A greedy algorithm for finding maximum spanning trees in infinite graphs

Christopher Thomas Ryan[a], Robert L Smith[b]

[a] *UBC Sauder School of Business, 2053 Main Mall, Vancouver, British Columbia, Canada V6M 3W2,*
[b] *University of Michigan, Industrial and Operations Engineering, 1205 Beal Ave., Ann Arbor, Michigan, United States of America 48109-2117,*

Abstract

In finite graphs, greedy algorithms are used to find minimum spanning trees (MinST) and maximum spanning trees (MaxST). In infinite graphs, we illustrate a general class of problems where a greedy approach discovers a MaxST while a MinST may be unreachable. Our algorithm is a natural extension of Prim's to infinite graphs with summable and strictly positive edge weights, producing a sequence of finite trees that converge to a MaxST.

Keywords: spanning trees, infinite graphs, infinite-dimensional optimization

1. Introduction

The minimum weight spanning tree (MinST) problem on finite graphs is a classic combinatorial optimization problem. The MinST problem has numerous applications [15, 17, 18, 22], including as a subroutine in other algorithms on graphs [3] and heuristics [12, 26]. MinST problems are also popular because they admit simple-to-implement "greedy algorithms" that solve the problem efficiently. In this paper, we consider the *maximum* spanning tree (MaxST) problem for countably infinite graphs which can be viewed as models for underlying problems with indefinitely large graphs. We will present a greedy algorithm that arbitrarily well approximates a MaxST which will be shown to always exist. As we shall see, both of these claims fail for the MinST case for infinite graphs.

To an uninitiated researcher, one may be naturally presented with a *maximum* weight spanning tree (MaxST) problem instead of a *minimum* and search the literature in vain to find solutions to this problem. The reason is not that the MaxST problem is difficult or unstudied, instead, it is because it is easily converted to a MinST by reversing the signs of the edge weights and minimizing. Unlike the difference between the minimum capacity cut (MinCut) problem and the maximum capacity cut (MaxCut) problem (the minimum cut problem is well known to be the dual of the maximum flow problem, which is polynomially solvable in finite graphs, whereas the maximum weight problem is NP-hard to both solve and approximate), the differences between the MinST problem and the MaxST problem on finite graphs are entirely cosmetic.

In this paper, we show that the situation can be completely different in infinite graphs. We consider a class of graphs with countably many nodes, and each node has at most finitely many incident edges. Moreover, we assume the edge weights are positive and the sum of weights is finite. In this setting, a greedy approach can be used to find a MaxST but a MinST may not even exist. If a MinST does exist, it may be unreachable by a greedy approach. Since edge weights are summable, they converge to zero towards the "outskirts" of the graph. Intuitively, a MinST may be unreachable by a greedy approach as we search in these "outskirts" for lighter and lighter edges and get "indefinitely distracted" without returning to explore other regions of the graph to form a spanning tree. By contrast, a MaxST can safely ignore the "outskirts" of light-weight edges until trees are greedily constructed on a growing family of finite subgraphs leading, eventually, to a MaxST.

More concretely, our greedy approach for computing a MaxST is a straightforward extension of Prim's algorithm

applied to infinite graphs. When the graph has summable and strictly positive edge weights, it produces a sequence of spanning trees on finite subgraphs that converge to a MaxST in the limit (we make the notion of convergence precise later in the paper). Such solution convergence is a rare outcome in infinite-dimensional optimization problems where convergence to the optimal solution is usually difficult to guarantee.

To our knowledge, there are only two previous papers—[23] and [20]—that study the problem of constructing MinSTs and MaxSTs in infinite graphs (in fact, they study the problem in the more general setting of infinite matroids.) These papers, however, generalize Kruskal's algorithm to the infinite setting, whereas we extend Prim's algorithm. While their approach can also find MaxSTs in our setting (we describe how their ideas can be applied to our setting in Section 4), our approach has two advantages over Kruskal's approach.

