A Fast Heuristic for Stochastic Steiner Tree Problems

Berend Markhorst[1,2] [0000-0002-3383-6862], Alessandro Zocca[2] [0000-0001-6585-4785], Joost Berkhout[2] [0000-0001-5883-9683] and Rob van der Mei[1,2] [0000-0002-5685-5310]

Abstract Network design under uncertainty arises in countless real-world settings and can be captured by the Stochastic Steiner Tree Problem (SSTP). Although there are a few approaches specifically tailored to this stochastic optimization problem, there are considerably more state-of-the-art heuristics for its deterministic variant, the Steiner Tree Problem (STP). In this work, we show how to leverage an existing STP heuristic in building a novel method for solving its stochastic variant, the SSTP. This approach is a powerful, yet simple and easy-to-implement way of solving this complex problem. We test our method using benchmark instances from the literature. Numerical results show considerably faster computation times compared to the state-of-the-art, with a gap of approximately 5%.

Key words: Stochastic Steiner Tree Problem, Heuristic, Combinatorial Optimization

1 Introduction

The two-stage Stochastic Steiner Tree Problem (SSTP) and its variants have a wide range of applications in different industries, such as telecommunication network design [1, 2], wire routing in Very Large-Scale Integration circuits [3], studying protein-interaction networks in bioinformatics [4], and future-proof ship pipe routing [5]. This problem is a generalization of the Steiner Tree Problem (STP) [6], in which one must find a minimum-cost subgraph spanning one set of vertices, referred to as *terminals*. Contrary to the STP, in the SSTP, the edge costs and terminals are affected by uncertainty and are revealed after the first stage. We assume there are finitely many possible outcomes for the second-stage parameters and that the probabilities of these scenarios are known.

CWI, Science Park 123, 1098 XG Amsterdam, The Netherlands e-mail: berend.markhorst,r.d.van.der.mei@cwi.nl · VU Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands e-mail: a.zocca,joost.berkhout@vu.nl

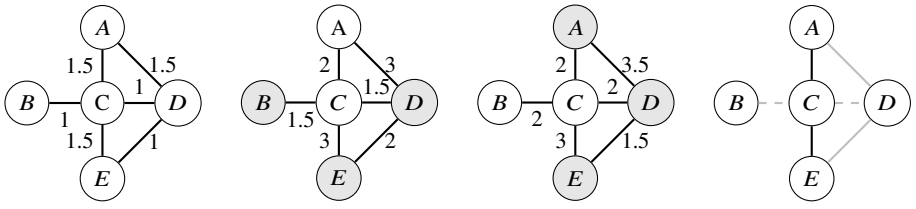
Approximation algorithms for the SSTP emerged in the 2000s [7, 8, 9, 10, 11, 12], while the 2010s saw a shift toward exact methods, spurred by the 2014 DIMACS Challenge [13]. Key SSTP developments in the SSTP literature include a genetic algorithm [3], branch-and-cut techniques [4, 1], ILP formulations [14], and a state-of-the-art decomposition model [2].

Due to the problem's NP-hardness, exact methods like the state-of-the-art algorithm from [2] scale poorly with instance size and the number of scenarios, making them unsuitable for real-time or interactive use. In contrast, a heuristic aims to quickly provide, high-quality solutions, enabling large-scale and adaptive applications in domains such as telecommunications or location planning [6]. It can also serve as a warm start to speed up exact methods, which can be done with (commercial) solvers, for example Gurobi [15].

We now formally introduce the SSTP, similar to [2, Definition 1]. We are given an undirected graph $G = (V, E)$ with the sets of vertices V and edges E, first-stage edge costs $c_e^{(0)} \in \mathbb{R}_+$ for each edge $e \in E$, and scenario set S. Each scenario $s \in S$ occurs with probability $p^{(s)} \in [0, 1]$, $\sum_{s \in S} p^{(s)} = 1$, and has (possibly different) second-stage edge costs $c_e^{(s)} \in \mathbb{R}_+$ (typically $> c_e^{(0)}$) and its own set of terminals $T^{(s)} \subseteq V$. We must choose a subset of edges in the first stage, $\bar{E}^{(0)} \subseteq E$ and second-stage, $\bar{E}^{(s)} \subseteq E$, for every scenario $s \in S$, whose union connects all terminals in $T^{(s)}$ and minimizes the following sum of the first stage costs and the expected second stage costs:

$$\sum_{e \in \bar{E}^{(0)}} c_e^{(0)} + \sum_{s \in S} p^{(s)} \sum_{e \in \bar{E}^{(s)}} c_e^{(s)}. \tag{1}$$

