Partial Low Stretch Trees

Géraud Le Falher

August 8, 2015

Problem

Let $G = (V, E, w)$ be an undirected weighted graph, $X \subset V$ a set of observed nodes, $\overline{X}$ its complement and $\mathcal{T}_G$ the set of spanning trees of G. We are looking for

$$T^* = \underset{T \in \mathcal{T}_G}{\operatorname{argmin}} f(T)$$
$$\text{where} \quad f(T) = \sum_{i \in X,\, j \in \overline{X}} |path_T(i, j)| \tag{1}$$

Note that by symmetry of (1), we may assume that $|X| \leq \left|\overline{X}\right|$. A variant of f is g, which measures how closely a given tree T projects $\overline{X}$ onto X:

$$g(T) = \sum_{j \in \overline{X}} \min_{i \in X} |path_T(i, j)| \tag{2}$$

Contrary to f, g is not symmetric in X and $\overline{X}$: consider for instance the case where X is a singleton.

Methods

Let's first consider the unweighted case: $w = \mathbf{1}$. Here are three ideas on how to solve the problem:

Merged k-BFTs The first one is quite natural albeit rather unpractical: for each $i \in X$, we build an arbitrary Breadth First Tree (BFT) rooted at i (let's call it T_i. There are several BFTs rooted at i hence we could build k of them. In the following experiments though, we set $k = 1$). This yields the shortest paths from i to all $j \in \overline{X}$, and we denote the sum of the lengths of these $|\overline{X}|$ paths as S_i. Therefore a (non tight) lower bound of $f(T^*)$ is $\sum_{i \in X} S_i$. The last step is then to merge these $|X|$ BFTs into a single spanning tree. Unfortunately I couldn't think of a straightforward way to do that. One method would be to count by how many BFTs each edges is used, and use the opposite of these counts as weights for a minimum spanning tree algorithm, the intuition being that we want to preserve edges that are important for many nodes in X.

Expanding BFTs The second is based on the observation that, if $X = \{i\}$ is made of a single node, then $T^* = T_i$. Thus if $|X| > 1$, we want to extend that property by expanding BFTs simultaneously from all $i \in X$. Yet to ensure we get a spanning tree at the end, we need to avoid creating cycle. It's motivated by the cycle example presented in Figure 1 on the following page. In that case, this method could find T^* since the optimal edge will be among the last to be reached and thus cut.

