Bonsai: Growing Interesting Small Trees

Stephan Seufert, Srikanta Bedathur, Julián Mestre, and Gerhard Weikum

MPI–I–2010–5–005 November 2010

Authors’ Addresses

Stephan Seufert
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken
Germany
Email: `sseufert@mpi-inf.mpg.de`

Srikanta Bedathur
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken
Germany
Email: `bedathur@mpi-inf.mpg.de`

Julián Mestre
School of Information Technologies
University of Sydney
Sydney, NSW 2006
Australia

Gerhard Weikum
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken
Germany
Email: `weikum@mpi-inf.mpg.de`

Acknowledgements

This work is supported by DFG (German Research Foundation) within the priority research program SPP-1355: “Scalable Visual Analytics”.

Abstract

Graphs are increasingly used to model a variety of loosely structured data such as biological or social networks and entity-relationships. Given this profusion of large-scale graph data, efficiently discovering interesting substructures buried within is essential. These substructures are typically used in determining subsequent actions, such as conducting visual analytics by humans or designing expensive biomedical experiments. In such settings, it is often desirable to constrain the size of the discovered results in order to directly control the associated costs.
In this report, we address the problem of finding *cardinality-constrained* connected subtrees from large node-weighted graphs that maximize the sum of weights of selected nodes. We provide an efficient constant-factor approximation algorithm for this strongly NP-hard problem. Our techniques can be applied in a wide variety of application settings, for example in differential analysis of graphs, a problem that frequently arises in bioinformatics but also has applications on the web.

Keywords

Graph Mining, Heavy Subgraphs, Visual Analytics, Cardinality Constraints

Contents

1 Introduction

Given a large graph with weighted nodes, how can we efficiently identify a heavy, connected subtree within a given size? When each node exhibits an individual interestingness factor, how can we find small but highly interesting subnetworks?

The problem of discovering interesting subgraphs from large graphs has for long attracted the attention of researchers from different streams. A variety of measures are used for determining the interestingness of a subgraph, ranging from the sum of scores of selected nodes or edges [16, 17, 24], edge density [14, 18], or the frequency of its (isomorphism class) occurrence in the larger graph [19]. It is striking to see that the proposed methods don't offer any support to directly control the size of the discovered subgraph. As a consequence, the results can be extremely large, or their size can vary as an arbitrary function of the parameters of the algorithm. There are many application settings where it is not enough to identify heavy or interesting subgraphs, but it is also essential to keep their size small. A well-known application of this problem arises in the field of bioinformatics [1, 9]. In this setting, we are given the protein-protein interaction (PPI) network of an organism, where each node is annotated with a score signifying its deviation from normal behavior in response to a disease. In order to unearth the biological processes involved and thus aid the design of targeted drugs, it is important to identify not only subnetworks with high score, but also to limit their size in order to keep costs of biomedical trials manageably low.
Similar needs arise in visual analytics applications of large-scale graphs. Due to varying visual fatigue levels (either due to individuals or the device used), it is important to enable users to explicitly control the size of the output graph they are comfortable with for navigation. While substantial progress has been made in visual exploration of large graphs [22, 25], such a control still is not in the hands of the users.
In this report, we take first steps towards efficiently addressing these requirements

in graph mining. Specifically, we consider solving the following computationally hard problem: Given a large undirected graph, where a weight indicating individual score/relevance is associated with every vertex, identify a maximum-weight connected set of nodes whose size is upper-bounded by a user-specified threshold k. This set of nodes corresponds to a subtree of k nodes with maximal weight.
Our main contribution is an efficient constant-factor approximation algorithm for this strongly NP-hard problem. For any given cardinality k, our algorithm is guaranteed to discover a subtree spanning at most k vertices that sum to a weight of at least $\frac{1}{5(1+\epsilon)}$ times the weight of the optimal subtree of this size.
The remainder of this report is organized as follows: In the next chapter, we lay out the formal framework for our algorithm and show its relation to another well-known graph mining problem. In Chapter 3 we explain our algorithm in detail. In Chapter 4 we provide an experimental evaluation on synthetic and real-world graphs. Implementation issues are discussed in Chapter 5. A related problem is covered in Chapter 6 and the report is concluded in Chapter 7.

2

Preliminaries

For a given graph $G = (V, E)$ let $\mathcal{T}(G)$ denote the set of subtrees of G. For any integer k, let $\mathcal{T}_k(G)$ denote the set of subtrees of G spanning not more than k vertices:

$$\mathcal{T}_k(G) := \{T = (V_T, E_T) \in \mathcal{T}(G) \mid |V_T| \leq k\}.$$

Let f be a mapping defined on a set S. By abuse of notation, let $f(X) := \sum_{x \in X} f(x)$ for a subset $X \subseteq S$.

2.1 Cardinality-Constrained Weighted Trees

We address the following combinatorial optimization problem in the remainder of this report:

Problem 1: Node-Weighted k-Cardinality Tree (KCT)

Given Undirected graph $G = (V, E)$, a non-negative weight function defined on the vertices, $w : V \to \mathbb{R}_{\geq 0}$, and a cardinality $k \in \mathbb{N}$.

Goal Identify a subtree $T = (V_T, E_T)$ of G with the maximum sum of node weights that satisfies the cardinality constraint $|V_T| \leq k$:

$$T := \operatorname*{arg\,max}_{T \in \mathcal{T}_k(G)} w(V_T). \tag{2.1}$$

This problem was proven strongly NP-hard by Fischetti et al., using a reduction to the node-weighted Steiner tree problem [12].
Although a large body of literature exists for similar problems (like the variant of KCT with edge costs instead of node weights), the node-weighted KCT problem has not received much attention yet. The existing algorithms rely on:

- (meta-)heuristics that do not provide any guarantees, such as Tabu Search and Genetic Algorithms [4], Variable Neighborhood Search [5], and Ant-Colony Optimization [3], or
- Integer Programming via branch-and-bound to obtain exact solutions, however at the expense of worst-case exponential running time or
- reduction to the related k-MST problem (as described in Chapter 6).

2.2 Prize-Collecting Steiner Trees

As a subroutine, our algorithm solves carefully-chosen instances of the Prize-Collecting Steiner tree problem (PCST):

Problem 2: Prize-Collecting Steiner Tree (PCST)

Given Undirected graph $G = (V, E)$, a non-negative cost function defined on the edges, $c : E \to \mathbb{R}_{\geq 0}$, and a non-negative penalty function defined on the vertices: $\pi : V \to \mathbb{R}_{\geq 0}$.

Goal Identify a subtree $T = (V_T, E_T)$ of G, minimizing the sum of costs of the included edges and the penalties of the vertices not included:

$$T := \operatorname*{arg\,min}_{T \in \mathcal{T}(G)} c(E_T) + \pi(V \setminus V_T). \tag{2.2}$$

Note that this problem does not include a constraint on the size of T, rather we assign a penalty for the nodes that are not spanned.
The PCST problem, which is known to be NP-hard [23], has been studied intensively in the literature because many real-world problems – like utility network design – can be expressed in its terms. Several good approximation algorithms are known for the PCST. In their seminal work, Goemans and Williamson [15] propose an $\mathcal{O}(n^3 \log(n))$ clustering algorithm that guarantees an approximation ratio of $2 - \frac{1}{n-1}$:

Theorem 1 (Goemans and Williamson). There is a polynomial-time algorithm that, given an instance (G, c, π) of PCST, returns a tree $T \in \mathcal{T}(G)$ such that

$$c(E_T) + 2\pi(V \setminus V_T) \leq 2 \min_{S \in \mathcal{T}(G)} \{c(E_S) + \pi(V \setminus V_S)\}. \tag{2.3}$$

In this section we give a brief description of the approximation technique of Goemans and Williamson for the PCST, as described in [21]. The algorithm contains two stages: a growth and a pruning phase. In the growth phase, initially every vertex forms a singleton component (cluster). Every component is assigned a growth

potential that corresponds to the sum of penalties of all vertices included in the component. A component is called active if it has positive remaining potential and passive otherwise. Additionally, we maintain a residual value $r(e)$ for every edge e, that initially corresponds to the edge cost. The active components grow uniformly over time, meaning that for each time increment δ, the potential of each active component is reduced by δ. At the same time, the residual value of an edge with one active endpoint component is reduced by δ, the residual value of an edge with two active endpoint components is reduced by 2δ.
This growth procedure continues until either

- the potential of an active component reduces to 0 or
- the residual value $r(e)$ of an edge e reduces to 0.

