Hunting Multiple Bumps in Graphs

Yahui Sun[1], Jun Luo[2], Theodoros Lappas[3], Xiaokui Xiao[4], Bin Cui[5]
[1,2]School of Computer Science and Engineering, Nanyang Technological University
[3]School of Business, Stevens Institute of Technology
[4]School of Computing, National University of Singapore
[5]School of EECS & Key Laboratory of High Confidence Software Technologies (MOE), Peking University
yahui.sun, junluo@ntu.edu.sg; tlappas@stevens.edu; xkxiao@nus.edu.sg; bin.cui@pku.edu.cn

ABSTRACT

Bump hunting is an important approach to the extraction of insights from Euclidean datasets. Recently, it has been explored for graph datasets for the first time, and a single bump is hunted in an unweighted graph in this exploration. Here, we extend this exploration by hunting multiple bumps in a weighted graph. Given a weighted graph and a set of *query* nodes exhibiting a property of interest, our objective is to find k non-overlapping and connected subgraphs, i.e., bumps, in which the discrepancy between the numbers of query and non-query nodes is maximized and the sum of edge costs is minimized simultaneously.

We prove that our extended bump hunting problem can be transformed to a recently formulated Prize-Collecting Steiner Forest Problem (PCSFP). We further prove that PCSFP is NP-hard even in trees. Then, we propose a fast approximation algorithm for solving PCSFP in trees. Based on this algorithm, we improve the state-of-the-art approximation algorithm for solving PCSFP in graphs, and prove that the solutions of our improvement are always better than or equal to those of the state-of-the-art algorithm. Moreover, we adapt the existing bump hunting algorithms for solving our extended bump hunting problem.

We evaluate our methodology via real datasets, and show that 1) our improvement scales well to large graphs, while producing solutions that dominate those of the state-of-the-art algorithm; and 2) our adaptation of an existing bump hunting algorithm can also produce solutions that are better than those of the state-of-the-art algorithm in some cases.

PVLDB Reference Format:
Yahui Sun, Jun Luo, Theodoros Lappas, Xiaokui Xiao, and Bin Cui. Hunting Multiple Bumps in Graphs. *PVLDB*, 13(5): 656-669, 2020.
DOI: https://doi.org/10.14778/3377369.3377375

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 13, No. 5
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3377369.3377375

Bump hunting is a data analysis approach that has been continuously studied in the last few decades [3, 14, 19]. The main idea is to search for regions of a dataset where a property of interest occurs frequently. Traditional bump hunting methods apply geometric knowledge to find such regions of Euclidean datasets. These methods do not suit graph datasets, such as social networks and knowledge graphs. To address this issue, Gionis *et al.* [12] recently explored the bump hunting approach for graph datasets for the first time. They divided nodes in a graph into two groups: *query* nodes that exhibit a property of interest, and *non-query* nodes that do not exhibit a property of interest. Given a graph and a set of query nodes, their objective was to find a connected subgraph with the maximum discrepancy between the numbers of query and non-query nodes. As Gionis *et al.* pointed out, this discrepancy maximization objective is different from the objectives of the other graph mining approaches (e.g. finding high-modularity graph divisions for community detection [28, 5]), and enables us to find a *region* of the graph dataset where the property of interest occurs frequently, i.e., nodes are often query nodes. We describe two application scenarios as follows:

Scenario 1 [12]: Given a social network where vertices and edges represent persons and personal interactions respectively; and a set of query nodes representing persons who exhibit a property of interest, we can hunt bumps that represent communities of this property of interest. For example, given the DBLP network where vertices and edges represent researchers and collaborations between researchers respectively; and a set of query nodes representing researchers who have publications in an area, we can hunt bumps that represent communities of researchers in this area (see Figure 1; details in Section 5.1).

Scenario 2 [32]: Given an activity network where vertices and edges represent road intersections and roads respectively; and a set of query nodes representing road intersections near which high-level activities are detected, we can hunt bumps that represent regions that exhibit high levels of activity. For example, given the Twitter activity network [29]; and a set of query nodes representing road intersections near which abundant geo-located Twitter posts are detected, we can hunt bumps that represent regions that exhibit high levels of Twitter activity (details in Section 5.2).

Motivation: In the exploratory work of Gionis *et al.*, a single bump is hunted in a graph. However, it may be prefer-

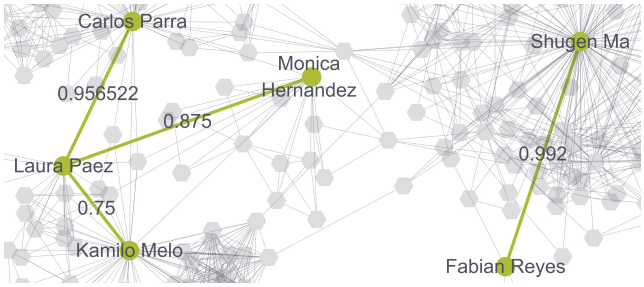


Figure 1: The DBLP network in Section 5.1, where vertices represent researchers; edges represent collaborations between researchers; and edge costs are Jaccard distances that indicate the dis-closeness between researchers. For a query: "snake robots", green vertices represent researchers who have related publications, and green edges highlight bumps that represent communities of these researchers.

able to hunt multiple bumps in a graph in many cases. For example, in the above Scenario 1, finding multiple communities of researchers via DBLP helps analyze the development of a queried research area statistically; and in the above Scenario 2, finding multiple regions that exhibit high levels of Twitter activity helps analyze the behaviors of Twitter users statistically. Moreover, edges are unweighted in this exploratory work, while edges are often weighted in real graph datasets, and it may be preferable to hunt bumps in such weighted graphs in many cases. For example, in the above Scenario 1, edges are often associated with costs that indicate the dis-closeness between researchers (e.g. Jaccard distances [25]), and it may be preferable to minimize such costs in bumps for finding close communities of researchers [25]; and in the above Scenario 2, edges are often associated with costs that indicate road distances, and it may be preferable to minimize such costs in bumps for finding regions that exhibit high levels of close activity, which often indicate events in the physical world [32].

The above issues can be addressed by hunting multiple bumps in a weighted graph, yet it is highly non-trivial to do this. Specifically, we will later prove that it is NP-hard to do this even when the graph is a tree, and the existing bump hunting algorithms developed by Gionis *et al.* can be ineffective for several reasons. First, the TreeOptimal algorithm [12], which is the core of these existing algorithms, requires a given bump root that may not be available in many cases. Second, these existing algorithms do not consider edge costs in graphs. Third, these existing algorithms keep hunting a single bump with a high discrepancy between the numbers of query and non-query nodes, and cannot be modified easily for hunting multiple bumps in an efficient way.