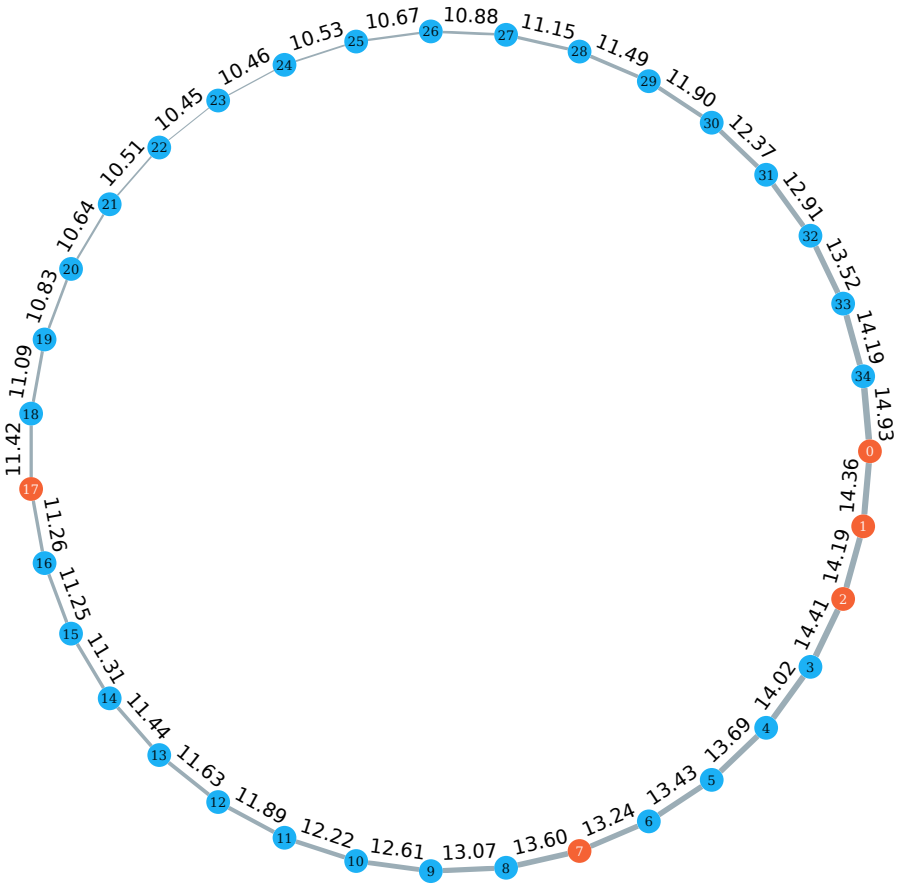


Figure 1: In a simple cycle, drawing a spanning tree T_e amounts to cutting one edge e. Here X is in orange and the label of each edge e is the cost $f(T_e)$ of the associated tree. The optimal edge is between 0 and 17 yet we observe that because of the concentration of X nodes around 0, the edge is pushed closer to 17.

In practice, we maintain a tree rooted at each node of X (line 7). We expand them in a Breadth first manner (line 9–20) as long as they don't encounter nodes belonging to another tree. When that happened, there are two cases. Either these two trees joined previously and there is nothing to do (line 16). Otherwise, we add an edge between the two tree and we make note that there is now a path between them (connecting all the others trees they may have encounter before). The formal description is provided in Algorithm 1 on page 7.

As said in the caption of Figure 1, the optimal edge is farther from 0, suggesting that larger subtrees (as defined by the number of nodes of X they contain) should grow faster (for instance the one starting from $0 - 1 - 2$). Taking into account this "**Momentum**" of each subtree, upon dequeuing a node at line 10, we skip it (i.e. enqueue it back) with probability based on the ratio of the size of the subtree this node belongs and the size of the currently largest subtree. This process favor the growth of subtree that contains more X nodes.

Modified SGT Although BFTs enjoy alluring practical performances, they don't provide any theoretical guarantees. Therefore we might adapt *Galaxy Tree* construction to that setting, by choosing star centers in priority in X. Practically, instead of sorting nodes by their degree, we sort them in lexicographical order according to $(1 - \mathbb{1}_X, \text{degree})$ (that is we put all nodes of X first). When collapsing the graph, we need to decide how to propagate the information about membership to X. The simplest way (currently implemented in the experiments) is to say that a star (i.e. a node in the future collapsed graph) belongs to X' (the successor of X in the new graph) if its center belongs to X. A more complicated method would replace the binary indicator function by a real value representing the strength of the tie with X (for instance, a star made only of nodes of X should be more strongly associated to X' than one with only one node in X). It's unclear whether this would provide any gain.

In addition, we compare with some baselines that don't rely very much on X.

BFT A Breadth First tree rooted at (one of) the node with highest degree in X.