In the former case, the endpoint is marked as inactive. In the latter case (we call the edge *e tight*), we merge both endpoint components of the edge into a new component. The potential of the newly formed component corresponds to the sum of potentials of its constituent two components. The growth phase continues until there are no more active components. The output of the procedure is the set of tight edges (which corresponds to a forest in the graph).
In Goemans and Williamson's algorithm, the growth phase if followed by a pruning phase. As this pruning step is not part of our algorithm, we omit its description and refer to the literature [15, 21].
It is worth noting that in their original paper, Goemans and Williamson reduce PCST to a rooted variant where we are given a designated vertex r, the root, which must be spanned by the output subtree. In order to obtain the algorithm in Theorem 1 one just runs the algorithm for the rooted version on each possible choice of r. However, due to the large size of the problem instances, this guessing step would be prohibitively slow for our purposes. Therefore, in our implementation, we use the algorithm of Johnson et al. [21], which is also based on the original work of Goemans and Williamson but works in the unrooted setting and offers an approximation guarantee of 2 while avoiding the guessing step altogether. The time complexity of this algorithm is $\mathcal{O}(n^2 \log(n))$. We will denote by UnrootedGrowth(G, c, π) the output of this algorithm on the PCST instance (G, c, π).

3 Algorithm

In this section we formally describe our algorithm. Our approach is to solve a number of carefully constructed PCST instances. We will use implicitly the framework of Lagrangian relaxation for approximation algorithms introduced by Jian and Vazirani [20] for location problems and by Chudak et al. [7] for Steiner tree problems. However, we only describe the parts relevant to our analysis. More specifically, we avoid introducing the underlying linear program and its Lagrangian relaxation.

3.1 Main Idea

The key of our algorithm is to construct and solve instances of the PCST problem in such a way that we can guarantee a constant-factor approximation to our original KCT problem. Throughout the rest of this section, we denote by OPT the weight of the optimal solution to the KCT instance at hand. The following theorem states our main result:

Theorem 2. There is an efficient algorithm that, given an instance (G, w, k) of the KCT problem, returns a tree T of at most k vertices such that $w(V_T) \geq \frac{\mathsf{OPT}}{5(1+\epsilon)}$ for any $\epsilon > 0$.

The problem is somewhat easier to solve if OPT is known beforehand, therefore we assume for now that the value is known. Indeed this will not be the case in our applications but we can easily guess its value up to an ϵ multiplicative error using binary search, as we will show in Section 3.4. In the next subsections, we describe the algorithm that satisfies Theorem 2 and prove its correctness.

3.2 Basic Algorithm

Given an instance of KCT and the value OPT, we derive several instances of the PCST problem. For this purpose, we identify the node weights with penalties and set the cost of every edge in the graph to 1. By scaling these node penalties (that is, multiplying them with a factor $\lambda \in \mathbb{R}_{>0}$), we can indirectly control the size of the output solution. For instance, if we use a multiplicator $\lambda_1 = 0$, the optimal solution of the associated PCST instance is given by the empty tree, whereas for a sufficiently large factor, e. g. $\lambda_2 > n \max_{e \in E} c(e)$, the optimum is any spanning tree of the graph.
We will use the existing algorithm of Johnson et al. [21] for the PCST (which provides a 2-approximation [11]) to obtain a tree that has a weight of at least OPT and is as small as possible.
The idea is to perform binary search over the range of scale factors $\lambda \in [\lambda_1, \lambda_2]$. At each step of this binary search procedure, we solve the PCST instance using the λ-scaled penalties. If the returned tree has weight of at least OPT, we decrease λ, thus requesting a smaller tree in the next iteration. If the returned tree has weight less than OPT, we increase λ, thus allowing for a larger output solution in the next run. This binary search procedure is continued until the final interval is sufficiently small.

As the solution for the original KCT problem, we finally extract the heaviest subtree spanning k vertices from the tree obtained in the last solved PCST instance. For this purpose, we use a dynamic programming procedure called $\mathsf{TreeDP}(T, w, k)$, consisting of the algorithm by Blum [2]. The complete procedure is described in Algorithm 1.

In the following we will provide a theoretical analysis of our approach and show that it ultimately leads to a constant-factor approximation of the KCT problem.

3.3 Approximation Guarantee

Let T_1 and T_2 be the two trees the algorithm holds at the end of the while loop of Algorithm 1 (just before line 12). We now establish a few important properties about these trees. Our aim is to show that a certain convex combination of T_1 and T_2 has large weight and small size. For the sake of brevity we will denote $w(V_{T_i})$ by W_i and $|V_{T_i}|$ by S_i, for $i = 1, 2$. The coefficients of the above-mentioned convex combination are as follows:

$$\alpha_1 = \frac{W_2 - \mathsf{OPT}}{W_2 - W_1} \quad \text{and} \quad \alpha_2 = \frac{\mathsf{OPT} - W_1}{W_2 - W_1}. \tag{3.1}$$

Algorithm 1: HeavySubtree(G, w, k, OPT)

```
Data: Graph $G = (V, E)$, weight function $w : V \to \mathbb{R}_{\geq 0}$, cardinality
      $k \in \mathbb{N}$
begin
    $[\lambda_1, \lambda_2] \leftarrow [0, n/(w(V) - \mathsf{OPT})]$            ▷ initial penalty interval
    $T_1 \leftarrow (\arg\max_{v \in V} w(v), \emptyset)$                        ▷ tree of heaviest node
    $T_2 \leftarrow$ SpanningTree$(G)$                                          ▷ any spanning tree of $G$
    while $\lambda_2 - \lambda_1 \geq \frac{1}{w(V) - \mathsf{OPT}}$ do
        $\lambda \leftarrow \frac{\lambda_1 + \lambda_2}{2}$
        $T \leftarrow$ UnrootedGrowth$(G, \mathbf{1}, \lambda w)$              ▷ solve PCST with unit edge costs
                                                                               and $\lambda$-scaled node penalties
        if $w(V_T) \leq \mathsf{OPT}$ then
            $(\lambda_1, T_1) \leftarrow (\lambda, T)$
        else
            $(\lambda_2, T_2) \leftarrow (\lambda, T)$
    $T_1 \leftarrow$ TreeDP$(T_1, w, k)$
    $T_2 \leftarrow$ TreeDP$(T_2, w, k)$
    $T \leftarrow \arg\max_{w(V)} \{T_1, T_2\}$                                 ▷ heaviest subtree of $T_1$ or $T_2$ hav-
                                                                               ing $k$ vertices
    return $T$
```

Lemma 1. Let T_1 and T_2 be the trees right after while loop in our algorithm. Then

(i) $\alpha_1 W_1 + \alpha_2 W_2 = \mathsf{OPT}$, and

(ii) $\alpha_1 S_1 + \alpha_2 S_2 < 2k$.