We will later prove that hunting multiple bumps in a weighted graph can be transformed to solving Hegde *et al.*'s Prize-Collecting Steiner Forest Problem (PCSFP) [21], which was formulated recently for graph-structured sparsity. To the best of our knowledge, Hegde *et al.*'s algorithm [21], which is based on their fast implementation of the Goemans-Williamson scheme [20] in the latest DIMACS Implementation Challenge on Steiner tree problems [1] , is the only existing and thus the state-of-the-art algorithm for solving PCSFP to date. We refer to this algorithm as FGWA (Fast Goemans-Williamson Algorithm). We observe that it may not be able to produce solutions with satisfactorily high qualities in some cases. The reason is that, given a weighted graph, it first clusters vertices into multiple trees, and then hunts the optimal bump in each of these trees, while missing sub-optimal bumps in some trees that are possibly better than the optimal bumps in some other trees.

Our contributions: To address the above issues, we make the following contributions in this paper.

- We formulate the Bump Hunting Problem in Graphs (BHPG). Our formulation extends the existing one by hunting multiple bumps in a weighted graph.
- We prove that BHPG can be transformed to PCSFP. We further prove that PCSFP is NP-hard even in trees. Given this hardness result, we propose a fast approximation algorithm for solving PCSFP in trees.
- Based on our algorithm in trees, we improve FGWA for solving PCSFP in graphs, and prove that the solutions of our improvement are always better than or equal to those of FGWA. Our improvement suits not only bump hunting, but also other applications that require solving PCSFP (e.g. graph-structured sparsity [21]).
- Based on our algorithm in trees, we adapt the existing bump hunting algorithms for solving BHPG.
- We evaluate our methodology via real datasets[1], and show that 1) our improvement scales well to large graphs, while producing solutions that dominate those of FGWA; and 2) our adaptation of an existing bump hunting algorithm can also produce solutions that are better than those of FGWA in some cases. We further show that two existing bump hunting applications can be extended by hunting multiple bumps in graphs.

Roadmap: The rest of this paper is organized as follows: in Section 2, we formulate BHPG, and describe our solutions for improving the readability of this paper; in Section 3, we propose our methodology; in Section 4, we evaluate our methodology; in Section 5, we extend two existing bump hunting applications; in Section 6, we review the related work; and ultimately in Section 7, we conclude this paper.

2. PRELIMINARIES

In this section, we first formulate BHPG, then present the Prize-Collecting Steiner Forest solution approach, and ultimately describe our solutions in brief.

2.1 Problem formulation

We consider an undirected (connected or disconnected) graph $G(V, E, c)$, where V is the set of vertices, E is the set of edges, and c is a function which maps each edge $e \in E$ to a positive value $c(e)$ that we refer to as edge cost. A set of *query* nodes $Q \subseteq V$ that exhibit a property of interest is provided as the input. We refer to a connected subgraph of G as a *component* of G. We consider a set of components of G as *non-overlapping* if they do not share vertices with each other. Let $C(V_C, E_C)$ be a component of G. We refer to p_C as the number of query nodes in C, i.e., $p_C = |V_C \cap Q|$. We refer to n_C as the number of non-query nodes in C, i.e., $n_C = |V_C \setminus Q|$. We define the discrepancy of C in the same way as the exploratory work [12] as follows.

[1]The codes and datasets are available at `https://github.com/YahuiSun/bump_hunting`

DEFINITION 1 (DISCREPANCY [12]). *Given a component $C(V_C, E_C)$ of a graph $G(V, E, c)$; and a set of query nodes $Q \subseteq V$, the discrepancy of C is*

$$g(C) = \alpha p_C - n_C \quad (1)$$

where α is the regulating weight between the numbers of query and non-query nodes in C, and $\alpha > 0$.

High-discrepancy components are regions of G where the property of interest occurs *frequently*, i.e., nodes are often query nodes. Finding these regions is useful in practice. For example, when one submits a query to DBLP's search engine, DBLP returns a list of researchers who have related publications. If we use query nodes to represent the returned researchers for a query, then high-discrepancy components of the DBLP network are communities of the returned researchers (see Figure 1).

Furthermore, high-discrepancy components with small edge costs inside are regions of G where the property of interest occurs both frequently and *closely*. Finding these regions is also useful in practice. For example, edges in the DBLP network are often associated with costs that indicate the discloseness between researchers (e.g. Jaccard distances [25]), and high-discrepancy components with small edge costs inside are close communities of the returned researchers.

We consider high-discrepancy components with small edge costs inside as *bumps*. Hunting bumps is then to find regions of G where the property of interest occurs both frequently and closely. To hunt such bumps, we refer to c_C as the sum of edge costs in C, i.e., $c_C = \sum_{e \in E_C} c(e)$, and consider the quality of C as the difference between $g(C)$ and c_C. This consideration enables us to add a possibly-zero regulating weight on c_C, and echoes the related graph mining approaches that minimize the sums of edge costs in the identified subgraphs (e.g. [25, 32]). Specifically, we define the quality of C as follows.

DEFINITION 2 (QUALITY). *Given a component $C(V_C, E_C)$ of a graph $G(V, E, c)$; and a set of query nodes $Q \subseteq V$, the quality of C is*

$$Qua(C) = g(C) - \beta c_C \quad (2)$$

where β is the regulating weight between the discrepancy of C and the sum of edge costs in C, and $\beta \geq 0$.

There are often multiple bumps in a real graph dataset. We rank these bumps with respect to their qualities. Our objective is to find the top-k non-overlapping bumps. This non-overlapping requirement is necessary, as otherwise we may hunt bumps with the same set of vertices, which should be avoided in most scenarios. Specifically, we define the Bump Hunting Problem in Graphs (BHPG) as follows.

PROBLEM 1 (BHPG). *Given a graph $G(V, E, c)$; a set of query nodes $Q \subseteq V$; and a target number $k \in \mathbb{N}$, the Bump Hunting Problem in Graphs is to find the top-k non-overlapping bumps in G: $\cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$ such that $\sum_{m=1}^{k} Qua(C_m)$ is maximized.*

Solving BHPG is useful in various scenarios. For example, given the DBLP network; a set of query nodes representing researchers in an area; and a target number k, we can solve BHPG for finding the top-k communities of researchers in this area (see Figure 1; details in Section 5.1).

Since edge costs in the hunted bumps are minimized when $\beta > 0$, the optimal solution to BHPG is a set of k non-overlapping trees when $\beta > 0$. It is preferable to restrict the hunted bumps to trees in scenarios where edge costs are minimized (e.g. [25, 32]), but not in scenarios where edge costs are not minimized (e.g. [12]). The exploratory formulation [12] does not restrict the hunted bumps to trees. By setting $\beta = 0$, BHPG does not restrict the hunted bumps to trees as well, given that every component C_m with the same set of vertices V_{C_m} has the same discrepancy $g(C_m)$. Specifically, we define BHPG in such a way that it degenerates to the exploratory formulation [12] when $k = 1$ and $\beta = 0$. Since the exploratory formulation is NP-hard, BHPG is NP-hard. Moreover, we will later prove that BHPG is NP-hard even when 1) G is a set of non-overlapping trees; and 2) $\beta = 0$.

