Bounds and Approximations for Maximum Leaf Spanning Tree

Marina Knittel[1], Yusuf Alnawakhtha[1], Nick Franzese[1], and Alexander Levine[1]

[1] *Department of Computer Science , University of Maryland*

1 Introduction

1.1 Motivation

In this paper, we will discuss state of the art approximation algorithms for finding the spanning tree of a connected graph with the maximum number of leafs. This problem is referred to as Maximum-Leaf Spanning Tree problem (MLST) and it has important applications in the field of network design, where it is sometimes beneficial to maximize the number of nodes connected to one peer [1]. It is also applied to the fields of distribution systems and circuit design [2].

In addition, we will be discussing the weighted version of the problem and a version of the problem where we place a limit on the max degree of the spanning tree. These variations of the problem can be applied the same fields addressed above. For example, the weighted version of the problem is relevant to network design where it is beneficial for some types of servers to have less strain than others. The limited max degree variation of the MLST could also be applied to circuit or communication layouts where the nodes have a limit on how many other nodes they can be connected to.

1.2 Preliminaries

The unweighted MLST problem considers a connected graph G with undirected edges and unweighted vertices. The leafs of the graph are defined as vertices of degree one. A spanning tree of G is a subset of the graph G that connects all the vertices of G and contains no cycle. The goal of the MLST problem is to find a spanning tree of G where the number of leafs in the spanning tree is maximized.

The weighted version of the problem has the same set up as the unweighted problem, except that the nodes of G are weighted. Instead of finding a spanning S where $|L(S)|$ is maximized, where $L(S)$ is the set of leafs of S, we want to find the spanning tree where the sum of the weights of the leafs is maximized. Namely, we want to find a spanning tree S such that $\sum_{v \in L(S)} w(v)$ is maximal, where $w(v)$ is the weight of the vertex v. We will also refer to the decidable version of the weighted MLST problem which has the additional integer input k and the goal is to answer whether or not there exists a spanning tree S such that $\sum_{v \in L(S)} w(v) \geq k$.

The limited-degree MLST problem has the same set up as the unweighted MLST problem, with the additional input of an integer k that represents the maximum degree of the spanning tree. In other words, the goal is to find the spanning tree S with the maximum number of leaves such that $d(v) \leq k \qquad \forall v \in S$, where $d(v)$ is the degree of vertex v.

1.3 Existing Results

The unweighted MLST problem is NP-Hard by reduction from the connected dominating set problem [3]. The best known approximation algorithm is a 2-approximation designed by Solis-Oba, Bonsma, and Lowski [2]. We will discuss this algorithm in Section 2.1.

We did not find an approximation algorithm for the weighted MLST problem. However, there has been a lot of progress in exploring the fixed parameter complexity of the problem. Namely, Jansen has found that the decidable weighted MLST problem, with input k as the minimum weight, admits $5.5k$ kernel [4]. However, we will not be going over kernelization in this paper.

We additionally propose a promising approach we initially had for the problem in Section 2.2, but we found that our approach cannot even guarantee finding a valid solution even if one exists.

2 Results

2.1 Unweighted

In the unweighted, undirected maximum-leaf spanning tree problem, the goal is to find the spanning tree of an unweighted graph with the maximum number of leaves. This problem is NP-hard; as of 2017, the best known polynomial-time approximation algorithm, given by Solis-Oba et. al., has an approximation ratio of two. [2] In this section, we will present Solis-Oba's algorithm and summarize the proof of the approximation ratio. In the process, we make note of a minor error in the proof, which fortunately does not affect the approximation ratio.

The algorithm is given as Algorithm 1. We define $N_G(v)$ as the vertices adjacent to vertex v in G. The algorithm runs in three phases. In the first two phases, a forest is constructed, while in the last phase, the components of the forest are connected arbitrarily. When constructing the forest F, vertices are only added through the 'expand' operation: this adds the vertex v to F if it is not already included, adds all of v's unvisited neighbors $N_G(v) \setminus V(F)$ to F, and adds the edges connecting v to vertices in $N_G(v) \setminus V(F)$ to F as well. Because this is the only way in which F grows, any vertex in F which has adjacencies not in F must be a leaf in F.

In Phase 1, vertices are expanded according to four prioritized cases: if a higher priority case holds true of some vertex v, it will be expanded before other vertices.