Proof. The first property follows easily from the definition (3.1). Indeed, the coefficients α_1 and α_2 have been defined so that property (i) holds.
Let T^* be a tree on k vertices with weight OPT. For the second property we need to exploit the approximation guarantee (2.3) of UnrootedGrowth[1]:

$$S_1 - 1 \leq 2\big(k - 1 + \lambda_1(W_1 - \mathsf{OPT})\big) \tag{3.2}$$
$$S_2 - 1 \leq 2\big(k - 1 + \lambda_2(W_2 - \mathsf{OPT})\big) \tag{3.3}$$

Taking the convex combination of these two inequalities using the coefficients (3.1)

[1]Even though the initial values of T_1 or T_2 are not the output of UnrootedGrowth, it is easy to verify that Equations (3.2) and (3.3) hold for these trees as well.

we get

$$\begin{aligned} \alpha_1 S_1 + \alpha_2 S_2 &\leq 2(k-1) + (\lambda_2 - \lambda_1)\big(w(V) - \mathsf{OPT}\big) + 1 \\ &< 2k, \end{aligned}$$

which gives us the second property. ■

If either T_1 or T_2 have no more than k vertices, then such a tree becomes a potential solution to be returned. On the other hand, if either tree spans more than k vertices, we need to argue that they contain a small subtree with large weight. The following lemma does exactly that.

Lemma 2. Let T be a tree having more than k vertices. Then there exists a subtree T with k vertices whose weight is at least $\frac{k}{2|V_T|} \cdot w(V_T)$.

Proof. We begin by doubling the edges in T and finding an Euler tour C in the resulting multi-graph. Let s denote some segment of C with k vertices. Notice that the vertices in s induce a subtree of T with at most k vertices.

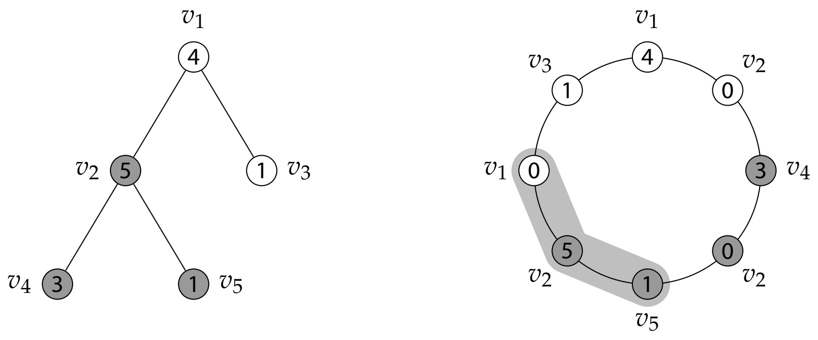


Figure 3.1: Tree T and its Euler tour C

For each vertex in $u \in V_T$, we pick one of its copies in C and assign to that copy w_u. It is not important which of the $\deg_T(u)$ copies of u in C we assign the weight to, but it is crucial that exactly one is used. Let $w(s)$ be the weight of segment s; that is, the weight assigned to the nodes in s. Notice that the vertices in s induce a subtree of T whose weight is at least $w(s)$. Hence, it suffices to show that there exists a segment s of C with k vertices whose weight is at least $\frac{k}{2|V_T|}w(V_T)$. Notice that we can start a segment at each of the $2|V_T|$ nodes in C. Using a simple averaging argument we get

$$\max_s w(s) \geq \frac{\sum_s w(s)}{2|V_T|} \geq \frac{\sum_{u \in V_T} k w_u}{2|V_T|} = \frac{k}{2|V_T|} w(V_T)$$

where the last inequality follows from the fact that each vertex $u \in V_T$ assigns its weight to some copy of itself in C, which appears in k of the $2|V_T|$ segments. ■

Using the results derived so far, we can now argue that the trees output by our algorithm have at least one fifth the weight of an optimal solution. The analysis is broken up into two key lemmas, which consider what happens when the tree T_1 has less or more than k vertices. Regarding T_2, we note that if T_2 has at most k vertices then T_2 is a feasible solution whose weight is $W_2 \geq \mathsf{OPT}$. Therefore, we assume from now on, without loss of generality, that T_2 spans more than k vertices.

Lemma 3. If T_1 has at most k vertices then the algorithm returns a tree whose weight is at least $\frac{\mathsf{OPT}}{5}$.

Proof. Since T_1 is small and T_2 is large, by Lemma 2, it suffices to show that

$$\max\left\{W_1, \frac{k}{2S_2}W_2\right\} \geq \frac{\mathsf{OPT}}{5}.$$

Applying Lemma 1, we get

$$\frac{k}{2S_2}W_2 \geq \frac{\alpha_2}{4}W_2 = \frac{\mathsf{OPT} - \alpha_1 W_1}{4} \geq \frac{\mathsf{OPT} - W_1}{4}.$$

Therefore,

$$\max\left\{W_1, \frac{k}{2S_2}W_2\right\} > \max\left\{W_1, \frac{\mathsf{OPT} - W_1}{4}\right\} \geq \frac{\mathsf{OPT}}{5},$$

which yields the desired guarantee. ■

Lemma 4. If T_1 has more than k vertices then the algorithm returns a tree whose weight is at least $\frac{\mathsf{OPT}}{4}$.

Proof. Since both T_1 and T_2 are large, by Lemma 2, it suffices to show that

$$\max\left\{\frac{k}{2S_1}W_1, \frac{k}{2S_2}W_2\right\} \geq \frac{\mathsf{OPT}}{4}.$$

Applying Lemma 1, we get

$$\frac{k}{2S_2}W_2 = \frac{k}{2\alpha_2 S_2}(\mathsf{OPT} - \alpha_1 W_1) \geq \frac{\mathsf{OPT} - \alpha_1 W_1}{2\left(2 - \alpha_1 \frac{S_1}{k}\right)}.$$

Notice that the expression the right hand side above is increasing in the interval $[0, \frac{2}{\beta})$, where $\beta = \frac{S_1}{k} > 1$. Therefore, since $\alpha_1 \in [0,1]$ and $\alpha_1\beta < 2$, the expression is minimized at $\alpha_1 = 0$ or $\alpha_1 = 1$. Putting everything together, we get

$$\begin{aligned}
\max\left\{\frac{k}{2S_1}W_1, \frac{k}{2S_2}W_2\right\} &\geq \max\left\{\frac{W_1}{2\beta}, \frac{\mathsf{OPT} - \alpha_1 W_1}{2(2-\alpha_1\beta)}\right\} \\
&\geq \max\left\{\frac{W_1}{2\beta}, \min\left\{\frac{\mathsf{OPT}}{4}, \frac{\mathsf{OPT} - W_1}{2(2-\beta)}\right\}\right\} \\
&\geq \min\left\{\frac{\mathsf{OPT}}{4}, \max\left\{\frac{W_1}{2\beta}, \frac{\mathsf{OPT} - W_1}{2(2-\beta)}\right\}\right\} \\
&\geq \frac{\mathsf{OPT}}{4}.
\end{aligned}$$

This finishes the proof. ■

Everything is in place to present the proof of the main result in the section.

Theorem 3. Given an instance (G, w, k, OPT) of the KCT problem, the algorithm HeavySubtree returns a tree T of at most k vertices such that $w(V_T) \geq \frac{\mathsf{OPT}}{5}$.

Proof. Right before the first iteration of the while loop we have

$$\lambda_2 - \lambda_1 = \frac{n}{w(V) - \mathsf{OPT}},$$

while right at the end

$$\lambda_2 - \lambda_1 < \frac{1}{w(V) - \mathsf{OPT}}.$$

In each iteration the value $\lambda_2 - \lambda_1$ is halved. Therefore, after $\log(n)$ iterations (and thus after that many calls to UnrootedGrowth) the algorithm terminates.
By Lemmas 3 and 4 it follows that the tree returned by the algorithm has weight at least $\frac{\mathsf{OPT}}{5}$. ■

3.4 Guessing the Optimal Weight

The last remaining issue is efficiently guessing OPT, the weight of the optimal solution. Let w^* denote the maximum weight of a node in the graph, $w^* := \max_{v \in V} w(v)$. The value OPT must then be contained in the interval $[w^*, kw^*]$. Our algorithm performs binary search over this interval. We introduce an additional parameter $\epsilon > 0$ to our algorithm and terminate the binary search when the final interval $[w_1, w_2]$ satisfies $w_2 - w_1 \leq \epsilon w^*$.