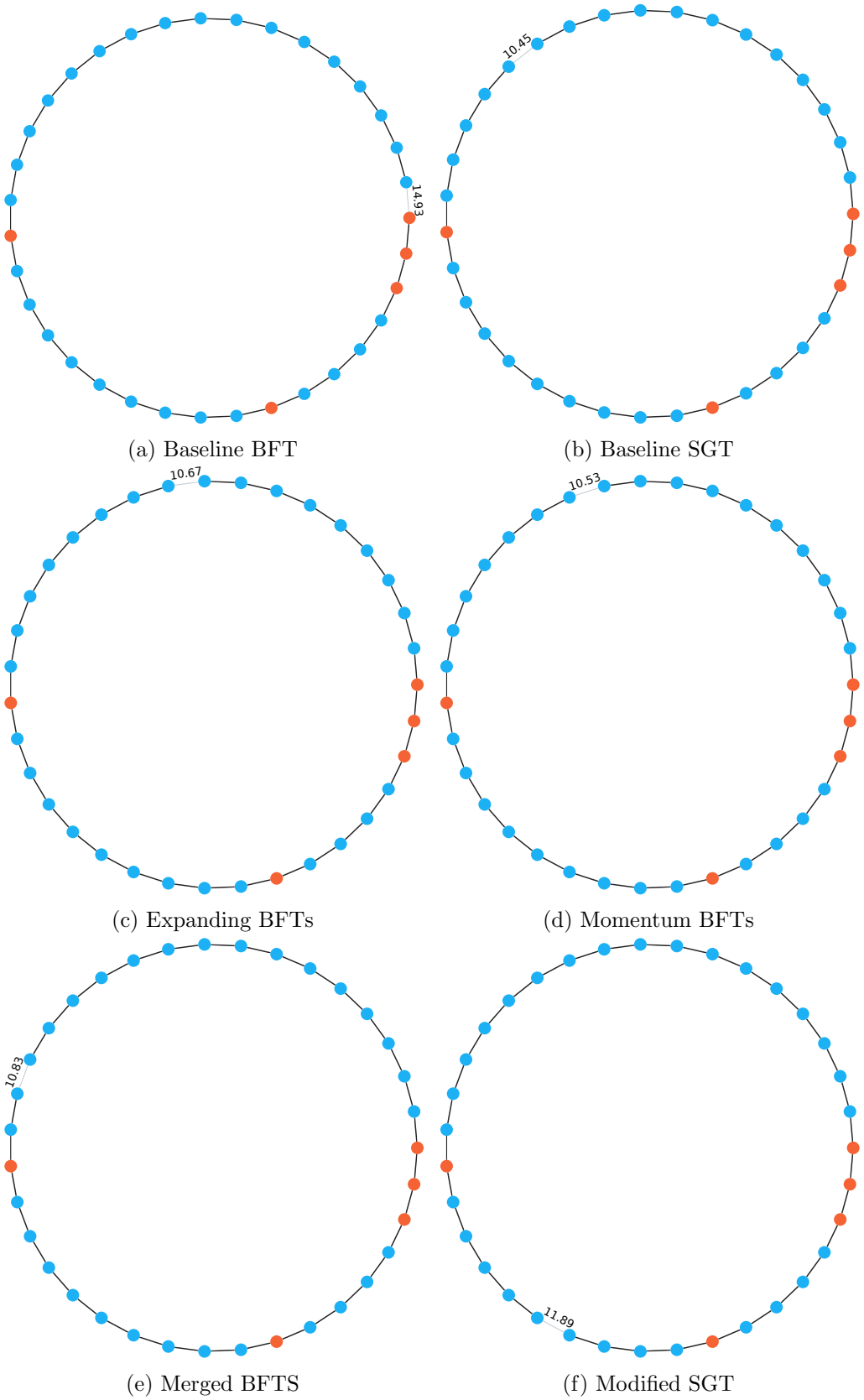
SGT The good old short galaxy tree, completely ignoring X.

Preliminary Results

We test these methods on a toy example in Figure 2 on the following page (the small cycle of Figure 1) and on two small ($n = 1024$) graph: preferential attachment Figure 3 on page 5 and a grid Figure 4 on page 6.

On these last two, more realistic examples, we see that the baselines outperforms all other methods introduced previously, especially the simple BFT. It also seems that the momentum heuristic improve the solution cost.

What it suggest to me is that it could be easier to start from the baseline BFT and see how it can be greedily improved based on path length computations.