Figure 1 shows a small example of the SSTP with two equally-likely scenarios. In Figure 1a, we show the graph including the first-stage edge-costs, and in Figures 1b and 1c, we show the second-stage edge-costs and indicate the terminals with a gray shade for scenario $s = 1$ and $s = 2$. Figure 1d contains the optimal solution, where solid gray edges are chosen in the first stage, and dashed gray edges in the second stage, if necessary.



(a) First stage. (b) 2nd stage, $s = 1$. (c) 2nd stage, $s = 2$. (d) Optimal solution.

Fig. 1: Illustrative example of a Stochastic Steiner Tree Problem.

While few solution methods target the SSTP directly, many exist for its deterministic counterpart, the STP [6]. This work shows how an STP method can be adapted to solve the SSTP by decomposing it into subproblems, solving them independently, and combining the results.

The remainder of this work is structured as follows. In Section 2, we explain our method, after which we provide the setup of our experiments and present the corresponding results in Section 3. Finally, we make concluding remarks in Section 4.

2 Methodology

We present our heuristic and illustrate the intuition behind it with a small example in Sections 2.1 and 2.2, respectively.

2.1 Heuristic description

Our heuristic in Algorithm 1 works by decomposing the stochastic Steiner tree problem into smaller deterministic subproblems that can be solved quickly using an existing STP heuristic. It first partitions the scenario set into subsets and, for each subset, merges the terminals of all scenarios in that subset into a single deterministic STP. Solving this aggregated problem for first-stage costs produces a tree $\mathcal{E}^{(k)}$ that contains edges which are, loosely speaking, simultaneously cheap in the first stage and useful across many scenarios in that subset. The edges in $\mathcal{E}^{(k)}$ are candidate backbone edges that might be worth buying up front.

Next, the algorithm considers each scenario separately but restricts itself to the candidate edges identified for its subset. By solving a scenario-specific STP on this reduced graph, it records which candidate edges are actually used in each scenario. Based on the scenarios in which each edge is used, it compares the one-time first-stage cost to the expected second-stage cost of paying for that edge in each relevant scenario. Edges that are frequently used and relatively expensive in expectation in the second stage are purchased in the first stage; the remaining required edges are then completed per scenario in the second stage. This way, the algorithm exploits the fast deterministic STP heuristic while still capturing the main stochastic trade-off between upfront investment and recourse. In other words, the heuristic accounts for *the value of flexibility*, namely the advantage of a first-stage network that balances expected performance across scenarios and thus can be adapted at relatively low cost. By using multiple scenario subsets, it aims to find edges that are systematically useful across scenarios, guiding more robust first-stage decisions. There is, however, a limit to how much we should aggregate: combining all scenarios creates an artificial problem that encourages excessive upfront spending. Moderate subsets allow the heuristic to find a robust backbone that is neither shortsighted nor overly conservative. The appropriate subset size and configuration are problem-dependent and remain a topic for further research.

The STP heuristic we use has time complexity $O\left(|T^{(s)}||V|^2\right)$ for scenario s, see [16], and therefore, the total time complexity of our heuristic is $O\left(|S| \max_{s\in S} |T^{(s)}||V|^2\right)$. It is guaranteed that the STP solution of [16] is at most a factor $2\left(1-\frac{1}{l}\right)$ worse than the optimum, where l is the number of leaves in the optimal tree. Exploring how the

STP heuristic choice affects the SSTP solution quality is an interesting topic for future research.

2.2 Illustrative example of Algorithm 1

For illustration, we apply Algorithm 1 to the example from Section 1 and assume that both scenarios are equiprobable. Take subset size $h = 1$ so that we get $K = 2$ scenarios subsets. Suppose we get subsets $S_1 = \{1\}$ and $S_2 = \{2\}$. Applying [16], we get $\mathcal{E}^{(1)} = \{(B, C), (C, D), (D, E)\}$ and $\mathcal{E}^{(2)} = \{(A, D), (D, E)\}$ for both subsets, respectively. For this simple example, this leads directly to $\mathcal{F}^{(1)} = \mathcal{E}^{(1)}$ and $\mathcal{F}^{(2)} = \mathcal{E}^{(2)}$.