Algorithm 2: Bonsai(G, w, k, ϵ)

```
Data: Graph $G = (V, E)$, weight function $w : V \to \mathbb{R}_{\geq 0}$, cardinality
      $k \in \mathbb{N}$, error bound $0 < \epsilon < 1$
1  begin
2      $[w_1, w_2] \leftarrow [w^*, kw^*]$                         ▷ initial interval for OPT
3      while $w_2 - w_1 \geq \epsilon w^*$ do
4          $\mathsf{OPT}_\gamma \leftarrow \frac{w_2 + w_1}{2}$                  ▷ guessed value for OPT
5          $T \leftarrow \mathsf{HeavySubtree}(G, w, k, \mathsf{OPT}_\gamma)$
6          if $w(T) \geq \frac{\mathsf{OPT}_\gamma}{5}$ then
7              $w_1 \leftarrow \mathsf{OPT}_\gamma$
8          else
9              $w_2 \leftarrow \mathsf{OPT}_\gamma$
10     return $T$
```

We then run our algorithm a total of

$$\left\lceil \log \left(\frac{kw^* - w^*}{\epsilon w^*} \right) \right\rceil \leq \left\lceil \log \left(\frac{k}{\epsilon} \right) \right\rceil$$

times, thus *achieving independence from graph properties* like the number of nodes and edges and the maximum node weight w^*. Using this termination criterion will ensure that the last guessed value, OPT_γ, differs in the worst-case by a factor of $\frac{1}{1+\epsilon}$ from the true optimum:

$$\mathsf{OPT}_\gamma(1 + \epsilon) \geq \mathsf{OPT}_\gamma + \epsilon w^* \geq \mathsf{OPT} \tag{3.4}$$

for the last guessed optimum value OPT_γ (line 4 in Algorithm Bonsai) and the true optimum OPT. Note that the binary search interval can be narrowed down further, for example by computing the greedy solution of the problem and using its weight, w_{greedy} as the lower bound. In fact, we use this improvement in our implementation.

3.5 Complete Algorithm

We can now combine the existing parts to obtain the complete procedure – which is subsequently called Bonsai algorithm – in Algorithm 2.
The Bonsai algorithm satisfies Theorem 2 from Section 3.1, which is repeated here:

Theorem 2. There is an efficient algorithm that, given an instance (G, w, k) of the KCT problem, returns a tree T of at most k vertices such that $w(V_T) \geq \frac{\mathsf{OPT}}{5(1+\epsilon)}$ for any $\epsilon > 0$.

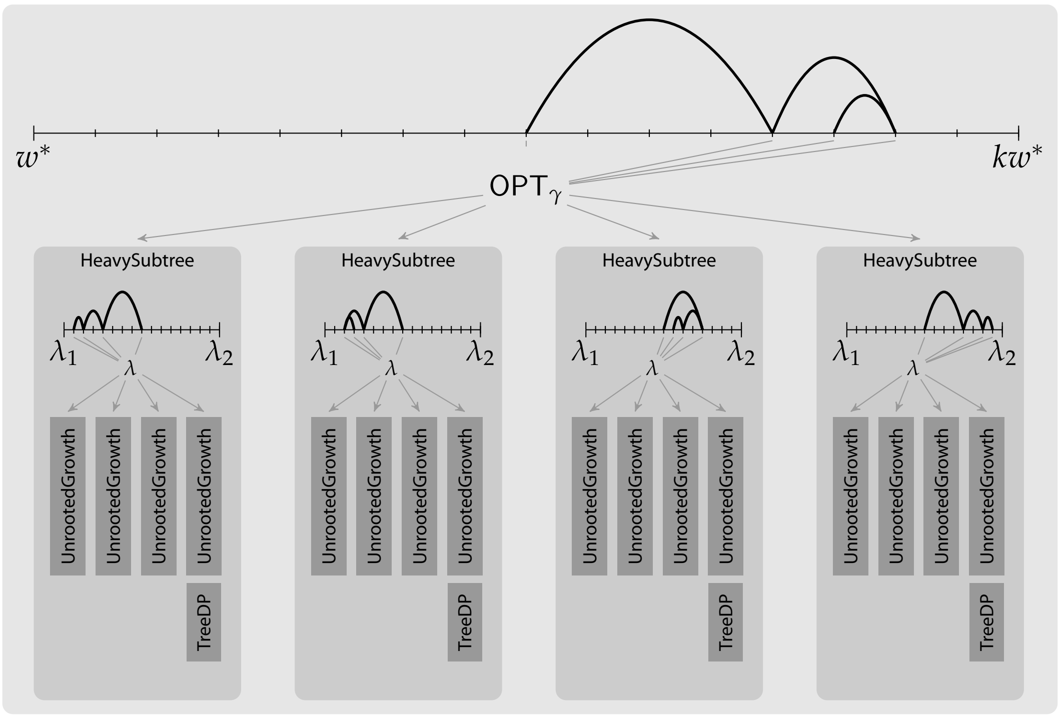


Figure 3.2: Execution schema of the Bonsai algorithm

Proof (Theorem 2). We guess the weight OPT of the optimal solution using the outer binary search procedure. For every guess, we run the HeavySubtree procedure which performs the inner binary search for the multiplication parameter λ. In each step of the inner binary search we try to obtain a tree with weight of at least OPT that is as small as possible, getting closer to this goal as the search progresses. For the last obtained trees, we retrieve the heaviest subtree T that satisfies our cardinality constraint, using the dynamic programming procedure TreeDP. Depending on the weight of tree T we proceed in the outer binary search procedure, increasing or reducing our guess for the optimum weight until the final interval is small enough. For the resulting tree returned in line 10 of Algorithm 2 we have (using Theorem 2):

$$w(T) \geq \frac{\mathsf{OPT}_\gamma}{5} \overset{(3.4)}{\geq} \frac{\mathsf{OPT}}{5(1+\epsilon)}. \tag{3.5}$$

■

A schematic overview of the algorithm is depicted in Figure 3.2. The impact of the cardinality constraint as well as the error bound on the required number of iterations of HeavySubtree is shown in Figure 3.3.

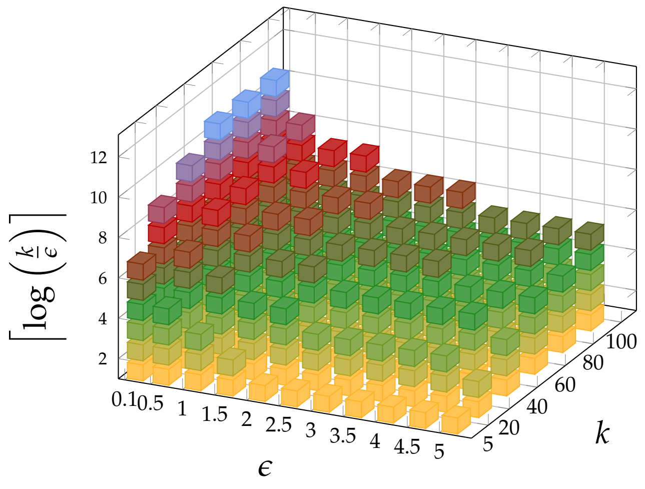


Figure 3.3: Impact of error bound and cardinality constraint on the maximum number of calls to HeavySubtree, which is upper-bounded by $\lceil \log(k/\epsilon) \rceil$

4 Experimental Evaluation

In this section, we provide the experimental evaluation of our algorithm. All experiments were conducted on Dell PowerEdge M610 servers, each of which has two Intel Xeon E5530 CPUs, 48 GB of main memory, a large iSCSI-attached disk array, and runs Debian GNU/Linux (SMP Kernel 2.6.29.3.1) as an operating system. Experiments were conducted using the Java Hotspot 64-Bit Server Virtual Machine (build 11.2-b01) installed on our servers. Note that our algorithm was implemented single-threaded.