These two advantages are (i) a generalized Prim's approach requires significantly less data about the graph at each iteration, and (ii) the iterates of Prim's algorithm are trees and therefore constitute optimal solutions of MaxSTs on finite subgraphs of the original graph. By contrast, the iterates of a generalized Kruskal's approach are typically forests and not trees. This distinction can be important in applications of spanning tree problems—like establishing communication links along edges from a source to nodes—where disconnected partial solutions lead to links that are not connected to the source. These two advantages are explored in more detail in Section 4.

It is also worth noting that there is extensive literature on algorithms on infinite graphs in other contexts (see, for instance, [2, 5, 13, 21]). Several references examine the properties of spanning trees in the limit of finite random graphs (see, for instance, [1, 4]), but these graphs enjoy special properties conferred by Poisson spatial processes.

The rest of the paper is organized as follows. In Section 2, we formally introduce the problem of finding minimum and maximum weight spanning trees and state our key assumptions. In Section 3, we discuss greedy algorithms for solving the minimum-weight and maximum-weight spanning tree problems, emphasizing the differences between the two. There we show our main result: a Prim's algorithm always finds a MaxST in the limit.

2. The minimum and maximum spanning tree problems

We begin by introducing the general class of infinite graphs we consider.

2.1. Basic definitions

Let $G = (\mathcal{V}, \mathcal{E})$ denote an undirected graph, with node set $\mathcal{V} = \{1, 2, \dots\}$ and edge set $\mathcal{E}$, which is a subset of all possible unordered pairs $\{i, j\}$, where $i, j \in \mathcal{V}$ with $i \neq j$. The graph has an edge-weight function $w : \mathcal{E} \to \mathbb{R}$ where w_e denotes the weight of edge $e \in \mathcal{E}$.

Let $I(i)$ denote the set of nodes adjacent to node i; that is, $I(i) := \{j \in \mathcal{V} : \{i, j\} \in \mathcal{E}\}$. The *degree* of node i in G is the cardinality of $I(i)$. A graph is *locally finite* if every node has finite degree. A *path* in G is a finite sequence of distinct nodes $i_1, i_2, \dots, i_n$, where $\{i_k, i_{k+1}\} \in \mathcal{E}$ for $k = 1, \dots, n-1$. A *ray* is an infinite sequence of distinct nodes $i_1, i_2, \dots$, where $\{i_k, i_{k+1}\} \in \mathcal{E}$ for $k = 1, 2, \dots$. Two nodes i and j are *connected* in G if there exists a path starting with node i and ending with node j. The graph G is *connected* if all nodes i and j in G are connected. We make the following assumption throughout the paper.

Assumption 1. The graph G is locally finite and connected. ◁

The results in this paper hold if local finiteness is relaxed, but for ease of exposition we stick with this assumption. Indeed, much of infinite graph theory is examined in the locally finite case, which is easier to visualize (see, for instance, Chapter 8 in [13]).

A *cycle* in G is a finite sequence of nodes $i_1, i_2, \dots, i_n, i_1$, where $i_1, i_2, \dots, i_n$ is a path and $\{i_1, i_n\} \in \mathcal{E}$. A *double ray* consists of a node i with two distinct rays, that is, rays $(i, i_1, i_2 \dots)$ and $(i, j_1, j_2, \dots)$, where all intermediate nodes i_k and j_ℓ are distinct for all k and ℓ.

Let H be a subgraph of G and let $\mathcal{V}(H)$ and $\mathcal{E}(H)$ denote the set of nodes and edges in H, respectively. In this paper, we restrict our attention to subgraphs with no isolated nodes; that is, for every node $i \in \mathcal{V}(H)$ there exists an edge $\{i, j\} \in E(H)$ for some node $j \in \mathcal{V}(H)$.Throughout we often refer to a subgraph H by its set $\mathcal{E}(H)$ of edges since the set of nodes is implicit once the edges are defined due to this restriction on the types of subgraphs we consider.

