Gain of F w.r.t. T is the cost difference between T and $T[F]$:

$$gain_T(F) = cost(T) - cost(T[F]) \tag{4}$$

The Loss of F is a minimum-cost subforest of F containing a path from each Steiner point in F to one of its terminals: $Loss(F) = MST(F \cup E_0(F)) \setminus E_0(F)$, where $E_0(F)$ denotes a complete graph containing all terminals of F, with all edge costs being 0. It follows that

$$loss(F) = cost(MST(F \cup E_0(F)) \setminus E_0(F)) \tag{5}$$

Data: 2-connected graph $G = (V, E, b)$, terminal set $K \subseteq V$, an integer k with $3 \leq k \leq |K|$
Result: A k-restricted Steiner tree $S(G)$ in G spanning K
Compute G_V and G_K
$T = MST(G_K)$
repeat
  Find a k-restricted full component F maximizing $r = gain_T(F)/loss(F)$
  $G_K = G_K \cup F$
  $T = MST(T \cup C_l[F])$
until $r \leq 0$
$S(G,K) = MST(G_K)$
Replace artificial edges in $S(G,K)$
Cut Steiner point leaves of $S(G,K)$
return $S(G,K)$

Algorithm 1: Approximation Allocation Algorithm A^{RZ}

https://doi.org/10.30844/wi_2020_c4-bichler

The algorithm starts by computing the complete graph G_V based on shortest paths, its subgraph G_K induced by the terminal set K (Line 1) and the MST on G_K (Line 2). Afterwards, the gain-over-loss ratios for all k-restricted full components are computed. After choosing the full component with the highest gain-over-loss ratio, the selected component is added to G_K (Line 5). The component is also added to T in loss-contracted form $C_l[F]$ (Line 6). To contract the loss of a full component F, we merge every connected tree of the forest $Loss(F)$ into a single vertex, the respective terminal of the component. Two terminals are connected in $C_l[F]$ if their respective components in $Loss(F)$ have an adjacent edge in F and the cost of the edge in $C_l[F]$ is equal to the cost of the respective edge in F.

After $C_l[F]$ was added to T, an MST is built on $T \cup C_l[F]$. By improving T the gain-over-loss ratio for the remaining full components is decreasing. Eventually, all components will have a gain-over-loss ratio of at most zero. At this point, the algorithm computes the MST(G_K) (Line 8), transforms all its artificial edges back into original edges, i.e. replaces artificial edges by the respective shortest path, and cuts leaves which are Steiner points (Lines 9, 10).

8.2 Distance-Network-based Approximation: The Algorithm by Mehlhorn

Mehlhorn's algorithm [19] partitions G into Voronoi regions, which are then utilized to construct a subgraph of G_K, called $\bar{G}$, in the first phase. In the second phase, edges (shortest paths) in the MST are decomposed into edges in E, and a MST is computed on the resulting graph to remove possible cycles. Finally, in the third phase, non-terminal leaves are deleted. We give definitions for Voronoi regions and $\bar{G}$, and provide a short pseudo code representation of the algorithm (Algorithm 2).

Definition 6 (Voronoi Regions $V(s)$) Given a general graph $G = (V, E, b)$ and the set of terminals $K \subseteq V$, the Voronoi region $V(s)$ of a terminal $s \in K$ contains all vertices $v \in V$ for which the shortest path $sp(s, v) \leq sp(t, v)$ for all $t \in K$. We break ties randomly, such that each vertex v uniquely belongs to one such region.

Definition 7 (Distance Network based on V) Let $\bar{G} = (K, E_{\bar{G}}, b_{\bar{G}})$ be the distance network with edges and weights as follows:

$$(s,t) \in E_{\bar{G}} \Leftrightarrow \exists\, (u,v) \in E \text{ such that } u \in V(s) \text{ and } v \in V(t) \quad (6)$$

$$b_{\bar{G}}(s,t) = min\{sp(s,u) + b(u,v) + sp(v,t) : u \in V(s), v \in V(t), (u,v) \in E\} \quad (7)$$

Data: 2-connected graph $G = (V, E, b)$, terminal set $K \subseteq V$
Result: A Steiner tree $S(G, K)$ in G spanning K
01 Compute Voronoi regions of G and generate $\bar{G}$
02 $S(G, K) = MST(\bar{G})$
03 Replace artificial edges in $S(G, K)$
04 Cut non-terminal leaves of $S(G, K)$
05 **return** $S(G, K)$

Algorithm 2: Approximation Allocation Algorithm A^{MH}

https://doi.org/10.30844/wi_2020_c4-bichler

8.3 Primal-Dual Approximation Algorithms

This section describes the general approach for primal-dual approximations and the approximation algorithm for the minimum Steiner tree problem by Goemans and Williamson [20]. Many problems in graph theory can be reduced to the hitting set problem. For a groundset E with cost $c_e \geq 0$ for every element $e \in E$ and subsets $T_1, T_2 \ldots T_n \subseteq E$, the hitting set problem is to find a subset $A \subseteq E$ of minimal cost such that $A \cap T_i \neq \emptyset$ for all subsets $i = \{1, \ldots n\}$. The primal integer program for the hitting set problem can be formulated as follows:

$$\begin{aligned} &\text{Min } \textstyle\sum_{e \in e} c_e x_e \\ &\text{subject to } \textstyle\sum_{e \in T_i} x_e \geq 1, \ \forall i \\ &\qquad\qquad x_e \in \{0,1\} \end{aligned}$$