Case 1 expands a vertex (a leaf) already in F, with two or more neighbors not in F. This increases the number of leaves by at least 1, and the number of non-leaves by 1.

Case 2 expands a vertex (a leaf) already in F, with one neighbor w not in F, such that w has at least three neighbors not in F. It then expands w. This increases the number of leaves by at least 2, and the number of non-leaves by 2.

Case 3 expands a vertex (a leaf) already in F, with one neighbor w not in F, such that w has two neighbors not in F. This increases the number of leaves by one, and the number of non-leaves by 2. This result is not ideal, because it increases the number of non-leaves in the forest more than the number of leaves. In order to keep track of the number of times that this happens, which will be useful in proving the approximation bound later, the vertices w expanded by this case are referred to as "black vertices", and are represented by the set B.

Case 4 expands a vertex not in F, with at least three neighbors not in F. This immediately increases the number of leaves by at least three, and the number of non-leaves by one; however, because this creates a new component of the forest, one of the leaves of this component will eventually, in Phase 3, have to be converted into a non-leaf, as will one leaf of some other component. This means that net-one leaf is created and three non-leaves are created. Again, we keep track of these instances, defining a set R of roots of components, where the root is defined as the first vertex added (by Case 4) to the tree.

Phase 2 of the algorithm arbitrarily expands leaves in F until every vertex belongs to the forest. (Note that this does not decrease the total number of leaves in the graph, because every expansion destroys one leaf and creates at least one leaf).

Phase 3 connects components of the graph. Two leaves are destroyed upon creating each connection.

We now summarize the ideas behind the proof of the approximation ratio. Given the leaf creation counts described in the explanation of the algorithm above, it is relatively simple to make in inductive argument to

Algorithm 1 Solis-Oba's algorithm for MLST

Require: A connected graph G.

$F \leftarrow$ a forest, initially empty.
(Phase 1:)
loop
  if (Case 1) $\exists v \in V(F) : |N_G(v) \setminus V(F)| \geq 2$ **then**
    $V(F') \leftarrow V(F) \cup N_G(v)$
    $E(F') \leftarrow E(F) \cup \{(v,u), \forall u \in (N_G(v) \setminus V(F))\}$
  else if (Case 2) $\exists v \in V(F) : |N_G(v) \setminus V(F)| = 1$ and $\exists w \in (N_G(v) \setminus V(F)) : |N_G(w) \setminus V(F)| \geq 3$ **then**
    $V(F') \leftarrow V(F) \cup \{w\} \cup N_G(w)$
    $E(F') \leftarrow E(F) \cup \{(v,w)\} \cup \{(w,u), \forall u \in (N_G(w) \setminus V(F))\}$
  else if (Case 3) $\exists v \in V(F) : |N_G(v) \setminus V(F)| = 1$ and $\exists w \in (N_G(v) \setminus V(F)) : |N_G(w) \setminus V(F)| = 2$ **then**
    $V(F') \leftarrow V(F) \cup \{w\} \cup N_G(w)$
    $E(F') \leftarrow E(F) \cup \{(v,w)\} \cup \{(w,u), \forall u \in (N_G(w) \setminus V(F))\}$
  else if (Case 4) $\exists v \in (V(G) \setminus V(F)) : |N_G(v) \setminus V(F)| \geq 3$ **then**
    $V(F') \leftarrow V(F) \cup \{v\} \cup N_G(v)$
    $E(F') \leftarrow E(F) \cup \{(v,u), \forall u \in (N_G(v) \setminus V(F))\}$
  else Proceed to Phase 2
  $V(F) \leftarrow V(F')$
  $E(F) \leftarrow E(F')$
(Phase 2:)
while $V(F) \neq V(G)$ **do**
  Select $v \in V(F) : |N_G(v) \setminus V(F)| \geq 1$
  $V(F') \leftarrow V(F) \cup N_G(v)$
  $E(F') \leftarrow E(F) \cup \{(v,u), \forall u \in (N_G(v) \setminus V(F))\}$
  $V(F) \leftarrow V(F')$
  $E(F) \leftarrow E(F')$
(Phase 3:)
while F not connected **do**
  Select $(v,u) \in E(F) : v$ and u are in different components of F
  $E(F) \leftarrow E(F) \cup (v,u)$
return F, now a tree.

construct lower bound the number of leaves in the final tree T:

$$|L(T)| \geq \frac{|V(F^*)| - |B|}{2} - |R| + 3$$

Where F^* denotes the forest as constructed at the end of Phase 1.