A *forest* F of G is an acyclic subgraph of G; i.e., a subgraph of G without cycles. A connected forest is a *tree*. If a subgraph of G has node set $\mathcal{V}$, it is said to *span* G. A connected spanning forest of G is called a *spanning tree*. The set of all spanning trees of the graph G is denoted $\mathcal{T}$.

Remark 1. Other papers that study infinite graphs may define trees differently. For instance, the papers [24, 25] study network flow problems on directed graphs. They say two nodes i and j are *connected at infinity* if both lie on directed rays to infinity, even if there is no finite path between these nodes. These papers talk about trees as graphs that do not contain either cycles or double rays, but allow connectivity "at infinity" between nodes. By contrast, our trees allow double rays (while disallowing cycles) but must be (finitely) connected. For a detailed discussion of the different definitions of spanning trees in infinite graphs, see [14]. ◁

2.2. On the existence of spanning trees

We now turn to our problems of interest. The *weight* $w(T)$ of a spanning tree T of G is the sum of the weights of the edges of T, i.e.,

$$w(T) \triangleq \sum_{e \in \mathcal{E}(T)} w_e. \tag{1}$$

We first define the more commonly stated problem of finding a minimum-weight spanning tree of G, i.e., solve

$$w_{\min} \triangleq \min\{w(T) \mid T \in \mathcal{T}\}. \tag{MinST}$$

We call any optimal solution of problem (MinST) a minimum spanning tree (MinST). A closely related problem is solving the maximum-weight spanning tree problem, i.e., solve

$$w_{\max} \triangleq \max\{w(T) \mid T \in \mathcal{T}\}. \tag{MaxST}$$

We call any optimal solution of problem (MaxST) a maximum spanning tree (MaxST).

These problems may not be well-defined if either the graph has no spanning trees or spanning trees of minimum or maximum weight do not exist. Regarding the existence of spanning trees, the following classical result shows they always exist in our setting.

Proposition 1 (Proposition 8.1.1 in [13])**.** Any graph that is locally finite and connected (Assumption 1) contains a spanning tree.

Although spanning trees exist, there may be no lower bound on their weight. It is straightforward to construct an infinite graph with an infinite sequence of spanning trees with strictly decreasing weights when the graph admits edges with negative cost. Another worrying setting is one where all spanning trees have infinite weight, which happens when the weights of edges are not "controlled" in some way. In this case, all spanning trees are degenerately both "minimal" and "maximal", making the MinST and MaxST problems uninteresting. We make the following assumption throughout to avoid these exceptions.

Assumption 2. The weight function $w : \mathcal{E} \to \mathbb{R}$ has $w_e > 0$ for all $e \in \mathcal{E}$ and $\sum_{e \in \mathcal{E}} w_e < \infty$. ◁

If we label the countably many edges in $\mathcal{E}$ by w_ℓ for $\ell = 1, 2, \dots$, then Assumption 2 becomes $w = (w_1, w_2, \dots) \in \ell_1$ (where ℓ_1 is the vector space of absolutely summable sequences) with $w > 0$.

Later, we show that this assumption (combined with Assumption 1) suffices to establish the existence of a MaxST. However, the following example illustrates that a MinST may not exist.

Example 1. Consider the infinite ladder graph in Figure 1, with top and bottom rays of decreasing weight edges connected by infinitely many rungs with decreasing weights. The MaxST has weight 3, consisting of the leftmost rung of weight 1 connecting the top and bottom rays of the ladder (each of which has weight 1). A spanning tree of weight $2\,1/4$ is drawn in non-dashed edges in the figure. One can similarly construct spanning trees of weight $2\,1/8$, $2\,1/16$, etc. Thus, there is a sequence of spanning trees whose weights converge to 2. However, no spanning tree has weight 2 or less. Therefore, a MinST does not exist. ◁

3. Greedy algorithms

We just provided an example where a MinST may not exist in a graph that satisfies Assumptions 1 and 2. In Section 3.1, we show that even when a MinST does exist, it may not be discoverable by an infinite extension of Prim's

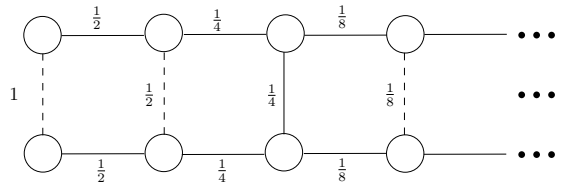

Figure 1: A graph with a MaxST but no MinST (see Example 1).