2.2 The Steiner forest solution approach

A set of non-overlapping trees is usually referred to as a *forest*. We observe that BHPG can be transformed to Hegde *et al.*'s Prize-Collecting Steiner Forest Problem (PCSFP) [21]. In PCSFP, we consider an undirected graph $G(V, E, w, c)$, where w is a function which maps each vertex $i \in V$ to a non-negative value $w(i)$ that we refer to as vertex prize. We present the definition of PCSFP as follows.

PROBLEM 2 (PCSFP). *Given a graph $G(V, E, w, c)$; and a target number $k \in \mathbb{N}$, the Prize-Collecting Steiner Forest Problem is to find k non-overlapping trees of G: $C_{sum}(V_{sum}, E_{sum}) = \cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$ such that the net-cost of these trees:*

$$c(C_{sum}) = \sum_{v \in V \setminus V_{sum}} w(v) + \sum_{e \in E_{sum}} c(e) \quad (3)$$

is minimized, or the net-weight of these trees:

$$w(C_{sum}) = \sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) \quad (4)$$

is maximized.

Since $\sum_{v \in V \setminus V_{sum}} w(v) + \sum_{v \in V_{sum}} w(v)$ is constant for a given graph, the above two objectives are equivalent. Hegde *et al.*'s PCSFP is a unique problem that is different from previous problems with the same or similar names (e.g. [15, 16, 35])[2]. We show the transformation from BHPG to Hegde *et al.*'s PCSFP below, for which the proof is in the appendix.

THEOREM 1. *Given two graphs $G(V, E, c)$ and $G'(V, E, w', c')$; a set of query nodes $Q \subseteq V$; and a target number $k \in \mathbb{N}$, if*

$$w'(v) = \begin{cases} \alpha + 1, & v \in V \cap Q \\ 0, & v \in V \setminus Q \end{cases} \quad (5)$$

$$c'(e) = \beta c(e) + 1, \quad e \in E \quad (6)$$

then the optimal solution to BHPG in G and the optimal solution to PCSFP in G' have the same set of vertices despite β, and further have the same set of edges when $\beta > 0$.

[2]In these previous problems, pairs of vertices are given, and the objective is to find a forest to minimize the prizes of the given pairs of vertices not connected by this forest plus the edge costs in this forest. We observe that, when pairs of vertices are copies of vertices (e.g. $\{v, v\}$), this objective is to find an unlimited number of non-overlapping trees to minimize $c(C_{sum})$, for which the optimal solution is simply the set of vertices with positive prizes.

This theorem shows that we can solve BHPG in G by solving PCSFP in G'. We will later develop Prize-Collecting Steiner Forest algorithms for solving BHPG.

2.3 The solution overview

The existing solutions: Hegde *et al.*'s FGWA [21] is the state-of-the-art algorithm for solving PCSFP. In FGWA, the fast implementation of the Goemans-Williamson growing scheme [20] is first used to cluster vertices into k non-overlapping trees, and the Goemans-Williamson pruning algorithm [13] or the strong pruning algorithm [23] is then used to find the maximum-net-weight subtree in each of these trees. The combination of these maximum-net-weight subtrees is the solution of FGWA. We show this problem-solving process via the following example.

Applying Hegde et al.'s FGWA to solve PCSFP ($k = 2$) in the graph in Figure 2: First, the fast implementation of the Goemans-Williamson growing scheme returns the two trees in this graph (details in [21]). Then, the Goemans-Williamson pruning algorithm or the strong pruning algorithm returns the maximum-net-weight subtrees in these two trees: $\{(v_1, v_2)\}$ and $\{(v_6, v_7)\}$, of which the net-weights are $2 + 2 - 1.5 = 2.5$ and $2 + 2 - 1.7 = 2.3$ respectively. The solution of FGWA is the combination of these two subtrees, of which the net-weight is $2.5 + 2.3 = 4.8$.

Hegde *et al.*'s FGWA has a tight approximation guarantee of 2 with respect to minimizing $c(C_{sum})$, and a nearly-linear time complexity of $O(d|E|log|V|+|V|)$, where d is the precision of prizes and costs, and $O(d|E|log|V|)$ and $O(|V|)$ correspond to the above growing and pruning processes respectively. We observe that the solutions of Hegde *et al.*'s FGWA can be improved. The reason is that, given a graph, it first clusters vertices into k non-overlapping trees, and then combines the maximum-net-weight subtree in each of these trees as the solution, while missing non-maximum-net-weight subtrees in some trees that may have larger net-weights than the maximum-net-weight subtrees in some other trees. For example, in Figure 2, it misses the non-maximum-net-weight subtree $\{(v_4, v_5)\}$ in the left tree, which has a larger net-weight of $2 + 2 - 1.6 = 2.4$ than the maximum-net-weight subtree $\{(v_6, v_7)\}$ in the right tree.

Our improved solutions: To address the above issue, we develop the following improved solutions in this paper.

Phase 1: Given a graph, we first cluster vertices into a set of non-overlapping trees. This can be done via the Goemans-Williamson growing scheme, like Hegde *et al.*'s FGWA. For example, given the graph in Figure 2, we can employ the fast implementation of the Goemans-Williamson growing scheme, which returns the two trees in this graph.

Phase 2: Different from Hegde *et al.*'s pruning method of combining the maximum-net-weight subtree in each tree, we propose a more effective pruning method for hunting bumps from the set of non-overlapping trees obtained via the above phase (see our ABHA in Section 3.2). We briefly introduce this method as follows. We employ a max priority queue to store candidate subtrees. Initially, we push the maximum-net-weight subtree in each tree into the queue with the priorities of their net-weights. For example, we push $\{(v_1, v_2)\}$ and $\{(v_6, v_7)\}$ into the queue with the priorities of 2.5 and 2.3 respectively. Then, we pop out the top subtree in the queue, e.g., $\{(v_1, v_2)\}$, into the solution, and remove it from the trees. New trees are induced due to this removal.

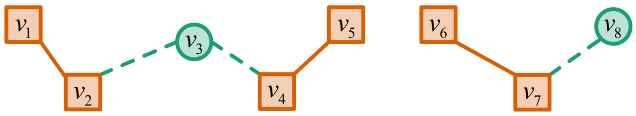


Figure 2: A graph composed of two non-overlapping trees. The prize of each box vertex is 2; the prize of each dot vertex is 0. The costs of solid edges $\{(v_1, v_2), (v_4, v_5), (v_6, v_7)\}$ are 1.5, 1.6, 1.7 respectively; the cost of each dash edge is 1.5.

For example, $\{(v_3, v_4), (v_4, v_5)\}$ is induced after removing $\{(v_1, v_2)\}$ from $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5)\}$. Subsequently, we push the maximum-net-weight subtree in each of the induced new trees into the queue with the priorities of their net-weights. For example, we push $\{(v_4, v_5)\}$, which is the maximum-net-weight subtree in $\{(v_3, v_4), (v_4, v_5)\}$, into the queue with the priority of 2.4. We keep doing this until k subtrees are popped out into the solution.