Our goal then is to upper-bound the number of leaves in the optimal maximum leaf spanning tree T^*, in terms of the quantities $|V(F^*)|$, $|B|$, and $|R|$. This requires considering how the optimal spanning tree T^* traverses the forest F^* created by Phase 1 of the algorithm.

We first note that:

$$|L(T^*)| = |V(F^*)| - (|V(F^*) \setminus L(T^*)| - |L(T^*) \setminus V(F^*)|)$$

To upper bound $|L(T^*)|$ in terms of $|V(F^*)|$, $|B|$, and $|R|$, we lower bound the quantity $(|V(F^*) \setminus L(T^*)| - |L(T^*) \setminus V(F^*)|)$ by defining an injective mapping f from the union of three disjoint sets of vertices (to be specified below) S_1, S_2, and S_3 to the set of vertices $V(F^*) \setminus L(T^*)$. In particular

$$|S_1| = |R| - 1, |S_2| = |L(T^*) \setminus V(F^*)|, |S_3| = |B| + |R| - 1$$

So:

$$|V(F^*) \setminus L(T^*)| \leq |B| + 2|R| - 2 + |L(T^*) \setminus V(F^*)|$$

Or:

$$|V(F^*) \setminus L(T^*)| - |L(T^*) \setminus V(F^*)| \geq |B| + 2|R| - 2$$

Which lets us establish an appropiate upper bound on $|L(T^*)|$ by substituting in to the above:

$$|L(T^*)| \leq |V(F^*)| - |B| - 2|R| + 2$$

Comparing to the bound for $|L(T)|$ gives the appropriate approximation ration of 2.

To show this injection f, the sets S_1, S_2 and S_3 must first be defined. In particular, defining S_1 requires introducing a notion of a T^*-predecessor of a vertex. To define this, we note that we can arbitrarily define any vertex in a tree as its root. We then define the root of T^* as the root $r_0 \in R$ of the first forest component added to F^*. A vertex w is a T^* predecessor to v if it lies on the path in T^* between r_0 and v. We also define $T_0, ..., T_{|R|-1}$ as the components of F^*, in order of creation. Now we are ready to define the sets:

S_1: for each tree T_i except T_0, select an arbitrary vertex t_i such that t_i has no T^*-predecessors in T_i. Note that at least one such vertex must exist for each T_i, giving $|S_1| = |R| - 1$ as stated above.

$S_2 = L(T^*) \setminus V(F^*)$.

$S_3 = B \cup (R \setminus r_0)$. Because roots and black vertices are disjoint, this gives $|S_3| = |B| + |R| - 1$. Solis-Oba makes an error here: the authors earlier introduce a variable $k = |R| - 1$, and then give $|S_3| = |B| + k - 1$; it should be $|S_3| = |B| + k$. This leads them to give the upper bound on leaves as

$$|L(T^*)| \leq |V(F^*)| - |B| - 2k + 1 = |V(F^*)| - |B| - 2|R| + 3$$

which is looser than it could be by a constant value of 1. Note however that this inequality is still correct, so the approximation ratio is unaffected.

The authors go on to show that S_1, S_2 and S_3 are disjoint (which is clearly true of S_2 and the others, because the others are subsets of $V(F^*)$, but requires more work to demonstrate for S_1 and S_3), to define a mapping f from each set to $V(F^*) \setminus L(T^*)$, and to show that this mapping is injective. This requires a series of technical lemmas that are unfortunately beyond the scope of this summary.

2.2 Limited Degree

We are interested in adapting Solis-Oba's max leaf spanning tree algorithm to handle limiting the degree of the tree. In this version of the problem, we are given a parameter k that determines the maximum allowed

degree in the tree. We must output a tree with a maximum possible number of leaves with maximum degree k.

The algorithm uses Solis-Oba's max leaf spanning tree as a near blackbox. To do this, we utilize each phase of the algorithm unchanged, and insert an additional step for degree limitation. Our algorithm thus starts with forest construction and expansion like Solis-Oba does, then adds an additional degree limiting step, and then completes by connecting trees in the forest in the same way as the Solis-Oba algorithm. As long as our degree limiting step does not violate the assumptions required for Solis-Oba's analysis, we would achieve the same approximation ratio.