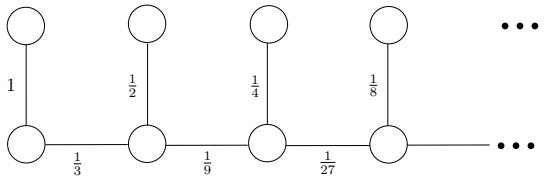

Figure 2: A graph satisfying Assumptions 1 and 2 where a MinST exists that cannot be found by Prim's algorithm (see Example 2).

algorithm. Later in Section 3.2, we show that a MaxST always exists and can be found using Prim's algorithm.

3.1. Greedy algorithms for minimum spanning trees

There are multiple greedy approaches to finding MinSTs in finite graphs, including algorithms attributed to Prim, Kruskal, and Sollin (see, for instance, Chapter 13 of [3]). In the infinite case, these algorithms perform differently. For instance, Kruskal's algorithm seeks a minimum-weight edge in the entire graph in the first iteration. Not only does identifying a minimum-weight edge in an infinite graph require infinite work, but the operation may not even be well defined since the minimum of edge weights need not exist. By contrast, each iteration of the natural extension of Prim's algorithm to infinite graphs is finitely implementable since it always considers finitely-many edges in each iteration, as we detail now.

Algorithm 1 Prim's algorithm for MaxST (resp. MinST) in spanning trees in infinite graphs

1: **Input:** A locally finite and connected graph $G = (\mathcal{V}, \mathcal{E})$ with edge weights.
2: **Initialize:** Initialize a tree T with one node, chosen arbitrarily from G.
3: **while** T is not spanning **do**
4: **Append an edge:** Append to T the maximum-weight (resp minimum-weight) edge emanating from T (that is, having one node in T and one outside of T).

Prim's algorithm (stated in Algorithm 1) produces a sequence of non-spanning finite trees $\{T^n\}$, one for each pass of the while loop, where T^1 is the initial single-node tree, T^2 is the tree at the end of the first iteration of the while loop, etc. The algorithm is finitely implementable since each iterate has finitely many nodes, and so because each node has finite degree, only finitely many edges need to be considered when finding the maximum or minimum weight edge.

The challenge with Prim's algorithm is that the limiting tree it creates may not be spanning.

Example 2. Consider the graph in Figure 2. The graph itself is a spanning tree with total weight $(1 + 1/2 + 1/4 + \cdots) + (1/3 + 1/9 + \cdots) = 2\,1/2$. Hence, the minimum-weight spanning tree is unique with weight $2\,1/2$. However, if Prim's algorithm is started with, for example, the lower-left node, it fails to span all nodes in the graph. Indeed, at every iteration, it adds the next horizontal edge available, and never adds any of the vertical edges. ◁

3.2. A greedy algorithm for maximum spanning trees

We now show that Prim's algorithm *always* finds a MaxST in any graph satisfying Assumptions 1 and 2.

Theorem 1. Let G be an infinite graph that is locally finite and connected (Assumption 1) and has positive and summable weights (Assumption 2). Let T^n be the nth tree generated by Prim's algorithm for a maximum weight spanning tree and let

$$T^* = \bigcup_{n=1}^{\infty} T^n, \tag{2}$$

where the operator $\cup$ merges nodes and edges. The subgraph T^* is a MaxST. In particular, the graph G has a MaxST.

Proof. First, we claim that T^* is a forest. Observe that T^* has no finite cycles since any finite cycle of T^* would eventually be contained in T^n while T^n being a tree is acyclic. This implies that T^* is a forest.