4.1 Biological Networks

As a first example we present the results of our algorithm for a real-world graph. We run Bonsai on the protein-protein interaction network used by Dittrich et al. [9] for discovering functional modules. The node scores provided in this dataset are real numbers. Therefore, in order to execute our algorithm, we map the scores to non-negative values by adding to each score the minimum score in the network. The graph contains 2034 proteins (nodes) and 8399 interactions (edges).
Table 1 contains the experimental evaluation of this network for different cardinalities k and error bounds ϵ. Note that the implementation of our algorithm returns a first candidate solution after the first execution of the UnrootedGrowth procedure, followed by a call to the TreeDP routine. In the table, t_{final} denotes the total running time of the algorithm, t_{first} the time to return the first candidate solution and $w_{\text{first}}, w_{\text{final}}$ the weight of the first candidate tree and the weight of the final tree respectively.

k	ϵ	t_{first} [s]	t_{final} [s]	w_{first}	w_{final}	$w_{\text{first}}/w_{\text{final}}$
	0.1	0.004	10.100	40.9	52.5	0.78
5	0.5	0.003	6.503	40.9	52.5	0.78
	1.0	0.004	6.275	40.9	52.5	0.78
	0.1	0.005	8.389	163.2	185.2	0.88
20	0.5	0.005	6.427	163.2	185.2	0.88
	1.0	0.005	7.802	163.2	185.2	0.88
	0.1	0.009	15.588	642.6	726.0	0.89
100	0.5	0.010	11.369	642.6	726.0	0.89
	1.0	0.009	8.218	642.6	726.0	0.89

Table 4.1: Experimental results for the biological network

4.2 Synthetic Graphs

In the following, we demonstrate the running time and quality of our algorithm for synthetically created graphs. We execute the Bonsai(G, w, k, ϵ) algorithm over a wide variety of settings:

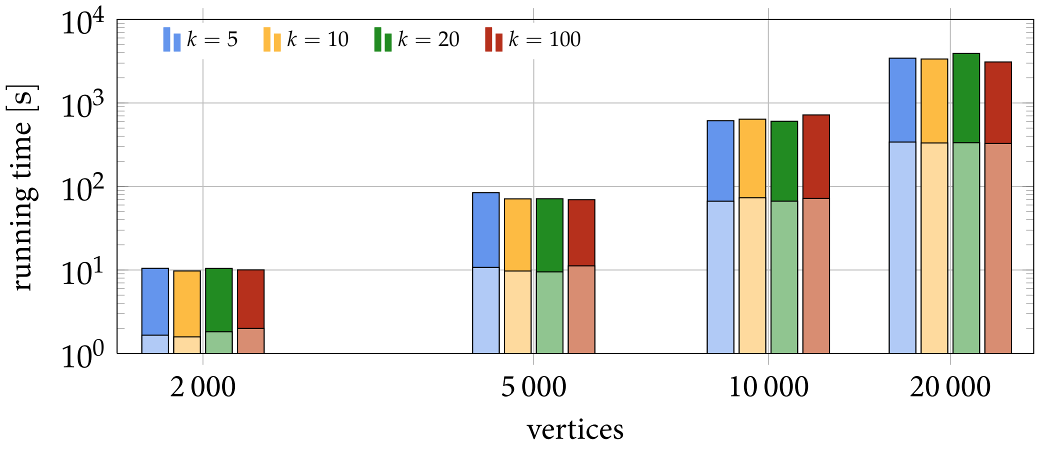
- as input graphs we generate power-law random graphs using the R-MAT graph generator[1] [6] with $n \in \{i \cdot 10^3 \mid i = 2, 5, 10, 20\}$ nodes and $m \in \{4n, 10n, 50n\}$ edges,
- a weight function $w : V \to \mathbb{R}_{\geq 0}$, that assigns power-law distributed values from the interval $[0, 1]$ to the nodes,
- cardinality constraints $k \in \{5, 10, 20, 100\}$, and
- error bound $\epsilon = 0.5$.

Figure 3 provides an overview over the resulting running times with an error bound of $\epsilon = 0.5$ and different graph sizes (nodes, edges) and cardinality values using a logarithmic scale. The lower part of each bar represents the required time for computing the first candidate solution (UnrootedGrowth followed by TreeDP). The full bar represents the total running time for the complete Bonsai algorithm.
The impact of the cardinality k is negligible in all the problem instances. This is due to the fact that we use the weight of the greedy solution as the lower bound for the value OPT, which is a much tighter bound than the maximal node weight.

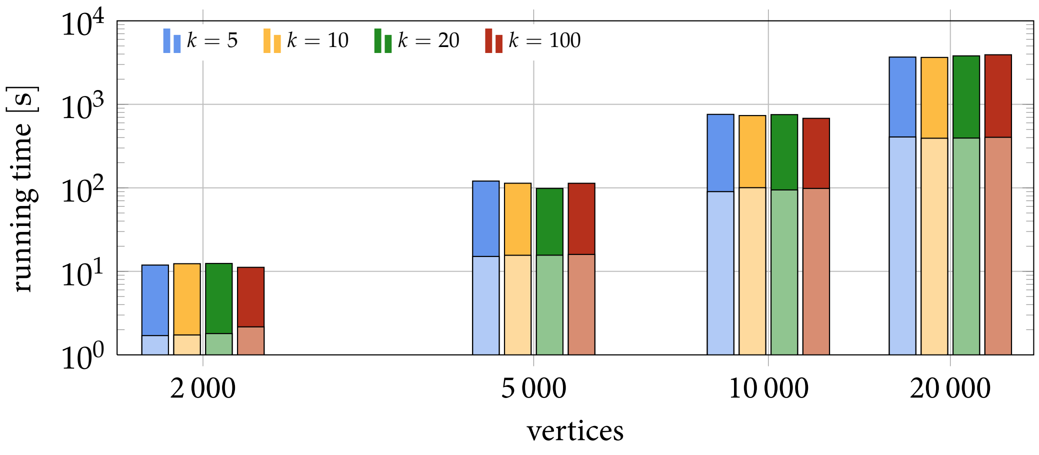
[1]with parameters $a = 0.45, b = 0.15, c = 0.15, d = 0.25$

In Figure 4 we compare the weight of the first returned candidate with the weight of final solution over a varying number of vertices and edges for different cardinalities k. It is striking that the difference between the weight of the first candidate tree (lower part of each bar) and the weight of the final output (full bar) is in all the cases very small, although the time to obtain it is almost an order of magnitude lower than the total running time of the algorithm. Note also that on average in all experiments we obtain a solution that is much better than the worst-case approximation guarantee, as – by the design of the experiments – the value OPT is upper-bounded by k.

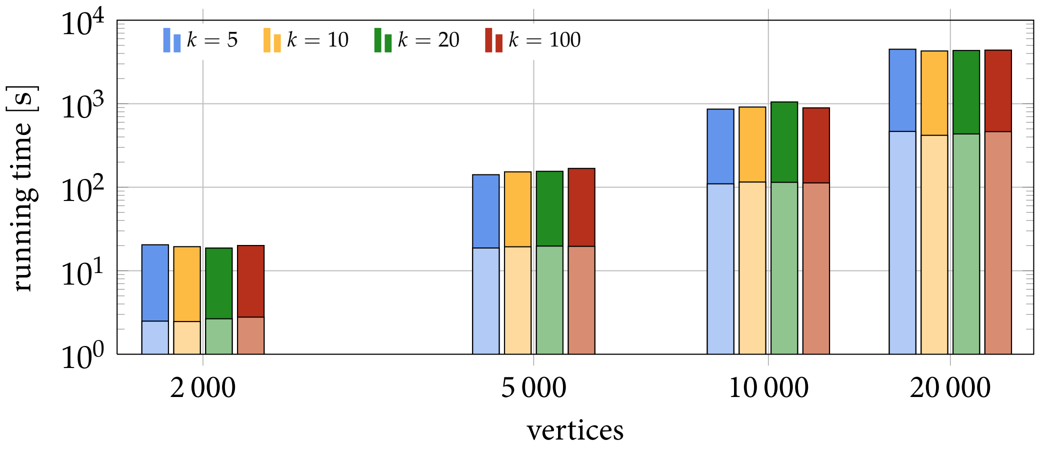
Figure 3: Running times (wall-clock) for various no. of vertices, edges, and cardinalities