By applying the Goemans-Williamson growing scheme to cluster vertices into the same set of non-overlapping trees for the pruning process, our improved solutions are always better than or equal to those of Hegde *et al.*'s FGWA. For example, in Figure 2, we can produce the solution $\{(v_1, v_2), (v_4, v_5)\}$, which has a larger net-weight than Hegde *et al.*'s solution $\{(v_1, v_2), (v_6, v_7)\}$. The time complexity of producing our improved solutions is $O(d|E|log|V| + k|V|)$, where $O(k|V|)$ corresponds to our improved pruning process above. We note that the set of non-overlapping trees produced via the Goemans-Williamson growing scheme are raw solution trees. Since the pruning process is implemented on these raw solution trees, the time complexity of producing our improved solutions can be considered as $O(d|E|log|V| + k|V_{raw\_solu}|)$, while the time complexity of Hegde *et al.*'s FGWA can be considered as $O(d|E|log|V| + |V_{raw\_solu}|)$, where V_{raw_solu} is the set of vertices in these raw solution trees. We further note that the number of query nodes is often much smaller than the number of vertices in practice, i.e., $|Q| \ll |V|$ (e.g. researchers in a single area are often a fraction of researchers in all areas). As a result, we have $|V_{raw\_solu}| \ll |V|$. Moreover, we often have $|V| < |E|$ and a limited value of k in practice. Consequently, we have $O(d|E|log|V| + k|V_{raw\_solu}|) \approx O(d|E|log|V| + |V_{raw\_solu}|)$, which means that our improved solutions and Hegde *et al.*'s FGWA have nearly the same scalabilities in practice. We will later show this via experiments.

3. METHODOLOGY

In this section, we present our complete methodology for hunting multiple bumps in graphs.

3.1 The NP-hardness in trees

As stated above, given a graph, we first cluster vertices into a set of non-overlapping trees, and then hunt multiple bumps in this set of non-overlapping trees. Here, we show the NP-hardness of hunting multiple bumps in a set of non-overlapping trees. First, we define the Prize-Collecting Steiner Forest Problem in Trees (PCSFPT) as follows.

PROBLEM 3 (PCSFPT). *Given a set of non-overlapping trees $\cup\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$; and a target number $k \in \mathbb{N}$, the Prize-Collecting Steiner Forest Problem in Trees is to find k non-overlapping subtrees of these trees:*

$C_{sum}(V_{sum}, E_{sum}) = \cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$ *such that the net-weight of these subtrees:*

$$w(C_{sum}) = \sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) \quad (7)$$

is maximized.

Since the connectivity of G is not specified in the definition of PCSFP, PCSFPT can be considered as a special case of PCSFP where G is a set of non-overlapping trees, i.e., a forest. Thus, hunting multiple bumps in a set of non-overlapping trees can be transformed to PCSFPT via Theorem 1. We prove the NP-hardness of PCSFPT as follows.

THEOREM 2. *PCSFPT is NP-hard.*

PROOF. We prove the NP-hardness of PCSFPT by proving the NP-completeness of the decision version of PCSFPT via a reduction from the Boolean 3-satisfiability (3-SAT) problem [24]. The decision version of PCSFPT is as follows: given a set of non-overlapping trees $\cup\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$; a target number $k \in \mathbb{N}$; and a constant M, are there k non-overlapping subtrees of these trees: $C_{sum}(V_{sum}, E_{sum})$ such that $\sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) \geq M$? The 3-SAT problem is as follows: given a collection of clauses $CL = \{c_1, c_2, \cdots, c_n\}$ on a set of Boolean variables $BV = \{b_1, b_2, \cdots, b_k\}$ such that $|c_i| = 3 \mid \forall c_i \in CL$, is there a truth assignment for BV that satisfies all the clauses in CL?

First, a given solution to the decision version of PCSFPT can be verified in polynomial time, which means that the decision version of PCSFPT is in NP. Then, we give the reduction as follows. For each variable $b_i \in BV$, we have a root vertex $r(b_i)$ and a literal class $V_i = \{v(b_i), v(\overline{b_i})\}$ of two literal vertices representing the positive and negative literals respectively. There are k root vertices and k literal classes in total. We connect each $r(b_i)$ with $v(b_i)$ and $v(\overline{b_i})$ by an edge of cost δ respectively. For each clause $c_i \in CL$, we create a clause class V_{k+i} that contains a clause vertex $v_i(l)$ for each literal l in c_i. We connect each clause vertex $v_i(l)$ by an edge of cost δ to the corresponding literal vertex $v(l)$. The prizes of root vertices are P, and $P \gg \delta$. For each clause class V_{k+i}, we select a single clause vertex $v_i(l)$ in V_{k+i} and associate $v_i(l)$ with the prize of P. Every root vertex is connected with at least one of these special clause vertices. The prizes of all the other vertices are 0. The graph above is a set of non-overlapping trees. An instance of the decision version of PCSFPT in this graph is that: are there k non-overlapping subtrees: $C_{sum}(V_{sum}, E_{sum})$ such that $\sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) \geq M = (n+k)(P-\delta)$? Suppose that there are such k non-overlapping subtrees. In this case, all the root vertices and clause vertices with prizes of P are in these k non-overlapping subtrees. Since every root vertex connects at least one of the clause vertices with prizes of P, these k non-overlapping subtrees contain at least one literal vertex $v(l)$ in each literal class V_l. Suppose that there are $k+x$ literal vertices in these k non-overlapping subtrees, and $x \geq 0$. We have $\sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) = (n+k)P - (n+k+x)\delta$. If $\sum_{v \in V_{sum}} w(v) - \sum_{e \in E_{sum}} c(e) \geq M = (n+k)(P-\delta)$, then $x = 0$, which means that these k non-overlapping subtrees contain exactly one literal vertex $v(l)$ in each literal class V_l. As a result, there is a truth assignment for BV that satisfies all the clauses in CL. Given that the 3-SAT problem is among Karp's original 21 NP-complete problems [24], the decision version of PCSFPT is NP-complete. Hence, this theorem holds. □

In the above reduction from 3-SAT to PCSFPT, all the positive vertex prizes are equal and all the edge costs are equal. This shows that PCSFPT is NP-hard even when all the positive vertex prizes are equal and all the edge costs are equal. Solving BHPG in a set of non-overlapping trees can be transformed to solving PCSFPT via Theorem 1. It can be seen from Theorem 1 that, in the transformed graph G', all the positive vertex prizes, i.e., $\alpha+1$, are always equal, and all the edge costs, i.e., $\beta c(e)+1$, are equal when $\beta = 0$. Thus, we have the following corollary via the above reduction.