Second, it is straightforward to see that T^* is connected. Let i and j be two arbitrary nodes in T^*. Let n_i be the smallest value of n such that i is in T^n. Define n_j similarly. Without loss, assume $n_i < n_j$. At iterations n_j, there is a node k in tree T^{n_j-1} so that $\{k, j\} \in T^{n_j}$. If $k = i$ then we are done. Otherwise, since the iterates of Prim's algorithm are a growing sequence of trees, i is a node in T^{n_j-1}. Moreover, since T^{n_j-1} is connected, there is a path P from i to k in T^{n-1}. Then $P \cup \{k, j\}$ is a path connecting i and j.

Third, we observe that T^* is a spanning tree by arguing that every node i in G is in T^n for some n. Suppose not for node i; i.e., i is not in T^n for any n. Let P_i be a finite path in G from node 1 to node i. Let $i^*(i)$ be the last node along this path that lies in T^*. The weight of the edge out of node $i^*(i)$ along the path to i is strictly greater than 0 by assumption and was eligible to be added to T^n for sufficiently large n but was not. However, the weights of edges added to T^n decrease to 0. This contradicts the requirement to add the largest weight edge to T^n in step 4 of Prim's algorithm.

Fourth, we show that T^* has maximum weight. Let S be any spanning tree of G. Let G^n be the subgraph of G spanned by the nodes of T^n. Note that T^n is a MaxST for G^n since Prim's algorithm produces a MaxST on the finite graph G^n. This follows from the classical properties of Prim's algorithm on finite graphs. Let F^n be the forest of edges consisting of the edges of S that are contained in G^n. Since edges can always be added from G^n to extend F^n to a spanning tree S^n of G^n, we have

$$w(F^n) \le w(S^n) \le w(T^n)$$

for all n, where $w(H)$ is the total weight associated with the edges of subgraph H as defined in (1). Since every node in G is eventually in T^n, we have $G = \cup_{n=1}^{\infty} G^n$ and $S = \cup_{n=1}^{\infty} F^n$. Then, since edge weights associated with G are summable, we have

$$w(S) = \lim_{n\to\infty} w(F^n) \le \lim_{n\to\infty} w(T^n) = w(T^*). \tag{3}$$

Since S was an arbitrary spanning tree of G, we conclude that T^* is a MaxST of G. □

It is immediate from (3) that we have optimal value convergence of the iterates of Prim's algorithm, namely that $w(T^n) \to w(T^*)$ as $n \to \infty$ and in fact $w(T^n)$ monotonically converges to $w(T^*)$. The sequence of trees T^n converges to the tree T^* in the following sense: a sequence of subgraphs S^k of a graph G *converges* to a subgraph S in G if there is a positive integer K_e for each edge $e \in \mathcal{E}$ so that for all $k \ge K_e$ we have $e \in S^k$ if $e \in S$ while $e \notin S^k$ if $e \notin S$. Indeed, for every edge $e \in T^*$, K_e is the minimum value of n such that $e \in T^n$. That is, an edge e enters T^n for some n only when e lies in T^* and this edge stays in all remaining iterates. For additional discussion of topologies on infinite graphs see [16].

We can say more about this value K_e. Without loss of optimality, let node 1 be the initial node of G for Prim's Algorithm. Let T^* be the MaxST delivered in the limit by Prim's Algorithm (as defined in (2) above). For any edge $e \in T^*$, let $p^*(e)$ be the unique path in T^* from node 1 up to (and including) edge e. Let $\underline{w}^*(e) = \min_{e' \in p^*(e)} w_{e'} > 0$ be the minimum weight of an edge in the path $p^*(e)$. For every positive real number γ, let $\mathcal{E}(\gamma)$ be the set of edges in G with weights greater than or equal γ. By Assumption 2 the set $\mathcal{E}(\gamma)$ is finite. Let $N(\gamma)$ denote the (finite) cardinality of $\mathcal{E}(\gamma)$.