(a) $m = 4n$ edges

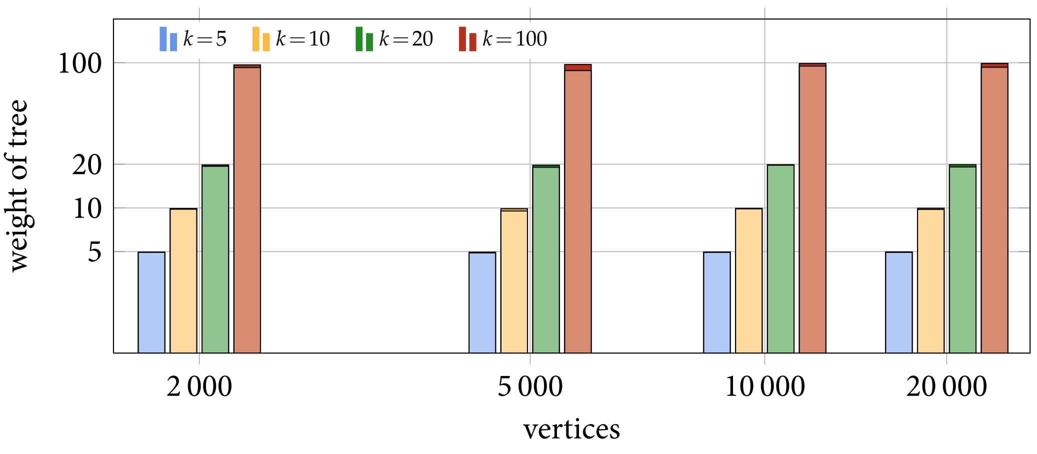


(b) $m = 10n$ edges

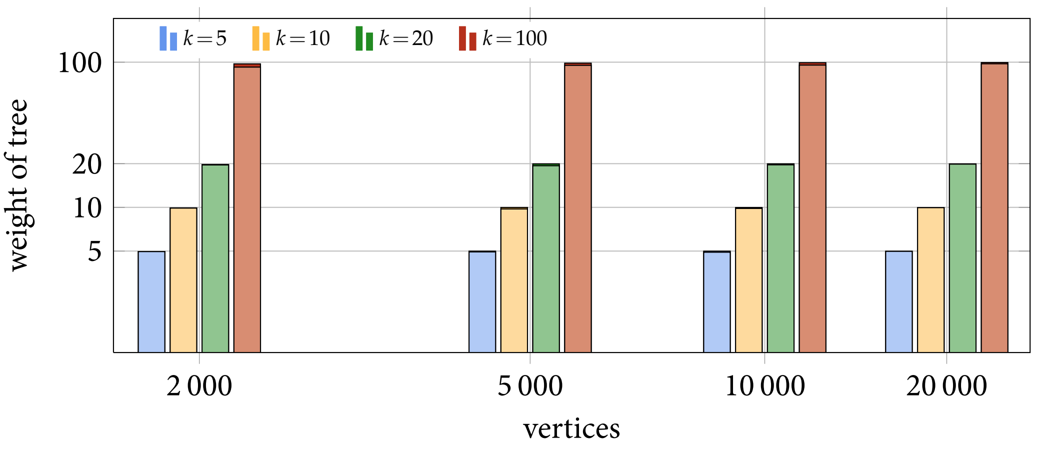


(c) $m = 50n$ edges

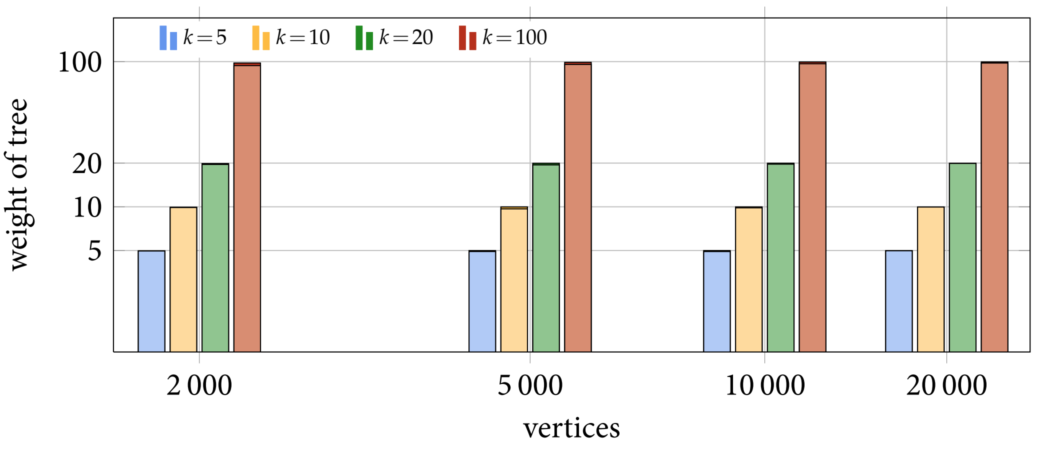
Figure 4: Solution quality for various number of vertices, edges, and cardinalities



(a) $m = 4n$ edges



(b) $m = 10n$ edges



(c) $m = 50n$ edges

5
Implementation Details

In this section, we briefly present the details of our implementation of Bonsai and its constituent subroutines. Our algorithms were implemented using Java 1.6. To the best of our knowledge, our algorithm is the *first ever practical implementation* of an constant-factor approximation algorithm for the KCT problem.
Our implementation of UnrootedGrowth essentially follows the ideas outlined in [21] and [10]. The key data structure used during execution of UnrootedGrowth is a collection of *Min-Heaps*, each of which corresponds to a component, i. e. a connected set of nodes – either *active* (the cluster of nodes that continues to grow), or *passive* (the cluster of nodes which have stopped growing). We implemented the Min-Heaps as FibonacciHeaps, which are known to be superior in performance [8].
Despite a highly optimized implementation of UnrootedGrowth, a naive implementation of Bonsai exhibits super-linear growth in execution time as we increase the size of the graph – which is not surprising in itself since UnrootedGrowth which we call poly-logarithmically many times, has complexity in $\mathcal{O}\big(n^2 \log(n)\big)$. As a result, we explored the potential of optimizing the number of probes (via both the outer and the inner binary search) we need to explicitly perform, results of which we present next:

5.1 Optimizing the Inner Binary Search

The key to optimizing the number of probes made in the inner binary-search lies in keeping track of the best results found in earlier iterations of outer binary-search, and using them to cut down the range of values to be considered as multiplication parameters λ. In other words, after the first complete execution of the HeavySubtree routine, we can improve the range of values $[\lambda_1, \lambda_2]$ that are explored in its subsequent executions. Consider an instance of the HeavySubtree routine for one of the choices, say OPT_i, of the outer binary search. Also, let the weights of the *final trees* obtained for already probed choices of OPT be $W_0, W_1, \ldots, W_{i-1}$, and corresponding values of λ be $l_0, l_1, \ldots, l_{i-1}$. If we keep track of the W_i and l_i values, we can tighten the range of values explored from both sides:

(i) **Improving λ_1.** If the weight of the final tree found in an earlier iteration is *smaller* than the value of OPT_i being used in the current execution of HeavySubtree, then we know that the best λ we can hope to find presently cannot be *smaller*. That is, if $\mathsf{OPT}_i \leq W_j$ for some $0 < j < i$, then l_i (i. e., the final value found by the current HeavySubtree instance) is strictly lower-bounded by l_j.