COROLLARY 1. *BHPG is NP-hard even when 1) G is a set of non-overlapping trees; and 2) $\beta = 0$.*

3.2 Our fast approximation algorithm in trees

Given a set of k non-overlapping trees, Hegde *et al.* used the Goemans-Williamson pruning algorithm [13] to find the maximum-net-weight subtree in each tree, and then combined these subtrees as a solution. Their method only suits scenarios where the input is a set of k non-overlapping trees. Even in these scenarios, their solutions may not have satisfactorily high qualities. To address these issues, we propose the Arborescent Bump Hunting Algorithm (ABHA), which can produce fast and high-quality solutions to PCSFPT for any input set of non-overlapping trees. There are three phases in ABHA. We introduce them as follows.

Phase 1: pushing bumps into a max priority queue: Given a set of non-overlapping trees $\cup\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$ and a target number $k \in \mathbb{N}$, we first initialize a max priority queue: $Q_P = \emptyset$ (Step 1). We will later push bumps into this queue with priorities of their net-weights. By doing this, we can hunt k large-net-weight bumps by popping out bumps

Algorithm 1 Our Arborescent Bump Hunting Algorithm (ABHA)

Input: a set of non-overlapping trees $\cup\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$, a target number $k \in \mathbb{N}$
Output: k non-overlapping subtrees $C_{sum}(V_{sum}, E_{sum}) = \cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$

1: Initialize $Q_P = \emptyset$
2: **for** each non-overlapping tree Θ_i **do**
3: $nw(v) = w(v), up(v) = 1 \mid \forall v \in V_{\Theta_i}$
4: **while** there is a vertex v: $up(v) = 1$ & $\xi(v) = 1$ **do**
5: **if** $c(v, v_{adj}) < nw(v)$ **then**
6: Update $nw(v_{adj})$ using Equation (9)
7: **end if**
8: $up(v) = 0$
9: **end while**
10: Find vertex r: $nw(r) \geq nw(i) \mid \forall i \in V_{\Theta_i}$
11: $Q_P : enqueue([r, \Theta_i]; nw(r))$
12: **end for**
13: **while** $|C_{sum}| < k$ & $Q_P \neq \emptyset$ **do**
14: $Q_P : dequeue([v_{top}, \Theta_{v_{top}}]; nw(v_{top}))$
15: $\Theta_{bump} = StrongPruning(\Theta_{v_{top}}; r = v_{top})$
16: $C_{sum} = C_{sum} \cup \Theta_{bump}$
17: $\cup\Theta_x = \Theta_{v_{top}} \setminus \Theta_{bump}$
18: Do Steps 2-12 on $\cup\Theta_x$
19: **end while**
20: **while** $|C_{sum}| < k$ **do**
21: $E_{sum} = E_{sum} \setminus (i, j)_{max}$
22: **end while**
23: **return** $C_{sum}(V_{sum}, E_{sum})$

from this queue. For each non-overlapping tree Θ_i (Step 2), we associate each vertex in Θ_i with an nw value that equals the prize of this vertex, and mark all these vertices as unprocessed: $nw(v) = w(v), up(v) = 1 \mid \forall v \in V_{\Theta_i}$ (Step 3). If vertex v is processed, then $up(v) = 0$. We define the processing degree of vertex v, $\xi(v)$, as the number of its adjacent vertices that are unprocessed:

$$\xi(v) = \sum_{(v,x)\in E_{\Theta_i}} up(x) \tag{8}$$

Initially, only leaves have a processing degree of 1. While there is a vertex v such that $up(v) = 1$ and $\xi(v) = 1$ (Step 4), we update the nw values of vertices in Θ_i as follows. If $c(v, v_{adj}) < nw(v)$ (Step 5), where v_{adj} is the unprocessed adjacent vertex to v, then we update $nw(v_{adj})$ using the equation below (Step 6).

$$nw(v_{adj}) = nw(v_{adj}) + nw(v) - c(v, v_{adj}) \tag{9}$$

Subsequently, we mark vertex v as processed: $up(v) = 0$ (Step 8). Notably, $\xi(v_{adj})$ decreases one due to this change. We iterate the above process until there is no vertex v such that $up(v) = 1$ and $\xi(v) = 1$. This process of updating nw values is the same with the first few steps of the general pruning algorithm [36]. We find vertex r that has the largest updated nw value (Step 10). The optimality of the general pruning algorithm for the Node-Weighted Steiner Tree Problem in Trees (NWSTPT) [36] shows that r is the root of the maximum-net-weight bump in Θ_i, and the net-weight of this bump is $nw(r)$. This bump is possibly in the solution of ABHA. To speed up ABHA, we do not directly find and push this bump into Q_P. Instead, we only push r (and thus Θ_i) into Q_P, with the priority of $nw(r)$ (Step 11). After this phase, the root of the maximum-net-weight bump in each non-overlapping tree Θ_i is in Q_P. Clearly, the top element in Q_P is the root of the maximum-net-weight bump in $\cup\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$.

Phase 2: popping out bumps from the priority queue: While k bumps have not been hunted and Q_P is not empty yet, i.e., $|C_{sum}| < k$ & $Q_P \neq \emptyset$ (Step 13), we hunt the maximum-net-weight bump as follows. First, we pop out the top element in Q_P (Step 14). Then, we use the strong pruning algorithm [23] to hunt the maximum-net-weight bump Θ_{bump} for the root v_{top} (Step 15). We incorporate the hunted bump into the solution: $C_{sum} = C_{sum} \cup \Theta_{bump}$ (Step 16), and remove the hunted bump from the tree (Step 17). New trees may be induced due to this removal. We push the maximum-net-weight bumps in the new trees into Q_P in the same way as the first phase (Step 18).

Phase 3: guaranteeing the target number k: It is possible that we cannot hunt k bumps through the above phase. For example, in scenarios where we have a single input tree and the net-weight of any subtree is smaller than the net-weight of this input tree, we can only hunt this input tree through the above phase. If we have not hunted k bumps through the above phase, i.e., $|C_{sum}| < k$ (Step 20), we remove the largest-cost edges in the hunted bumps, i.e., $E_{sum} = E_{sum} \backslash (i, j)_{max}$ (Step 21), until k bumps are hunted. Ultimately, we return k bumps, i.e., non-overlapping subtrees (Step 23). Notably, it is implied that $k \leq \sum |V_{\Theta_i}|$, as otherwise the target number k can never be achieved.

The solution quality of our ABHA: We prove that the solutions of our ABHA are always better than or equal to those of Hegde *et al.*'s method of combining the maximum-net-weight subtree in each tree as follows.