Our limiting degree step is based off of the minimum degree spanning tree approximation algorithm described in Williamson and Shmoys [5]. This is a local search algorithm that iteratively reduces the degree of high degree vertices by finding cycles. We were interested in local search algorithms because this allows us to provide an arbitrary initialization of the tree. So if we take each individual tree after expanding our forest, we can apply this algorithm and hopefully find we will not reduce the number of leaves significantly. That way, we hope our approximation still holds.

Our version of this algorithm can be seen in Algorithm 2. We iterate over each tree T in our forest. As long as there is some vertex $v \in V(T)$ such that its degree $d(v) > k$, we perform our cycle-finding reduction. To do this, we find some edge adjacent to two other vertices in T whose degrees are less than $d(v) - 1$. Then we swap this new edge for one of the edges adjacent to v in this cycle. This way, we reduce the degree of v by 1 and increase the degrees of the other two nodes by one. But since those vertices had degrees less than $d(v) - 1$, they must have degrees less than $d(v)$ after this swap. Do this process until the max degree in T is at most k for all trees T. If we are unable to achieve this, we immediately fail.

Algorithm 2 Degree Limiting

Require: A connected graph G, a spanning forest $F \subseteq G$ with set of trees $\mathcal{T}$, an integer k, and a degree function $d : V \to \mathbb{R}$

```
for all T ∈ 𝒯 do
    while ∃v ∈ V(T) such that d(v) > k do
        e ← (u, w) ∈ E(G) where e ∉ E(T), u, w ∈ V(T) − {v}, and d(u), d(w) < d(v) − 1
        if ∃C ⊆ E(T) ∪ {e} a cycle with e′ = (v′, v) ∈ C then
            E(T) ← E(T) ∪ {e} − {e′}
```

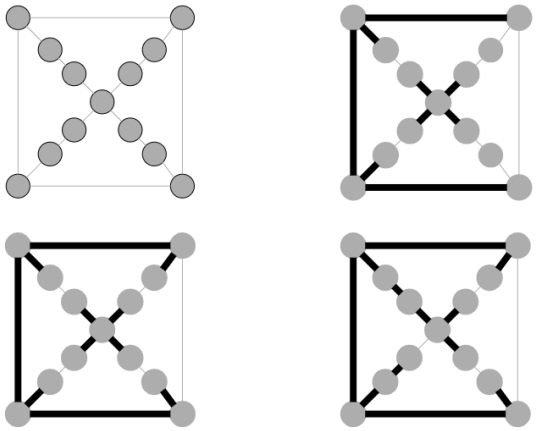
A potential problem with this process is that the edge we add may connect leaves in T. This means that any step in this algorithm may decrease the number of leaves in our forest by 2. The number of times this occur is only bounded by the number of reduction steps in the algorithm. This is why we elected to insert the degree limiting step before the tree connection step: because the state of the forest at that point allows Solis-Oba to make a number of assumptions in their analysis that we thought we may be able to use to limit the number of degree limiting steps required. Performing degree limiting after connecting the forest will render some of the assumptions invalid at that point. However, an exploration of our algorithm with degree limiting occurring after the forest connection step may still be interesting.

Interestingly, the real flaw in this algorithm occurred when we were unable to find spanning trees with max degree k even if there existed such trees. In this case, our algorithm fails to even produce a result, much less an approximation, even when a valid result exists. This counterexample can be seen in Figure 1.

In the first forest-constructing stage of our algorithm, we will construct two trees. The first will contain the five innermost vertices, while the other contains all corner vertices, as well as two more vertices adjacent to corners. We call these the inner tree and outer tree respectively. In the next step, the algorithm might expand the outer tree to connect with the two remaining vertices. At this point in our algorithm, we have a spanning forest.

Now, we apply the degree limiting step. Say our minimum degree parameter is $k = 3$. We consider the inner tree. Note there are no additional edges we can use in consideration of just this tree, therefore we cannot reduce the maximum degree of this tree to 3. Thus our algorithm fails. However, in this graph, there exists a spanning tree with maximum degree 3. Therefore, our algorithm does not provide a solution where

Figure 1: Counterexample for limited degree solution. Top left: counterexample graph. Top right: forest construction after step 1. Bottom left: forest expansion after step 2, and the graph state after degree limiting fails. Bottom right: successful spanning tree with max degree 3



it should.