We now complete the algorithm for $e = (A, D)$ as an example. For $e = (A, D)$, it holds that $y_e^{(1)} = 0$ and $y_e^{(2)} = 1$ and since $c_e^{(0)} = 1.5$ is smaller than $\sum_{s \in S} c_e^{(s)} \cdot p^{(s)} \cdot y_e^{(s)} = 1.75$, we choose (A, D) in the first stage and thus add it to $\bar{E}^{(0)}$. Informally, this means that buying edge (A, D) in the first stage is cheaper than the expected cost of deferring the purchase, based on the edge's projected usage across the weighted scenarios. Completing the same steps for the remaining edges leads to the optimal solution as shown in Figure 1d, i.e., $\bar{E}^{(0)} = \{(A, D), (D, E)\}$, $\bar{E}^{(1)} = \{(B, C), (C, D)\}$, and $\bar{E}^{(2)} = \emptyset$.

Algorithm 1 SSTP heuristic.

Input: Number of scenarios per subset: h. Assume $|S|$ is a multiple of h for ease of presentation.
Output: Solution with first-stage edge set $\bar{E}^{(0)}$ and second-stage edge sets $\bar{E}^{(s)}$ for $s \in S$.

Set $y_e^{(s)} \leftarrow 0, \quad \forall s \in S, e \in E$. ▷ Equals 1 if edge e is used in scenario s, 0 else.
Set number of subsets $K \leftarrow \frac{|S|}{h}$.
Randomly partition scenario set S into K pairwise disjoint subsets $S_1, \ldots, S_K$ with $\bigcup_{k=1}^{K} S_k = S$.
for all subsets $k \in \{1, \ldots, K\}$ **do** ▷ For each partition, find an STP solution.
    Using [16], heuristically solve the STP with terminals $T = \cup_{s \in S_k} T^{(s)}$ based on the first-stage edge costs; collect the solution in $\mathcal{E}^{(k)}$.
end for
for all scenarios $s \in S$ **do** ▷ Asses which edges from $\mathcal{E}^{(k)}$ are used for scenario s.
    Set k to the (unique) subset index such that $s \in S_k$.
    Solve the STP on graph $\bar{G} = (V, \mathcal{E}^{(k)})$ connecting terminals $T^{(s)}$ based on the second-stage edge costs corresponding to scenario s; collect the solution in $\mathcal{F}^{(s)}$.
    for all edges $e \in \mathcal{F}^{(s)}$ **do**
        Update: $y_e^{(s)} \leftarrow 1$.
    end for
end for
for all edges $e \in E$ **do** ▷ Per edge, decide whether it is cheaper to select it in the first stage.
    if $c_e^{(0)} \leq \sum_{s \in S} c_e^{(s)} \cdot p^{(s)} \cdot y_e^{(s)}$ **then**
        Include e in the first stage solution $\bar{E}^{(0)}$.
    end if
end for
The code below is only necessary for obtaining the second-stage solution.
for all scenarios $s \in S$ **do** ▷ For each scenario, find a second-stage solution.
    Solve the STP corresponding to scenario s (including terminals and second-stage edge costs), while the edge costs from the edges $\bar{E}^{(0)}$ are set to zero. Store the solution in $\bar{E}^{(s)}$.
end for

3 Numerical Experiments

We test numerically the performance of our heuristic. Specifically, we discuss the experimental setup in Section 3.1 and the corresponding results in Section 3.2. Finally, we include important remarks in the discussion in Section 3.3.

3.1 Experimental setup

Similar to the state-of-the-art method [2], we test the performance of our algorithm on benchmark instances created for the 11th DIMACS Challenge on Steiner Tree problems, which are available on GitHub [17]. More specifically, we consider four datasets: K100, P100, LIN, and WRP, whose basic properties are presented in [2, Table 1]. The instances in these datasets range between five and a thousand scenarios.

In preliminary results, we compared five values for h with each other: $\{1, 2, 5, 10, 20\}$. This shows that $h = 2$ yields the best results, which is therefore used as the default in the remainder of this work. Due to space limitations, we will not elaborate on the sensitivity analysis on h. Yet, this can be a topic for further research. In this work, we consider h primarily as a tunable problem-dependent parameter in our heuristic.