THEOREM 3. *The solutions of our ABHA are always better than or equal to those of Hegde et al.'s method for solving PCSFPT.*

PROOF. Suppose that $\cup_{i=1}^{k}\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c)$ are k non-overlapping trees; the roots of the maximum-net-weight subtrees in each $\cup_{i=1}^{k}\Theta_i$ are $r_1, \cdots, r_k$; and their priorities in Q_P satisfy $nw(r_1) \geq \cdots \geq nw(r_k)$. Since there are k non-overlapping trees, ABHA hunts k bumps in Phase 2, and Phase 3 is not triggered. Suppose that the roots of the hunted k bumps by ABHA are $x_1, \cdots, x_k$; and their priorities in Q_P satisfy $nw(x_1) \geq \cdots \geq nw(x_k)$. The optimality of the general pruning algorithm for NWSTPT [36] shows that the net-weight of the solution of ABHA is $\sum_{i=1}^{k} nw(x_i)$, where the nw values are the priorities of these roots in Q_P. Similarly, the net-weight of the solution of Hegde *et al.*'s method is $\sum_{i=1}^{k} nw(r_i)$. There are two possible scenarios:
Scenario 1: $\nexists i \in \{1, \cdots, k\}$ $x_i \neq r_i$, i.e., the roots of the hunted k bumps by ABHA are $r_1, \cdots, r_k$. In this scenario, the net-weights of the solutions of both ABHA and Hegde *et al.*'s method are $\sum_{i=1}^{k} nw(r_i)$.
Scenario 2: $\exists i \in \{1, \cdots, k\}$ $x_i \neq r_i$, i.e., there is at least one root of the hunted k bumps by ABHA that is not the root of the maximum-net-weight bump in an input tree. Suppose that x_i is such a root, and r_j is the root of a maximum-net-weight bump that is not hunted by ABHA. Since Q_P is a max priority queue, we have $nw(x_i) \geq nw(r_j)$. Thus, $\sum_{i=1}^{k} nw(x_i) \geq \sum_{i=1}^{k} nw(r_i)$, i.e., the net-weight of the solution of ABHA is larger than or equal to that of Hegde *et al.*'s method. Hence, this theorem holds. □

Given an instance for PCSFPT, it is implied that $k \leq \sum |V_{\Theta_i}|$ and there is at least one vertex with a positive prize. Therefore, there is always a feasible solution to PCSFPT that has a positive net-weight. For example, a set of k vertices in which at least one vertex has a positive prize is such a feasible solution. The above proof also indicates that the net-weights of the solutions of our ABHA are always positive, as these net-weights are larger than or equal to $\sum_{i=1}^{k} nw(r_i)$. We prove the approximation guarantee of our ABHA for solving PCSFPT as follows.

THEOREM 4. *Our ABHA has an approximation guarantee of $1/k$ for solving PCSFPT.*

PROOF. Given a set of non-overlapping trees $\cup\Theta_i$, let Θ_{ABHA} and Θ_{opt} be the solution of ABHA and the optimal solution respectively. Suppose that C_{max} is the maximum-net-weight component in $\cup\Theta_i$. Clearly, C_{max} is the first bump hunted by ABHA in Phase 2, and we have $w(\Theta_{ABHA}) \geq w(C_{max})$. Let C_{maxopt} be the maximum-net-weight bump in the optimal solution to PCSFPT. We have $w(C_{max}) \geq w(C_{maxopt})$. Thus, we have

$$kw(\Theta_{ABHA}) \geq kw(C_{max}) \geq kw(C_{maxopt}) \geq w(\Theta_{opt}) \tag{10}$$

Hence, this theorem holds. □

The time complexity of our ABHA: Our ABHA has a polynomial time complexity of $O(k|V|)$, where $|V| = \sum |V_{\Theta_i}|$. The details are as follows: the time complexity of Phase 1 is $O(|V|)$, as the time complexity of updating nw values by traversing all the vertices is $O(|V|)$; the time

complexity of Phase 2 is $O(k|V|)$, as the time complexity of de-queue for priority queues is $O(log|V|)$, the time complexity of the strong pruning algorithm is $O(|V|)$ [23], and we need to implement it k times in the worst case; and the time complexity of Phase 3 is $O(|V|)$, as we need to locate and remove the largest-cost edges, and $\sum|E_{\Theta_i}| < \sum|V_{\Theta_i}|$.

3.3 Our improved fast approximation algorithm for hunting bumps in graphs

We improve Hegde *et al.*'s FGWA using our ABHA. We refer to the improved algorithm as the Graphical Bump Hunting Algorithm (GBHA). There are two phases in GBHA.

Phase 1: clustering vertices: Same as FGWA, we employ the fast implementation of the Goemans-Williamson growing scheme [20, 21] to cluster vertices into k non-overlapping trees (Step 1).

Phase 2: hunting k non-overlapping subtrees: Different from FGWA that combines the maximum-net-weight subtree in each of k non-overlapping trees, we apply our ABHA to find k non-overlapping subtrees (Step 2). These subtrees are the solution of GBHA (Step 3).

The solution quality of our GBHA: Our GBHA is different from Hegde *et al.*'s FGWA in the second phase. Since the solutions of our ABHA are always better than or equal to those of Hegde *et al.*'s method in the second phase (see Theorem 3), we have the following corollaries.

COROLLARY 2. *The solutions of our GBHA are always better than or equal to those of Hegde et al.'s FGWA for solving PCSFP.*

COROLLARY 3. *Given a graph $G(V, E, w, c)$; and a target number $k \in \mathbb{N}$, our GBHA returns k non-overlapping trees such that*

$$\frac{\sum_{e\in E_{sum}} c(e) + 2\sum_{i\in V\setminus V_{sum}} w(i)}{2\sum_{e\in E_{OPT}} c(e) + 2\sum_{i\in V\setminus V_{OPT}} w(i)} \leq \quad (11)$$

$$\frac{2\sum_{i\in V_{sum}} w(i) - \sum_{e\in E_{sum}} c(e)}{2\sum_{i\in V_{OPT}} w(i) - 2\sum_{e\in E_{OPT}} c(e)} \geq \quad (12)$$

where $C_{sum}(V_{sum}, E_{sum})$ and $C_{OPT}(V_{OPT}, E_{OPT})$ are the solution of our GBHA and the optimal solution to PCSFP respectively.

Equation (11) is from the approximation guarantee of FGWA, and induces Equation (12). Equation (11) shows that our GBHA has an approximation guarantee of 2 with respect to minimizing $c(C_{sum})$, while the work of Feigenbaum *et al.* [10] shows that it is NP-hard to approximately maximize $w(C_{sum})$ within any constant factor.

The time complexity of our GBHA: Our GBHA has a polynomial time complexity of $O(d|E|log|V| + k|V|)$, as the time complexity of the fast implementation of the Goemans-Williamson growing scheme is $O(d|E|log|V|)$ [20], where d is the precision of prizes and costs, and the time complexity of our ABHA is $O(k|V|)$.