Note that in this case, our algorithm would find a feasible spanning tree with max degree 3 if we apply the degree limiting step after the final step of the Solis-Oba algorithm. However, as mentioned previously, there are likely instances where such an algorithm fails. Nevertheless, future research in this area may be useful.

The reason our algorithm fails is because we inherently sparsify the graph before applying our degree limiting step. This occurs because all edges with end points in different trees will not be considered for edge swaps. Thus we lose a lot of power in reducing the degree of the graph.

This led to the consideration of explicitly subdividing graphs into sparse and dense components to handle separately. For a dense graph, where neighborhoods around vertices are more well connected, applying degree limitations via cycle finding is easy because there are so many neighboring edges to choose from. We conjecture that if we assume our graph is dense, we will be able to achieve a constant ratio approximation algorithm with these methods. However, the trick is handling sparse components. Sparse components would be simple to work with if sparsity implied low vertex degrees, however sparsity only defines the interconnectedness of neighborhoods. Thus any vertex in a sparse component could have an arbitrarily high degree, which may be difficult to resolve. This idea was only in its early stages before completion of this project. It may have potential, though we do not know if there are ways to handle sparse graph components explicitly.

2.3 Lower Bound

It is a known result that the Max Leaf Spanning Tree problem is NP-hard, by reduction from the Connected Dominating Set problem. It is also known that MLST is APX-hard, that is it has no polynomial time approximation scheme unless P=NP[3]. However, a lower bound on an approximation factor for this problem remains unknown. We conducted some exploration on this front. Finding the Connected Dominating Set problem difficult to leverage towards an approximation bound, we constructed a reduction to MLST from the Set Cover problem.

Definition 1. *The Set Cover problem takes as inputs a universe of n elements $U = \{x_1, x_2, ..., x_n\}$, a family of m sets $S = \{s_1, s_2, ..., s_m\}$ with each $s_i \subseteq U$, and an integer k. The Set Cover problem asks whether there exists $A \subseteq S$ with the following properties:*

- $|A| \leq k$

- $U = \bigcup_{s \in A} s$

Theorem 2. *Finding the Max Leaf Spanning Tree is NP-hard, by reduction from Set Cover.*

Proof. We begin by constructing a graph which characterizes an arbitrary instance of a given Set Cover problem via the following algorithm.

Algorithm 3 Construct Set Cover Graph G

Require: A universe U, a family of sets S, and an integer k as in the Set Cover problem

```
G = (V, E) ← an empty graph
for all x_i ∈ U do
    add two vertices x'_i and x''_i to V
    add an edge (x'_i, x''_i) to E
for all s_i ∈ S do
    add a vertex s'_i to V
    for all x_j ∈ s_i do
        add an edge (s'_i, x'_j) to E
for all s'_i vertices ∈ V do
    add an edge between it and every other s'_i vertex
```

We then solve the Max Leaf Spanning Tree problem with input G. If the MLST contains k or fewer s'_i vertices that are not leaves, we answer affirmative to the given instance of the Set Cover problem. Otherwise we answer negative.

To see that this approach correctly decides the Set Cover problem, first observe that for any spanning tree on a graph constructed by Algorithm 3, the x''_i nodes must always be leaves (since they have degree 1) and the x'_i nodes can never be leaves (since by construction they have two incident edges which must both be included in any spanning tree in order for the x''_i nodes to be reached). Thus to reason about the optimal number of leaves that a spanning tree can have, we may focus our attention on the s'_i nodes.

For an arbitrary spanning tree T on G, it must be that each x' node is adjacent to some non-leaf s'_i on T. To see this, observe that any s' node that is a leaf on T has a single incident edge on T to either some x'_a or some s'_b by the construction of G. In the former case, it must be that x'_a is connected to some other s'_c, or else T could not be spanning, since this would mean that s'_i, x'_a, and x''_a would form a component that could not be connected to the rest of T.

Consequently, if an arbitrary s'_i node is a leaf on T, it must be that for a such that $x_a \in s_i$, x'_a is adjacent to some non-leaf node s'_j on T with $i \neq j$. By construction of G, this is only possible if $x_a \in s_j$.