Lemma 1. Let $e \in T^*$. Then $e \in T^n$ for all $n \ge N(\underline{w}^*(e))$. That is, $K_e \le N(\underline{w}^*(e))$.

Proof. Consider the set of edges $\mathcal{E}(\underline{w}^*(e))$. Since every edge along the MaxST path $p^*(e)$ to edge $e \in T^*$ has weight at least $\underline{w}^*(e)$ and one of these edges in $p^*(e)$ is a candidate for adding in every step n of Prim's Algorithm before all of $p^*(e)$ has been formed in T^n, Prim's algorithm only adds edges from $\mathcal{E}(\underline{w}^*(e))$ while $p^*(e)$ is being formed. Since there are at most $N(\underline{w}^*(e))$ edges in $\mathcal{E}(\underline{w}^*(e))$, the path $p^*(e)$ (and, in particular, the edge e) must be added in the first $N(\underline{w}^*(e))$ steps of Prim's algorithm. That is, $e \in T^{N(\underline{w}^*(e))}$. Since edges are only added to the iterates T^n (and never removed) in the execution of Prim's algorithm, we conclude that $e \in T^n$ for all $n \ge N(\underline{w}^*(e))$. □

The following example helps illuminate the result in Lemma 1 in a context where the weights of the edges of the graph decay geometrically.

Example 3. Suppose G is a locally finite and connected graph with edges labeled $\ell = 1, 2, \ldots$. Suppose, in addition, that the edge weights are disciplined by a discounted upper bound as follows: for some $b > 0$ and $0 < \delta < 1$ we have

$$w_\ell < b\delta^\ell \tag{4}$$

for all $\ell = 1, 2, \ldots$. Clearly, this graph satisfies Assumption 2 and so by Theorem 1, a MaxST T^* (given by (2)) exists and can be found by Prim's algorithm. We now show that we can determine whether a given edge e is in T^* by only considering a finite subgraph whose size is a function of $\underline{w}^*(e)$, b, and δ.

Let γ be a positive real number and let $L(\gamma)$ be the L satisfying $b\delta^L = \gamma$. Note that $w_\ell \leq b\delta^\ell \leq b\delta^{L(\gamma)} = \gamma$ for all $\ell \geq L(\gamma)$, where the first inequality follows from (4). Observe that $\mathcal{E}(\gamma) \subseteq \{1, 2, \ldots, L(\gamma)\}$ since only those edges ℓ with $\ell \leq L(\gamma)$ are candidates to be in $\mathcal{E}(\gamma)$. Recall that $\mathcal{E}(\gamma)$ is the set of edges of G with weights greater than or equal γ. Hence, we can conclude that $N(\gamma) \leq L(\gamma)$.

Note that we have $\delta^{L(\gamma)} = \gamma/b$ or $L(\gamma) = \log_\delta(\gamma/b)$. By Lemma 1, we know that if $e \in T^*$ then $e \in T^n$ for $n \geq N(\underline{w}^*(e))$ where $N(\underline{w}^*(e)) \leq L(\underline{w}^*(e)) = \log_\delta(\underline{w}^*(e)/b)$. That is, $e \in T^n$ for $n \geq \log_\delta(\underline{w}^*(e)/b)$.

The quantity $\log_\delta(\underline{w}^*(e)/b)$ is computable in finite time for every edge e as a function of $w^*(e)$. ◁

Note that edge $e \in T^n$ at step n for every n is based on the topology and weights associated with the finitely many edges emanating from the tree T^n. This data, together with that contained in T^n itself, serves as a type of *forecast horizon* prominent in planning horizon research [6, 11, 7, 8, 9]. That is, edge e is guaranteed to be a part of a MaxST independently of any unforecasted data beyond the tree T^n and the edges emanating from it. Thus, for example, MaxST can be implemented sequentially as we forecast demand and supply data without the necessity of reworking previous builds of trees.