In this work, we compare our method with the state-of-the-art exact algorithm from [2] and the *Buy-None* heuristic from [3]. The latter does not acquire any edges in the first stage and waits until the uncertainty is revealed, after which the second-stage solution is computed. We refer to this strategy as the "wait-and-see" approach and aim to find a balance between the speed of this method and the accuracy of [2].

We implemented both our algorithm and the wait-and-see approach in Python; our code and data are publicly available on GitHub [18]. The experiments are executed single-threaded on a high-performance computing (HPC) cluster with a clock speed of 2.4GHz and 2GB of memory per core. Note that the results from [2] are based on a C++ implementation and executed on a single-threaded HPC with a clock speed of 2.5GHz. To present a fair comparison with the state-of-the-art, we decided to report the original results from [2] without correcting them with a scaling factor.

3.2 Numerical results

We compare our method with [2] in terms of solution quality and runtime in Table 1. The former is quantified by the gap, defined by $\text{Gap} = \frac{UB_H - UB_L}{UB_L} \cdot 100\%$, where UB_L denotes the best upper bound found by [2] and provided on GitHub [17] and UB_H represents the objective found by our heuristic. In the third column of Table 1, we see how often our heuristic obtains a strictly lower gap than wait-and-see. These numbers differ considerably per dataset, which could be explained by the different properties of the underlying graphs. For example, we find a negative correlation between these numbers and the average degree of the graphs. Additionally, we see that the average optimality loss is approximately 5% for the datasets K100, LIN, and P100, and even around 3% for the WRP dataset. These solutions are found considerably faster than [2] and outperform the wait-and-see heuristic with only a small addition in runtime.

Table 1: Runtime and gap comparison between our heuristic (H), wait-and-see (W&S) and [2] per dataset.

Dataset	Avg. Degree	Best Gap (%)	Gap H (%)	Time H (sec)	Gap W&S (%)	Time W&S (sec)	Time [2] (sec)
K100	7.4	14.94	4.84	3.66	5.14	1.85	17.39
LIN	3.3	70.71	4.59	97.45	5.30	56.58	1497.35
P100	5.0	58.57	4.68	15.19	5.29	7.65	27.65
WRP	3.7	61.73	2.64	90.33	2.74	49.02	2249.34

3.3 Discussion

At the cost of longer execution times, though still significantly faster than the exact algorithm, the proposed heuristic achieves smaller gaps than the wait-and-see heuristic, especially on the LIN dataset. However, these results might heavily depend on the specific features of the instances from [2]. For example, the scenarios are almost equiprobable and the ratio between second- and first-stage costs is slightly bigger than one. The latter aids wait-and-see to perform well compared to our method, as selecting everything in the second stage is more affordable than in the first stage. We have enforced this claim with preliminary numerical experiments by increasing the second-stage costs. Based on the results from Table 1, we observe that our method quickly yields results that outperform wait-and-see. However, a key drawback of our method is that it still might overlook the value of flexibility. To illustrate this, we assume that $h = 1$, yet the same idea holds for $h > 1$. Figure 2 shows two equiprobable scenarios: one with terminals A and B, the other with A and D. Assuming second-stage edge costs are 50% higher, the optimal solution selects edge (A, C) in the first stage and connects C with either B or D in the second. While (A, C) is suboptimal in isolation, it is best when both scenarios are considered. Since our heuristic treats each scenario independently, it misses this flexibility and may underperform in such instances. In this case, it will select nothing in the first stage and either (A, B) or (A, D) in the second stage. The problem in this instance could be addressed by setting $h = 2$, yet it is not trivial to prevent in larger problems.

(a) Instance. (b) Optimal solution.

Fig. 2: Drawback of our method: inability to see the value of flexibility. In Figure 2b, solid and dotted lines represent the optimal first and second stage solution, respectively.

To conclude, the proposed SSTP heuristic is fast and outperforms the wait-and-see heuristic, but is prone to finding sub-optimal solutions due to overlooking the value of flexibility, depending on the choice of h and the specific problem instance being solved.