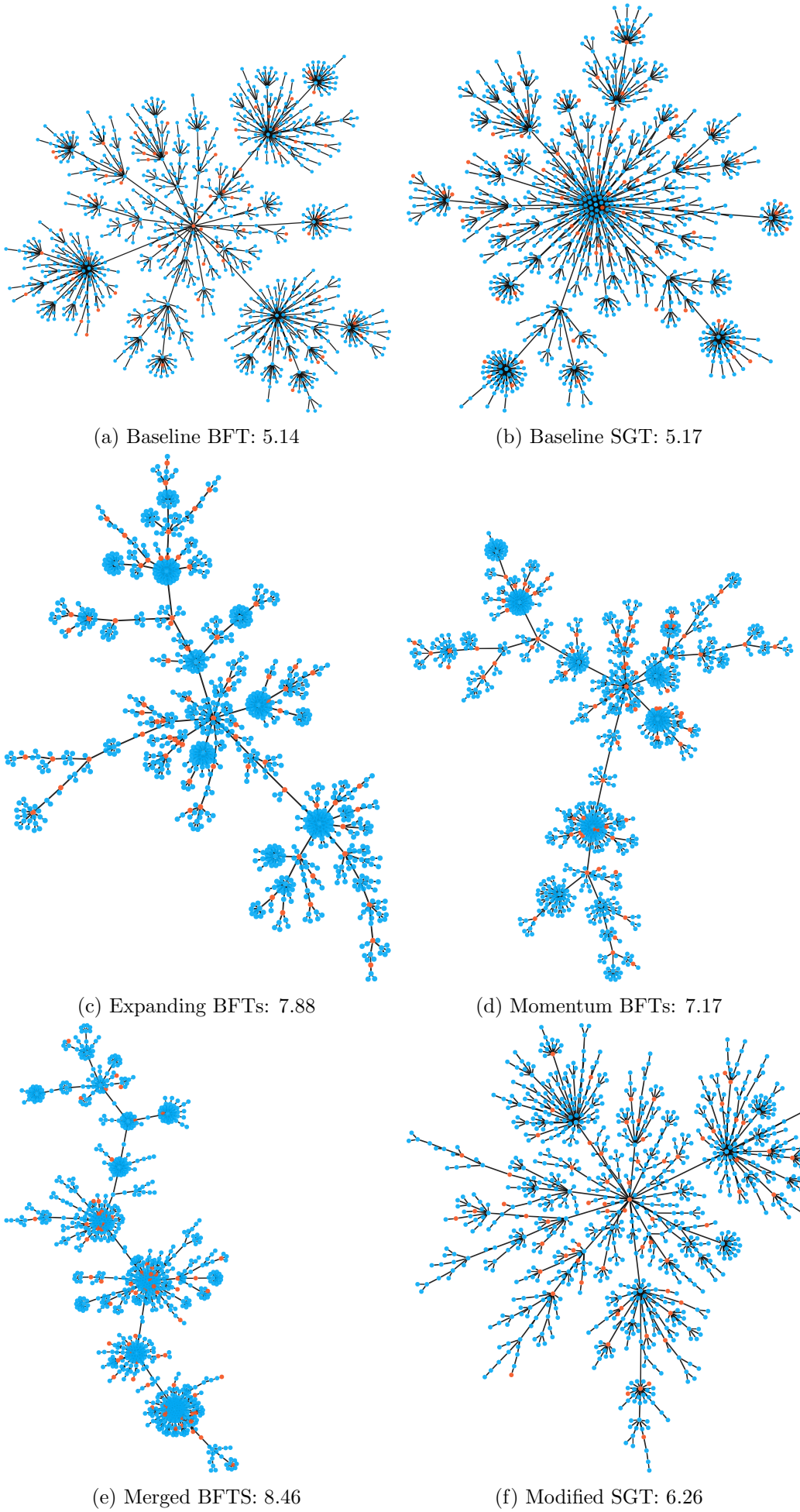


(a) Baseline BFT (b) Baseline SGT

(c) Expanding BFTs (d) Momentum BFTs

(e) Merged BFTS (f) Modified SGT

Figure 2: Results of the different methods and baselines on a simple cycle.