Algorithm 2 Our Graphical Bump Hunting Algorithm (GBHA)

Input: a graph $G(V, E, w, c)$; a target number $k \in \mathbb{N}$
Output: k non-overlapping trees
$C_{sum}(V_{sum}, E_{sum}) = \cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$
1: $\cup_{i=1}^{k}\Theta_i(V_{\Theta_i}, E_{\Theta_i}, w, c) = FastGW(G)$
2: $C_{sum}(V_{sum}, E_{sum}) = ABHA(\cup_{i=1}^{k}\Theta_i, k)$
3: Return $C_{sum}(V_{sum}, E_{sum})$

3.4 Our adaptions of the existing bump hunting algorithms in graphs

Three heuristic algorithms have been developed by Gionis *et al.* [12] for hunting a single bump in an unweighted graph: Breadth-first search trees (BF-ST), Random spanning trees (Random-ST), and Smart spanning trees (Smart-ST). The main idea is to apply their TreeOptimal algorithm to hunt a high-discrepancy bump in a spanning tree of the graph. Here, we adapt these algorithms by replacing their TreeOptimal algorithm with our ABHA.

Our adapted BF-ST: First, we select a root query node, and add dummy edges between this query node and a random query node in each maximal component that contains at least one query node and is disconnected with the root query node. Subsequently, we find the breadth-first search tree from the root query node, and remove dummy edges from this tree for generating a set of non-overlapping trees that contains all the query nodes. We apply ABHA to hunt k bumps in this set of non-overlapping trees. Gionis *et al.* select every query node to be a root query node for producing multiple heuristic solutions, and then consider the best solution as the final solution. It is too slow to do this in large graphs. Thus, we only randomly select a pre-fixed number of query nodes to be root query nodes. Since the time complexity of checking the graph connectivity is $O(|V| + |E|)$ [37]; the time complexity of breadth-first search is $O(|V| + |E|)$ [9]; and the time complexity of our ABHA is $O(k|V|)$, the time complexity of our adapted BF-ST is $O(mk|V| + m|E|)$, where m is the number of selected root query nodes.

Our adapted Random-ST: First, we assign new edge costs randomly. Then, we add a dummy vertex to connect all the other vertices using dummy edges with large costs for guaranteeing that there is a spanning tree that spans all the query nodes. We find a Minimum Spanning Tree (MST) for the above edge costs as a random spanning tree, and remove dummy edges from this tree for generating a set of non-overlapping trees that contains all the query nodes. We apply ABHA to hunt k bumps in this set of non-overlapping trees. We sample multiple random spanning trees for producing multiple heuristic solutions, and then consider the best solution as the final solution. Since the time complexity of adding and removing a dummy vertex for an adjacency list is $O(|V| + |E|)$; the time complexity of finding an MST is $O(|E| + |V|log|V|)$ [31]; and the time complexity of our ABHA is $O(k|V|)$, the time complexity of our adapted Random-ST is $O(p|E| + p|V|log|V| + pk|V|)$, where p is the number of sampled random spanning trees.

Our adapted Smart-ST: First, like the original Smart-ST, we assign new edge costs as follows.

$$c'(u, v) = 2 - I\{u \in Q\} - I\{v \in Q\} \quad (13)$$

where $I\{\cdot\}$ is the indicator function. We add a dummy vertex to connect all the other vertices using dummy edges with large costs for guaranteeing that there is a spanning tree that spans all the query nodes. We find an MST for the above edge costs as a smart spanning tree, and remove dummy edges from this tree for generating a set of non-overlapping

trees that contains all the query nodes. We apply ABHA to hunt k bumps in this set of non-overlapping trees. Since the time complexity of adding and removing a dummy vertex for an adjacency list is $O(|V| + |E|)$; the time complexity of finding an MST is $O(|E| + |V|log|V|)$ [31]; and the time complexity of our ABHA is $O(k|V|)$, the time complexity of our adapted Smart-ST is $O(|E| + |V|log|V| + k|V|)$.

4. EXPERIMENTAL EVALUATION

We conduct experiments using 8 virtual machines, each with 16 GB RAM, on a cloud with Intel 8168 Processors[3].

4.1 The experiment settings

Datasets: We apply two real datasets as follows.
1) Twitter: It was collected by Nikolakaki *et al.* [29] for building a Twitter activity network. This network corresponds to the road network in Austin. Each vertex represents a road intersection, and each edge represents a road. Each vertex is associated with a prize that is the number of geo-located Twitter posts near the corresponding road intersection. Each edge is associated with a cost that is the length of the corresponding road. There are 66,200 vertices and 92,707 edges in total.
2) DBLP: We collect it from the DBLP website [2]. We use it to build a social network, where vertices represent researchers, and two researchers are connected if they have co-authored publication(s). Each researcher is associated with a list of keywords that are in the titles of his or her publications. We use pairwise Jaccard distances as edge costs between researchers, i.e., $c(u, v) = 1 - |V_u \cap V_v|/|V_u \cup V_v|$, where V_u and V_v are the sets of adjacent vertices of u and v respectively. There are 1,094,552 vertices, 6,911,318 edges, and 82,492 keywords in total.

Algorithms: We compare five algorithms as follows.
1) Hegde *et al.*'s FGWA [21]: It is Hegde *et al.*'s algorithm for solving PCSFP.
2) Our GBHA: It is our improvement on Hegde *et al.*'s FGWA. Its solutions are always better than or equal to those of Hegde *et al.*'s FGWA (details in Corollary 2).
3) Our adapted BF-ST: It is our adaption of Gionis *et al.*'s [12] first bump hunting algorithm in graphs.
4) Our adapted Random-ST: It is our adaption of Gionis *et al.*'s [12] second bump hunting algorithm in graphs.
5) Our adapted Smart-ST: It is our adaption of Gionis *et al.*'s [12] third bump hunting algorithm in graphs.

The running times of our adapted BF-ST and Random-ST are proportional to m and p respectively (details in Section 3.4). We set $m = p = 10$ for guaranteeing that our adapted BF-ST and Random-ST are fast enough to be implemented.

Parameters: We vary five parameters as follows.
1) $|V|$: It is the number of vertices.
2) $|Q|$: It is the number of query nodes.
3) k: It is the target number of bumps.
4) α: It is the regulating weight between the numbers of query and non-query nodes in Equation (1).
5) β: It is the regulating weight between component discrepancies and edge costs in Equation (2).

Metrics: We evaluate two metrics as follows.
1) $w(C_{sum})$: It is the objective value of PCSFP, and is equivalent to that of BHPG (details in the proof of Theorem 1).

[3]The codes and datasets are available at `https://github.com/YahuiSun/bump_hunting`

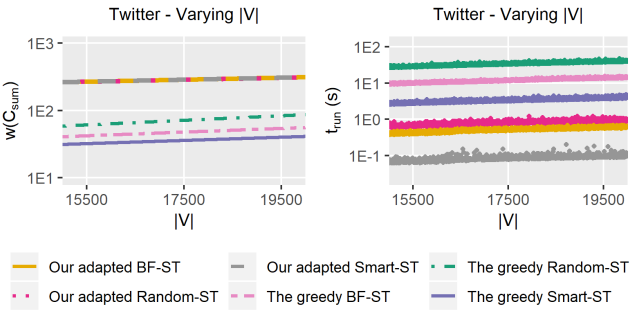


Figure 3: The greedy algorithms (best viewed in color)

2) t_{run}: It is the running time of algorithms.