4 Conclusion

We have presented a powerful, yet simple and easy-to-implement method for the SSTP, whose performance is analyzed with runs on benchmark instances from the literature. Using an STP heuristic, we solve the SSTP with fast computation times and a gap of approximately 5%. Extending this method to the Stochastic Steiner Forest Problem (SSFP) and other stochastic network design problems, e.g., [1], has considerable potential to boost research in network design under uncertainty. It is also worth investigating the heuristic's sensitivity to the chosen subset size and the specific STP heuristic used. Another topic for future research is combining our method with scenario reduction, as this could reduce the runtime even more. Finally, adding the Reduced Cost heuristic from [3] to our heuristic, possibly in an iterative fashion, could enhance the performance.

Acknowledgements The authors thank Ruurd Buijs and Joris Slootweg for the insightful discussions about this study.

Competing Interests This publication is partly financed by the Dutch Research Council (NWO) with project number *TWM.BL.019.002*. The authors have no competing interests to declare that are relevant to the content of this work.

References

[1] Ivana Ljubić, Petra Mutzel, and Bernd Zey. Stochastic survivable network design problems: Theory and practice. *European Journal of Operational Research*, 256(2):333–348, January 2017.

[2] Markus Leitner, Ivana Ljubić, Martin Luipersbeck, and Markus Sinnl. Decomposition methods for the two-stage stochastic Steiner tree problem. *Computational Optimization and Applications*, 69(3):713–752, April 2018.

[3] Pedro Hokama, Mario C San Felice, Evandro C Bracht, and Fabio L Usberti. A Heuristic Approach for the Stochastic Steiner Tree Problem. In *Preprint*, 2014.

[4] Immanuel Bomze, Markus Chimani, Michael Jünger, Ivana Ljubić, Petra Mutzel, and Bernd Zey. Solving Two-Stage Stochastic Steiner Tree Problems by Two-Stage Branch-and-Cut. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation*, volume 6506, pages 427–439. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes in Computer Science.

[5] Berend Markhorst, Joost Berkhout, Alessandro Zocca, Jeroen Pruyn, and Rob Van Der Mei. Future-proof ship pipe routing: Navigating the energy transition. *Ocean Engineering*, 319:120113, March 2025.

[6] Ivana Ljubić. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, March 2021.

[7] Nicole Immorilica, David Karger, Maria Minkoff, and Vahab S. Mirrokni. On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. In *Fifteenth Annual ACM-SIAM Symposium*

on Discrete Algorithms, pages 691–700, New Orleans, Louisiana, November 2004. Society for Industrial and Applied Mathematics.

[8] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: approximation algorithms for stochastic optimization. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 417–426, Chicago IL USA, June 2004. ACM.

[9] Anupam Gupta and Martin Pál. Stochastic Steiner Trees Without a Root. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580, pages 1051–1063. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. Series Title: Lecture Notes in Computer Science.

[10] Chaitanya Swamy and David B. Shmoys. Approximation algorithms for 2-stage stochastic optimization problems. *ACM SIGACT News*, 37(1):33–46, March 2006.

[11] Anupam Gupta, MohammadTaghi Hajiaghayi, and Amit Kumar. Stochastic Steiner Tree with Non-uniform Inflation. In Moses Charikar, Klaus Jansen, Omer Reingold, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 4627, pages 134–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[12] Anupam Gupta, R. Ravi, and Amitabh Sinha. LP Rounding Approximation Algorithms for Stochastic Network Design. *Mathematics of Operations Research*, 32(2):345–364, May 2007.

[13] 11th DIMACS Implementation Challenge: Steiner tree problems. `http://dimacs11.zib.de/`. Accessed: 2025-07-23.

[14] Bernd Zey. ILP formulations for the two-stage stochastic Steiner tree problem, November 2016. arXiv:1611.04324 [cs].

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2025.

[16] Lawrence Kou, George Markowsky, and Leonard Berman. A fast algorithm for Steiner trees. *Acta informatica*, 15:141–145, 1981.

[17] Markus Sinnl. GitHub repository: Stochastic Steiner Tree Problem. `https://msinnl.github.io/pages/sstp.html`, 2025. Accessed: 2025-05-13.

[18] Berend Markhorst. GitHub repository: SteinerForest, Python code to solve the deterministic and stochastic Steiner forest problem with Gurobi. `https://github.com/berendmarkhorst/SteinerForest`, 2025. Accessed: 2025-05-09.