4. Comparison with existing work

In the introduction, we briefly mentioned two closely related papers to our work: [23] and [20]. These papers develop algorithms to find the equivalent of MinSTs and MaxSTs in infinite matroids. These algorithms find these objects only under certain conditions, as specified in these papers. For brevity, we will not introduce the matroid framework in this paper to make these conditions precise. However, in order to compare their results to ours, we will briefly describe their results in the language of infinite graphs used in this paper.

[23] and [20] develop a generalized version of Kruskal's algorithm for finding spanning trees. The idea of Kruskal's algorithm is roughly as follows: find the maximum (resp. minimum) weight edge in the graph, and add it to a growing forest, ensuring that no cycles are introduced as we proceed. For the algorithm to be finitely implementable, the task of finding edges with maximum (resp. minimum) weight in the graph needs to be achievable in finite time. In the case of finding MinSTs, it's possible that no such minimum weight edge exists.

For the MaxST problem under our assumptions, a maximum weight edge can always be found. Let's briefly explore why this is. Let ℓ be the label of an arbitrary edge not added yet to the growing forest (using the labeling of edges defined in the paragraph after Assumption 2). Since the edge weights are positive and summable, there exists an edge labeled ℓ' such that $w_m \leq w_\ell$ for all $m \geq \ell'$. Then, the maximum weight edge not in F can be chosen among the edges in the finite set $\{1, \ldots, \ell-1, \ell, \ell+1, \ldots, \ell'-1, \ell'\}$. In other words, a maximum weight edge can be found in finite time.

Let's now compare this with what happens in our generalization of Prim's algorithm. First, the information required to compute iterates is much less in Prim's algorithm than in Kruskal's. Observe that line 4 of Prim's algorithm can be executed by only searching the finitely many edges emanating from the tree iterate T. Given knowledge T, the number of edges that need to be considered is known—at most the sum of degrees of the nodes in T. By contrast, although finding an edge to add in Kruskal's algorithm can be achieved in finite time, without further assumptions on the weights (as we did in Example 3) we cannot give a clear bound on how far into the graph we need to explore. Thus, while both Prim and Kruskal are greedy, Prim is greedy in a *myopic* way by finding heavy edges *early* in the graph.

This raises a question of how input is provided to the two algorithms. Since the graph is infinite, it is not possible to input all of the data in the graph in finite time. A finite subset of the data must be "streamed" to the algorithm as it proceeds. This raises the consideration of a streaming model of computation [19, 10]. We will not go into the fine details of streaming complexity here, except to say that the informational differences between Prim's and Kruskal's algorithms imply that Prim's can work when a much smaller amount of finite data is streamed to the algorithm (the data on edges incident with the current tree iterate). Alternatively, Kruskal's algorithm requires fore-

casting data without an *a priori* upper bound for recursive determination of maximal weight edges.

This brings us to a separate point. Whereas the iterate forests of Kruskal's algorithm can be disconnected, the iterates of Prim's algorithm are always trees. Thus, if our infinite extension of Prim's algorithm is finitely terminated, the result is a spanning tree on the finite graph so far explored by the algorithm. This serves as a partial optimal solution that can be implemented as it is constructed. This distinction may be important in rolling horizon applications of spanning tree problems—like establishing communication links along edges from a source to nodes—where disconnected partial solutions recommend making links that are not connected to the source.

Acknowledgements

We thank the associate editor and two reviewers for their help on improving this paper. In particular, one reviewer alerted us to the work of [23] on Kruskal's algorithm for infinite graphs. This helped us refine the positioning of the paper. Christopher Thomas Ryan is supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant RGPIN-2020-06488 and the UBC Sauder Exploratory Grants Program.

References

[1] Louigi Addario-Berry, Nicolas Broutin, Christina Goldschmidt, and Grégory Miermont. The scaling limit of the minimum spanning tree of the complete graph. *The Annals of Probability*, 45(5):3075–3144, 2017.