Consider then the set Q of non-leaf s' nodes on T, and the set $R = \{s | s' \in Q\}$. It must be that R is a set cover over U since by the above argument, on T each x' is adjacent to some node in Q, and any x'_i can be adjacent to any s'_j if and only if $x_i \in s_j$.

Finding the spanning tree with the maximum number of leaves also finds the minimal cardinality of Q and consequently R, since by a previous argument the number of non-s' leaves on a spanning tree over G is fixed. Thus, computing the Max Leaf Spanning Tree of G equivalently computes the smallest set cover for arbitrary inputs of U and S. Thus if $|Q| \leq k$, we are correct in answering affirmative, and we are correct in answering negative otherwise.

Algorithm 3 trivially takes polynomial time. Thus, if there exists a polynomial time algorithm for obtaining the Maximum Degree Spanning Tree of an arbitrary graph, then the Set Cover problem is decidable in polynomial time.

□

Unfortunately, it also proved difficult to leverage this new reduction towards an approximation bound. We had hoped that placing weights on vertices in G, an output graph of Algorithm 3, would provide information about a given Set Cover problem that might be robustly distinguishable even from an approximate solution,

but we discovered that approaches we designed could only be used to distinguish whether a particular set in S was a necessary part of an optimal solution. Further, the structure of this reduction does not appear robust to alterations of the topology of G, which would likely be necessary to leverage information from an approximate MLST solution over G towards the optimal solution.

3 Conclusion

This report provides an overview of important algorithms and bounds related to the problem of finding a maximum-leaf spanning tree (MLST). While this problem has putative applications in systems and networks, it is a rather understudied topic. The most interesting result from Solis-Oba, as exhibited in Section 2.1, is a 2-approximation algorithm for the unweighted MLST problem. While this is significant progress, it is noteworthy that this solution was formulated decades before its publication.

We provide a basic exploration of an algorithm for MLST with limited maximum degrees by adding a limiting degree step to Solis-Oba's algorithm. However, we show this modification is too naive, as it is unable to provide a valid solutions in certain cases where one does exist.

Finally, we provide another natural reduction from MLST to an NP-hard problem, Set Cover. This may provide additional intuition into the hardness of MLST, and perhaps may lead to methods for approximating it or its variants, however we were unable to do so in the present study.

We have reason to believe that, in the future, this work may be used to achieve stronger findings about MLST. For instance, in its counterexample, our limited degree MLST algorithm provided a spanning tree that was similar to a valid tree (ie, one with max degree at most k). Perhaps a small tweak to this algorithm will create a functioning approximation algorithm. In addition, our reduction to Set Cover could provide the aforementioned insights.

References

[1] B. Jansen, "Fixed parameter complexity of the weighted max leaf problem," Ph.D. dissertation, Master's thesis, Utrecht University, 2009. URL: http://www. cs. uu. nl/education/scripties/scriptie. php, 2009.

[2] R. Solis-Oba, P. Bonsma, and S. Lowski, "A 2-approximation algorithm for finding a spanning tree with maximum number of leaves," *Algorithmica*, vol. 77, no. 2, pp. 374–388, Feb 2017. [Online]. Available: https://doi.org/10.1007/s00453-015-0080-0

[3] G. Galbiati, F. Maffioli, and A. Morzenti, "A short note on the approximability of the maximum leaves spanning tree problem," *Inf. Process. Lett.*, vol. 52, no. 1, pp. 45–49, Oct. 1994. [Online]. Available: https://doi.org/10.1016/0020-0190(94)90139-2

[4] B. Jansen, "Kernelization for maximum leaf spanning tree with positive vertex weights," in *International Conference on Algorithms and Complexity*. Springer, 2010, pp. 192–203.

[5] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 1st ed. New York, NY, USA: Cambridge University Press, 2011.

[6] J. Daligault and S. Thomassé, "On finding directed trees with many leaves," in *Parameterized and Exact Computation*, J. Chen and F. V. Fomin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 86–97.

[7] M. Drescher and A. Vetta, "An approximation algorithm for the maximum leaf spanning arborescence problem," *ACM Trans. Algorithms*, vol. 6, no. 3, pp. 46:1–46:18, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1798596.1798599