(ii) **Improving λ_2.** In every iteration of HeavySubtree, we can continue to upper-bound the value of λ_2 by the best value l_j found in earlier iteration, j, where $\mathsf{OPT}_i \geq W_j$.

It should be noted that the inner search optimization presented above does not lead to any loss in the quality of results found.

5.2 Early Termination of the Outer Binary Search

In the outer binary search, the goal is to keep improving the estimated OPT value which then is passed as an input to the instance of HeavySubtree. Clearly, these estimates keep improving with each iteration, since we update them using the weight of the final tree we obtain at each iteration. However, in practice we observed that after first few rounds of outer binary search, the weight of the final tree is already very close to the tree we eventually find. Thus, it may be possible to terminate the outer binary search early on, without introducing a significant error to the final solution of Bonsai. Another way of looking at this is to assume that we are given with a fairly weak ϵ value to begin with, which brings out another nice practical benefit of Bonsai, namely, its ability to incrementally improve the result quality. This feature enables users, especially in interactive environments such as visual analytics of graph data, to obtain initial coarse results very quickly and refine them if they feel the need to.

6
Related Work

The edge-weighted variant of the KCT problem, which is better known as the k-MST problem, has attracted more attention in literature than its vertex-weighted counterpart. In that setting, we are interested in the least-cost subtree spanning k vertices of a graph with non-negative edge costs $c : E \to \mathbb{R}_{\geq 0}$ and no vertex weights. A large body of literature exists on constant-factor approximation algorithms, the latest result being a 2-approximation devised by Garg [13]. All of theses approaches rely on the primal-dual method.
Johnson et al. [21] propose a way to derive an $\mathcal{O}(1)$-approximation algorithm for the k-cardinality tree problem by reducing it to the k-MST problem:
As a first step, the vertex-weighted graph G_v is transformed into an edge-weighted instance G_e: Every vertex v with weight w_v is being replaced by a root vertex r_v connected to $w_v - 1$ leaf nodes. The root is connected to each of the leaves via a zero-cost edge. For every edge $\{u, v\} \in G_v$ of the original graph, an edge $\{r_u, r_v\}$ with cost 1 is introduced between the respective root nodes. Then, by setting k to $2n \cdot \mathsf{OPT}$ (OPT again denoting the weight of the optimal solution to the KCT problem), we can run an existing k-MST algorithm based on the UnrootedGrowth subroutine on this new graph. This algorithm has pseudo-polynomial running time, however by implicitly merging of the zero-cost edges we can obtain a polynomial algorithm. The algorithm also contains the guessing step to obtain the optimum weight OPT. This transformation is approximation-guarantee preserving. By using the current-best approximation algorithm for k-MST, which is the 2-approximation devised by Garg [13] we thus obtain a 2-approximation algorithm for our original KCT problem. However, this algorithm will entail a much higher execution time, as the UnrootedGrowth subroutine must be executed polynomial many times whereas in our algorithm we execute this subroutine only polylogarithmic many times.

7 Conclusions

In this report we have provided a practical constant-factor approximation algorithm for the KCT problem, named Bonsai. Our algorithm works by reducing KCT to certain instances of the related PCST problem. We have exploited an existing approximation algorithm for this related setting and derived an algorithm with approximation guarantee of $\frac{1}{5(1+\epsilon)}$ for the KCT problem. Furthermore, we proposed various optimizations to our algorithm that lead to an implementation that is very flexible and runs reasonably fast on the problem instances considered. Using a mixture of synthetic and real-world graphs we were able to demonstrate the practical viability of our approach. The Bonsai algorithm can return a first candidate solution after the first execution of the PCST subroutine. We have shown empirically that the quality of this first solution is in all considered cases close to the optimum.

As we do not make any assumption on the distribution of the node weights, our algorithm is suitable for a variety of application scenarios. Possibilities include practical applications such as identifying the most-deviant parts of protein-protein interaction networks for designing biomedical trials, as well as others like using weights based on the structural properties of the graph (like node degrees or PageRank values) or interestingness-scores (like activity measures for articles in the Wikipedia graph to extract an active topical core). Interesting future work encompasses running our algorithm for these choices of weights and on graphs from various data sources. Other possible future directions include solving the KCT problem in the presence of both edge and node weights.

Bibliography

[1] E. Althaus, G. W. Klau, O. Kohlbacher, H.-P. Lenhof, and K. Reinert. Integer Linear Programming in Computational Biology. In *Efficient Algorithms*, volume 5760/2009 of *Lecture Notes in Computer Science*, pages 199–218. Springer, 2009. Cited on page 2.

[2] C. Blum. Revisiting Dynamic Programming for Finding Optimal Subtrees in Trees. *European Journal of Operational Research*, 177(1):102–114, 2007. Cited on page 8.

[3] C. Blum and M. J. Blesa. *Optimization Techniques for Solving Complex Problems*, chapter 23: Solving the KCT Problem: Large-Scale Neighborhood Search and Solution Merging, pages 407–421. Wiley Series on Parallel and Distributed Computing. Wiley, 2009. Cited on page 5.

[4] C. Blum and M. Ehrgott. Local Search Algorithms for the k-Cardinality Tree Problem. *Discrete Applied Mathematics*, 128(2-3):511–540, 2003. Cited on page 5.

[5] J. Brimberg, D. Urošević, and N. Mladenović. Variable Neighborhood Search for the Vertex Weighted k-Cardinality Tree Problem. *European Journal of Operational Research*, 171(1):74–84, 2006. Cited on page 5.

[6] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *SDM'04: Proceedings of the 2004 SIAM International Conference on Data Mining*, 2004. Cited on page 17.

[7] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-MSTs and k-Steiner Trees via the Primal-Dual Method and Lagrangean Relaxation. *Mathematical Programming*, 100(2):411–421, 2004. Cited on page 7.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, third edition, 2009. Cited on page 21.

[9] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. Identifying Functional Modules in Protein-Protein Interaction Networks: An Integrated Exact Approach. *Bioinformatics*, 24(13):i223–i231, 2008. 2 citations on pages 2 and 16.

[10] P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina. $\mathcal{O}(n^2 \log(n))$ Implementation of an Approximation for the Prize-Collecting Steiner Tree Problem. 2002. Cited on page 21.

[11] P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina. Primal-Dual Approximation Algorithms for the Prize-Collecting Steiner Tree Problem. *Information Processing Letters*, 103(5):195–202, 2007. Cited on page 8.

[12] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli. Weighted k-Cardinality Trees: Complexity and Polyhedral Structure. *Networks*, 24:11–21, 1994. Cited on page 4.

[13] N. Garg. Saving an ϵ: A 2-Approximation for the k-MST Problem in Graphs. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 396–402. ACM, 2005. Cited on page 23.

[14] D. Gibson, R. Kumar, and A. Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *VLDB'05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 721–732. VLDB Endowment, 2005. Cited on page 2.

[15] M. X. Goemans and D. P. Williamson. A General Approximation Technique for Constrained Forest Problems. In *SODA'92: Proceedings of the third annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316, 1992. 2 citations on pages 5 and 6.

[16] H. He and A. K. Singh. Efficient Algorithms for Mining Significant Substructures in Graphs with Quality Guarantees. In *ICDM'07: Proceedings of the 7th IEEE Conference on Data Mining*, pages 163–172. IEEE Computer Society, 2007. Cited on page 2.

[17] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD'07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 305–316. ACM, 2007. Cited on page 2.

[18] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining Coherent Dense Subgraphs Across Massive Biological Networks for Functional Discovery. *Bioinformatics*, 21(S1):213–221, 2005. Cited on page 2.

[19] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. In *ICDM'03: Proceedings of the 3th IEEE Conference on Data Mining*. IEEE Computer Society, 2003. Cited on page 2.

[20] K. Jain and V. V. Vazirani. Approximation Algorithms for Metric Facility Location and k-Median Problems using the Primal-Dual Schema and Lagrangian Relaxation. *Journal of the ACM*, 48(2):274–296, 2001. Cited on page 7.