(a) Baseline BFT: 5.14

(b) Baseline SGT: 5.17

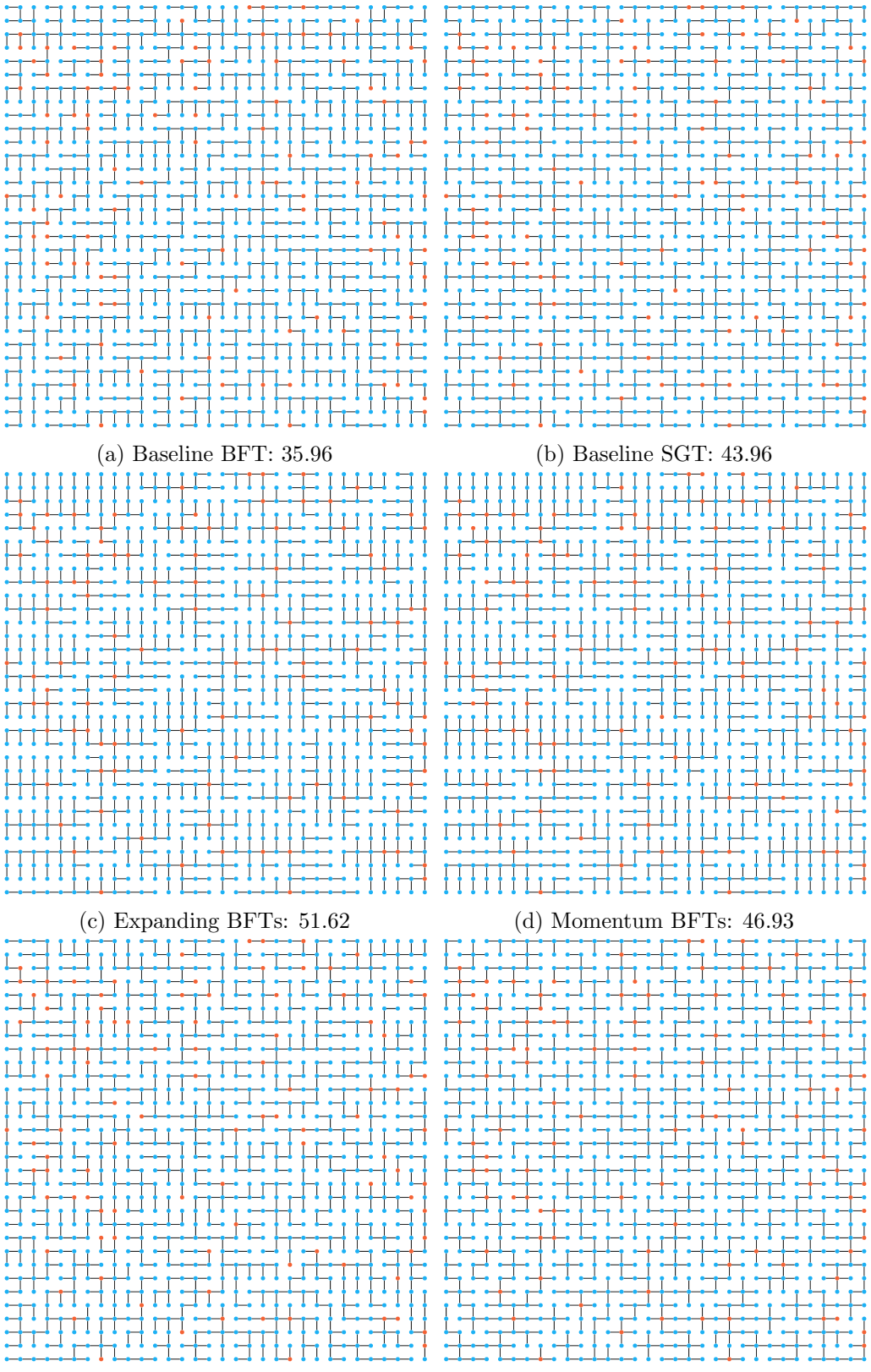
(c) Expanding BFTs: 7.88

(d) Momentum BFTs: 7.17

(e) Merged BFTS: 8.46

(f) Modified SGT: 6.26

Figure 3: Results of the different methods and baselines on a PA network.



(a) Baseline BFT: 35.96

(b) Baseline SGT: 43.96

(c) Expanding BFTs: 51.62

(d) Momentum BFTs: 46.93

(e) Merged BFTS: 52.34

(f) Modified SGT: 52.43

Figure 4: Results of the different methods and baselines on a grid network.

Algorithm 1 BFT from multiple roots

```
1: INPUT: G = (V, E), X ⊂ V
2: OUTPUT: T a spanning tree of G
3: T ← an empty tree, Q ← an empty queue
4: ∀i, j ∈ X, i < j, let Conn[i][j] = (i, j) ∈ E
5: ∀i ∈ V, let label[i] = i if i ∈ X else None
6: for all x ∈ X do
7:     Q.enqueue(x)
8: while Q ≠ ∅ do
9:     v ← Q.dequeue()
10:    for all w ∈ 𝒩(v) do
11:        if label[w] is None then
12:            Q.enqueue(w)
13:            T ← T ∪ {(v, w)}
14:            label[w] ← label[v]
15:        else if Conn[label[v]][label[w]] is true then
16:            continue
17:        else                          ▷ v and w are not yet connected through T
18:            T ← T ∪ {(v, w)}
19:            Conn[label[v]][label[w]] ← true
20:            update Conn
```