To obtain the relaxation, simply the constraint $x_e \in \{0, 1\}$ needs to be relaxed to $x_e \geq 0$. The corresponding dual program is stated below:

$$\begin{aligned} &\text{Max } \textstyle\sum_i y_i \\ &\text{subject to } \textstyle\sum_{i: e \in T_i} y_i \leq c_e, \ \forall e \in E \\ &\qquad\qquad y_i \geq 0 \ \ \forall i \end{aligned}$$

To obtain an α-approximation we compute a solution $\bar{x}$ to the primal integer program and a solution y to the dual of the relaxed primal program such that $\sum_{e \in E} c_e \bar{x}_e \leq \alpha \sum_{i=1}^{n} y_i$.

Mapping the hitting set problem to the minimum Steiner tree problem is straightforward: the ground-set is given by the edges E of the graph and c_e is the cost of the respective edge $e \in E$. Let S_i be a subset of vertices that contains at least one, but not all terminals, i.e. a cut. When all cuts are crossed, the solution is a feasible allocation for the minimum Steiner tree problem. By definition, the edges adjacent to exactly one vertex $v \in S_i$ are the edges crossing the cut S_i. Let $\delta(S_i)$ denote the set of these edges. Let $T_i = \delta(S_i)$.

Algorithm 3 describes the necessary steps to compute A. During the initialization, A is empty and all dual variables y are set to 0 (Lines 1, 2). In each iteration, we compute U to contain all T_k that are unsatisfied and minimal, i.e. there is no unsatisfied set T_j with $T_j \subset T_k$ (Line 5). Afterwards, the dual variables y_k corresponding to set in U are increased (loaded) until one of the constraints holds with equality (it goes ”tight”, Line 6). The corresponding element e is then added to the solution. If the allocation A is feasible, the algorithm stops and conducts a reverse deletion (Lines 9 – 14). In this phase, edges are assessed in regard to their necessity in reversed order (LIFO). Unnecessary edges either connect a Steiner point as a leaf or close a cycle. In either case, the edge is not contributing to the solution (apart from inflicting costs). Finally, the solution is returned.

https://doi.org/10.30844/wi_2020_c4-bichler

Data: 2-connected graph $G = (V, E, b)$, terminal set $K \subseteq V$
Result: A Steiner tree $S(G)$ in G spanning K
01 $y = 0\ \forall y$
02 $A_0 = \emptyset$
03 $i = 0$
04 **while** A_i not feasible **do**
05 Choose violated sets U
06 Increase y_k uniformly for all $T_k \in U$ until $\exists e_i \notin A_i$ s.t. $\sum_{i:e_i \in T_i} y_i = c_{e_i}$
07 $A_i = A_i \cup \{e_i\}$
08 $i = i + 1$
09 **end**
10 $A' = A_{i-1}$
11 **for** $i;\ i \geq 0; i = i - 1$ **do**
12 **if** $A' \setminus \{e_{t_i}\}$ still feasible **then**
13 $A' = A' \setminus \{e_{t_i}\}$
14 **end**
15 **end**
16 **return** A'

Algorithm 3: Approximation Allocation Algorithm A^{PD}

8.4 Deferred Acceptance Auctions for Steiner Trees

The DAA greedily excludes the least desirable option from the solution until further removal would lead to an infeasible solution. Algorithm 4 shows an implementation of the DAA for the Steiner tree problem. To decide which option should be excluded in each iteration a scoring function is used. In each iteration, the scoring function assigns a value of at least 0 to an edge e (Line 2) based only on the cost of e itself and what other options are still available (ignoring the other options' costs, however). Edges that cannot be removed without making the resulting solution infeasible receive a score of 0. When a score of 0 is assigned to an edge the first time we compute the payment $p(e)$ (Lines 3, 4) This payment is equal to the cost e could have stated such that her score would have been equal to an edge that was removed in the last iteration before her score was set to 0. If there are edges with a positive score left the highest scoring edge is removed from the solution (Line 9), otherwise the algorithm terminates.

https://doi.org/10.30844/wi_2020_c4-bichler

Data: 2-connected graph $G = (V, E, b)$, terminal set $K \subseteq V$, scoring function s
Result: A Steiner tree in G spanning K
01 **for** each edge e **do**
02 assign score $s(e)$ to e
03 **if** $s(e) = 0$ **then**
04 compute payment $p(e)$
05 **end**
06 **if** highest score equals 0 **then**
07 **return** remaining edges (Steiner tree)
08 **end**
09 remove e with the highest score
10 **end**
11 **return** remaining edges

Algorithm 4: Deferred-Acceptance Auction

https://doi.org/10.30844/wi_2020_c4-bichler