[21] D. S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. In *SODA'00: Proceeding of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000. 5 citations on pages 5, 6, 8, 21, and 23.

[22] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph Summarization with Bounded Error. In *SIGMOD'08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 419–432. ACM, 2008. Cited on page 2.

[23] A. Segev. The Node-Weighted Steiner Tree Problem. *Networks*, 17:1–17, 1987. Cited on page 5.

[24] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In *ICDE'09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 405–416. IEEE Computer Society, 2009. Cited on page 2.

[25] N. Wang, S. Parthasarathy, K.-L. Tan, and A. K. H. Tung. CSV: Visualizing and Mining Cohesive Subgraphs. In *SIGMOD'08: Proceedings of the 2008 ACM SIGMOD Intl. Conference on Management of Data*, pages 445–458. ACM, 2008. Cited on page 2.

MPI-I-2010-RG1-001	M. Suda, C. Weidenbach, P. Wischnewski	On the saturation of YAGO
MPI-I-2010-5-006	A. Broschart, R. Schenkel	Real-time text queries with tunable term pair indexes
MPI-I-2010-5-002	M. Theobald, M. Sozio, F. Suchanek, N. Nakashole	URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules
MPI-I-2010-5-001	K. Berberich, S. Bedathur, O. Alonso, G. Weikum	A language modeling approach for temporal information needs
MPI-I-2009-RG1-005	M. Horbach, C. Weidenbach	Superposition for fixed domains
MPI-I-2009-RG1-004	M. Horbach, C. Weidenbach	Decidability results for saturation-based model building
MPI-I-2009-RG1-002	P. Wischnewski, C. Weidenbach	Contextual rewriting
MPI-I-2009-RG1-001	M. Horbach, C. Weidenbach	Deciding the inductive validity of $\forall\exists^*$ queries
MPI-I-2009-5-007	G. Kasneci, G. Weikum, S. Elbassuoni	MING: Mining Informative Entity-Relationship Subgraphs
MPI-I-2009-5-006	S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, G. Weikum	Scalable phrase mining for ad-hoc text analytics
MPI-I-2009-5-005	G. de Melo, G. Weikum	Towards a Universal Wordnet by learning from combined evidenc
MPI-I-2009-5-004	N. Preda, F.M. Suchanek, G. Kasneci, T. Neumann, G. Weikum	Coupling knowledge bases and web services for active knowledge
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-002	M. Ramanath, K.S. Kumar, G. Ifrim	Generating concise and readable summaries of XML documents
MPI-I-2009-4-006	C. Stoll	Optical reconstruction of detailed animatable human body models
MPI-I-2009-4-005	A. Berner, M. Bokeloh, M. Wand, A. Schilling, H. Seidel	Generalized intrinsic symmetry detection
MPI-I-2009-4-004	V. Havran, J. Zajac, J. Drahokoupil, H. Seidel	MPI Informatics building model as data for your research
MPI-I-2009-4-003	M. Fuchs, T. Chen, O. Wang, R. Raskar, H.P.A. Lensch, H. Seidel	A shaped temporal filter camera
MPI-I-2009-4-002	A. Tevs, M. Wand, I. Ihrke, H. Seidel	A Bayesian approach to manifold topology reconstruction
MPI-I-2009-4-001	M.B. Hullin, B. Ajdin, J. Hanika, H. Seidel, J. Kautz, H.P.A. Lensch	Acquisition and analysis of bispectral bidirectional reflectance distribution functions
MPI-I-2008-RG1-001	A. Fietzke, C. Weidenbach	Labelled splitting
MPI-I-2008-5-004	F. Suchanek, M. Sozio, G. Weikum	SOFI: a self-organizing framework for information extraction
MPI-I-2008-5-003	G. de Melo, F.M. Suchanek, A. Pease	Integrating Yago into the suggested upper merged ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single phase construction of optimal DAG-structured QEPs
MPI-I-2008-5-001	G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, G. Weikum	STAR: Steiner tree approximation in relationship-graphs
MPI-I-2008-4-003	T. Schultz, H. Theisel, H. Seidel	Crease surfaces: from theory to extraction and application to diffusion tensor MRI
MPI-I-2008-4-002	D. Wang, A. Belyaev, W. Saleem, H. Seidel	Estimating complexity of 3D shapes using view similarity
MPI-I-2008-1-001	D. Ajwani, I. Malinger, U. Meyer, S. Toledo	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for finite domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : a large ontology from Wikipedia and WordNet

MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A time machine for text search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: searching and ranking knowledge
MPI-I-2007-4-008	J. Gall, T. Brox, B. Rosenhahn, H. Seidel	Global stochastic optimization for robust and accurate human motion capture
MPI-I-2007-4-007	R. Herzog, V. Havran, K. Myszkowski, H. Seidel	Global illumination using photon ray splatting
MPI-I-2007-4-006	C. Dyken, G. Ziegler, C. Theobalt, H. Seidel	GPU marching cubes on shader model 3.0 and 4.0
MPI-I-2007-4-005	T. Schultz, J. Weickert, H. Seidel	A higher-order structure tensor
MPI-I-2007-4-004	C. Stoll, E. de Aguiar, C. Theobalt, H. Seidel	A volumetric approach to interactive shape editing
MPI-I-2007-4-003	R. Bargmann, V. Blanz, H. Seidel	A nonlinear viseme model for triphone-based speech synthesis
MPI-I-2007-4-002	T. Langer, H. Seidel	Construction of smooth maps with mean value coordinates
MPI-I-2007-4-001	J. Gall, B. Rosenhahn, H. Seidel	Clustered stochastic optimization for object recognition and pose estimation
MPI-I-2007-2-001	A. Podelski, S. Wagner	A method and a tool for automatic veriication of region stability for hybrid systems
MPI-I-2007-1-003	A. Gidenstam, M. Papatriantafilou	LFthreads: a lock-free thread library
MPI-I-2007-1-002	E. Althaus, S. Canzar	A Lagrangian relaxation approach for the multiple sequence alignment problem
MPI-I-2007-1-001	E. Berberich, L. Kettner	Linear-time reordering in a sweep-line algorithm for algebraic curves intersecting in a common point
MPI-I-2006-5-006	G. Kasnec, F.M. Suchanek, G. Weikum	Yago - a core of semantic knowledge
MPI-I-2006-5-005	R. Angelova, S. Siersdorfer	A neighborhood-based approach for clustering of linked document collections
MPI-I-2006-5-004	F. Suchanek, G. Ifrim, G. Weikum	Combining linguistic and statistical analysis to extract relations from web documents
MPI-I-2006-5-003	V. Scholz, M. Magnor	Garment texture editing in monocular video sequences based on color-coded printing patterns
MPI-I-2006-5-002	H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum	IO-Top-k: index-access optimized top-k query processing
MPI-I-2006-5-001	M. Bender, S. Michel, G. Weikum, P. Triantafilou	Overlap-aware global df estimation in distributed information retrieval systems
MPI-I-2006-4-010	A. Belyaev, T. Langer, H. Seidel	Mean value coordinates for arbitrary spherical polygons and polyhedra in $\mathbb{R}^3$
MPI-I-2006-4-009	J. Gall, J. Potthoff, B. Rosenhahn, C. Schnoerr, H. Seidel	Interacting and annealing particle filters: mathematics and a recipe for applications
MPI-I-2006-4-008	I. Albrecht, M. Kipp, M. Neff, H. Seidel	Gesture modeling and animation by imitation
MPI-I-2006-4-007	O. Schall, A. Belyaev, H. Seidel	Feature-preserving non-local denoising of static and time-varying range data
MPI-I-2006-4-006	C. Theobalt, N. Ahmed, H. Lensch, M. Magnor, H. Seidel	Enhanced dynamic reflectometry for relightable free-viewpoint video