[2] R. Aharoni, E. Berger, A. Georgakopoulos, A. Perlstein, and P. Sprüssel. The max-flow min-cut theorem for countable networks. *Journal of Combinatorial Theory Series B*, 101(1):1–17, 2011.

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] David Aldous and J Michael Steele. Asymptotics for Euclidean minimal spanning trees on random points. *Probability Theory and Related Fields*, 92(2):247–258, 1992.

[5] Edward J Anderson and Andrew B Philpott. A continuous-time network simplex algorithm. *Networks*, 19(4):395–425, 1989.

[6] James C. Bean and Robert L. Smith. Conditions for the existence of planning horizons. *Mathematics of Operations Research*, 9(3):391–401, 1984.

[7] James C Bean and Robert L Smith. Conditions for the discovery of solution horizons. *Mathematical Programming*, 59(1):215–229, 1993.

[8] Christian Bès and Suresh P. Sethi. Concepts of forecast and decision horizons: Applications to dynamic stochastic optimization problems. *Mathematics of Operations Research*, 13(2):295–310, 1988.

[9] Suresh Chand, Vernon Ning Hsu, and Suresh Sethi. Forecast, solution, and rolling horizons in operations management problems: A classified bibliography. *Manufacturing & Service Operations Management*, 4(1):25–43, 2002.

[10] Yi-Jun Chang, Martín Farach-Colton, Tsan-Sheng Hsu, and Meng-Tsung Tsai. Streaming complexity of spanning tree computation. *arXiv preprint arXiv:2001.07672*, 2020.

[11] Torpong Cheevaprawatdomrong, Irwin E Schochetman, Robert L Smith, and Alfredo Garcia. Solution and forecast horizons for infinite-horizon nonhomogeneous markov decision processes. *Mathematics of Operations Research*, 32(1):51–72, 2007.

[12] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University Technical Report, 1976.

[13] R. Diestel. *Graph Theory*. Springer, 4th edition, 2010.

[14] Reinhard Diestel and Daniela Kühn. Topological paths, cycles and spanning trees in infinite graphs. *European Journal of Combinatorics*, 25(6):835–862, 2004.

[15] Maman Abdurachman Djauhari and Siew Lee Gan. Optimality problem of network topology in stock

market analysis. *Physica A: Statistical Mechanics and Its Applications*, 419:108–114, 2015.

[16] Agelos Georgakopoulos. Graph topologies induced by edge lengths. *Discrete mathematics*, 311(15):1523–1542, 2011.

[17] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[18] Daniel Granot and Gur Huberman. Minimum cost spanning tree games. *Mathematical Programming*, 21(1):1–18, 1981.

[19] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

[20] Victor Klee. The greedy algorithm for finitary and cofinitary matroids. In Theodore S. Motzkin, editor, *Combinatorics: Proceedings of Symposia in Pure Mathematics*, pages 137–152. 1971.

[21] Sevnaz Nourollahi and Archis Ghate. Duality in convex minimum cost flow problems on infinite networks and hypernetworks. *Networks*, 70(2):98–115, 2017.

[22] Alice Paul, Daniel Freund, Aaron Ferber, David B Shmoys, and David P Williamson. Budgeted prize-collecting traveling salesman and minimum spanning tree problems. *Mathematics of Operations Research (articles in advance)*, 2019.

[23] Richard Rado. Note on independence functions. *Proceedings of the London Mathematical Society*, 3(1):300–320, 1957.

[24] Christopher Thomas Ryan, Robert L. Smith, and Marina A. Epelman. A simplex method for uncapacitated pure-supply infinite network flow problems. *SIAM Journal on Optimization*, 28(3):2022–2048, 2018.

[25] Thomas C. Sharkey and H. Edwin Romeijn. A simplex algorithm for minimum-cost network-flow problems in infinite networks. *Networks*, 52(1):14–31, 2008.

[26] Kenneth J Supowit, David A Plaisted, and Edward M Reingold. Heuristics for weighted perfect matching. In *ACM STOC Symposium on Theory of Computing*, pages 398–419, 1980.