Notably, the original BF-ST, Random-ST and Smart-ST can hunt multiple bumps by iteratively hunting and removing a single bump. Such a greedy implementation does not consider edge costs, and is too slow to be implemented in large graphs due to 1) the requirement of bump roots in the TreeOptimal algorithm [12] (a detailed comparison of rooted and unrooted pruning algorithms is in [36]); and 2) the requirement of finding a large number of spanning trees (e.g., such a greedy implementation of Smart-ST finds k smart spanning trees for hunting k bumps). As a result, such a greedy implementation cannot hunt high-quality bumps in large weighted graphs. We show this in Figure 3, for which the implementation details are the same with our following experiments in Figures 5a (1-A,B), and we also set $m = p = 10$ for such a greedy implementation of BF-ST and Random-ST. Due to the above issues, we do not compare such a greedy implementation in our following experiments.

4.2 The implementation details

Here, we describe how the experiments are implemented.

Building raw networks: We read raw datasets to build the raw Twitter and DBLP networks as described above.

Querying nodes: Query nodes are nodes that exhibit a property of interest. For Twitter, we set a random Lower Bound (LB) of prizes, and consider vertices that have larger or equal prizes as query nodes. For DBLP, we first randomly query a keyword, and then randomly find some other keywords that share researchers with the queried keyword. The set of all these keywords is a set of correlated keywords with the queried keyword being the core. We consider vertices that represent researchers who are associated with at least one of these correlated keywords as query nodes. The reason for finding a set of correlated keywords is that the number of researchers who are associated with a single keyword is often limited. As a result, considering only vertices that represent researchers who are associated with the queried keyword as query nodes often makes the generated instances too trivial to solve for the experimental evaluation.

Varying parameters: We vary five parameters as follows.
Varying $|V|$: We vary $|V|$ to the maximum in Figures 5a-5b (1) for Twitter and DBLP respectively.
Varying $|Q|$: The results of varying $|Q|$ by varying the LB of prizes for Twitter and the number of correlated keywords for DBLP are in Figures 4 (1-2). We vary $|Q|$ in Figures 5a-5b (2) by randomly selecting an LB of prizes in the range of $[10, 20]$ for Twitter, and randomly selecting a set of 30

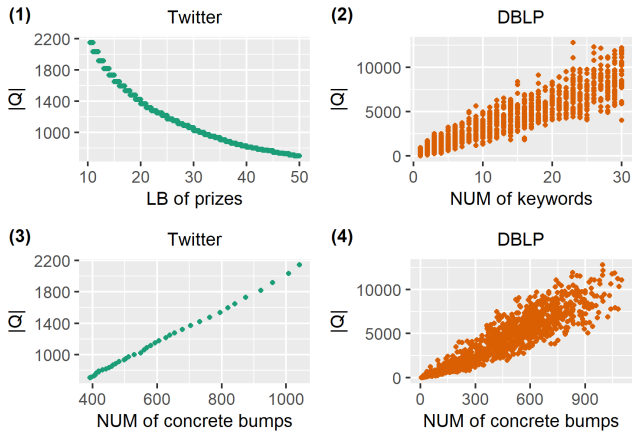


Figure 4: Some statistic results on Twitter and DBLP

correlated keywords for DBLP. We visualize such ranges of $|Q|$ that the experiment results are visualized clearly.

Varying k: We explain the logic behind varying k as follows. We are interested in components with positive qualities. Such components represent regions or communities that exhibit the property of interest in the Twitter or DBLP network. We consider such components as bumps. We further neglect bumps that overlap with another bump that has a higher quality (a tie is broken randomly) for guaranteeing that bumps do not overlap with each other, as otherwise different bumps may correspond to the same region or community. We refer to such non-overlapping bumps as *concrete bumps*. The number of concrete bumps in a graph can then be counted by iteratively finding and removing the highest-quality concrete bump. Theorem 1 indicates that finding the highest-quality concrete bump is equivalent to solving the Prize-Collecting Steiner Tree Problem [23], for which a state-of-the-art solution approach is the Goemans-Williamson scheme [13, 20, 36]. Applying the Goemans-Williamson scheme to count the number of concrete bumps in a graph by iteratively finding and removing the highest-quality concrete bump can be done efficiently by 1) applying the Goemans-Williamson growing scheme to cluster vertices to such a degree that only one active cluster remains; and 2) applying Steps 1-19 in our ABHA (push r into Q_P when $nw(r) > 0$ in Step 11; and without $|C_{sum}| < k$ in Step 13) to hunt concrete bumps in both active and non-active clusters. The time complexity of conducting the above two phases is $O(d|E|log|V| + h|V|)$, where h is the number of concrete bumps in a graph. We visualize the numbers of concrete bumps with respect to $|Q|$ (when $\alpha = \beta = 1$) for Twitter and DBLP in Figures 4 (3-4). The running times of counting each of these numbers are around 1.5s and 250s for Twitter and DBLP respectively.

When $k \leq h$, the top-k highest-quality concrete bumps are hunted, otherwise divided concrete bumps are hunted. In practice, it is often preferable to set $k \leq h$ for guaranteeing that concrete bumps are not divided (e.g. hunting bumps in the DBLP network for analyzing naturally formed communities). We vary k in Figures 5a-5b (3) for Twitter and DBLP respectively. For Twitter, the number of concrete bumps is 1104, and the percentage of these concrete bumps that contain more than one vertex is 16%. For DBLP, the number of concrete bumps is around 750, and the percentage of these concrete bumps that contain more than one vertex is around 30%. This means that most of these concrete bumps only contain a single vertex and thus have the same low quality, while, in practice, it is often preferable to hunt large and high-quality bumps that have collective representations (e.g. regions or communities). Following this logic and with the statistic results above, we vary k in the ranges of $[100, 150]$ and $[150, 200]$ for Twitter and DBLP respectively for hunting large and high-quality bumps.

Furthermore, we note that, as k is closer to h, $w(C_{sum})$ and t_{run} values of our GBHA and Hegde *et al.*'s FGWA become closer, since the number of concrete bumps in each cluster of vertices produced via the Goemans-Williamson growing scheme is closer to 1 (see Figure 2). In the extreme scenario where $k = h$ and the Goemans-Williamson growing scheme returns h concrete bumps, $w(C_{sum})$ and t_{run} values of our GBHA and Hegde *et al.*'s FGWA are the same. This does not undermine our contributions in this paper, since setting k close to h requires the prior-knowledge of h, and this prior-knowledge is obtained via the above two phases that incorporate our ABHA, which is the basic difference between our GBHA and Hegde *et al.*'s FGWA.

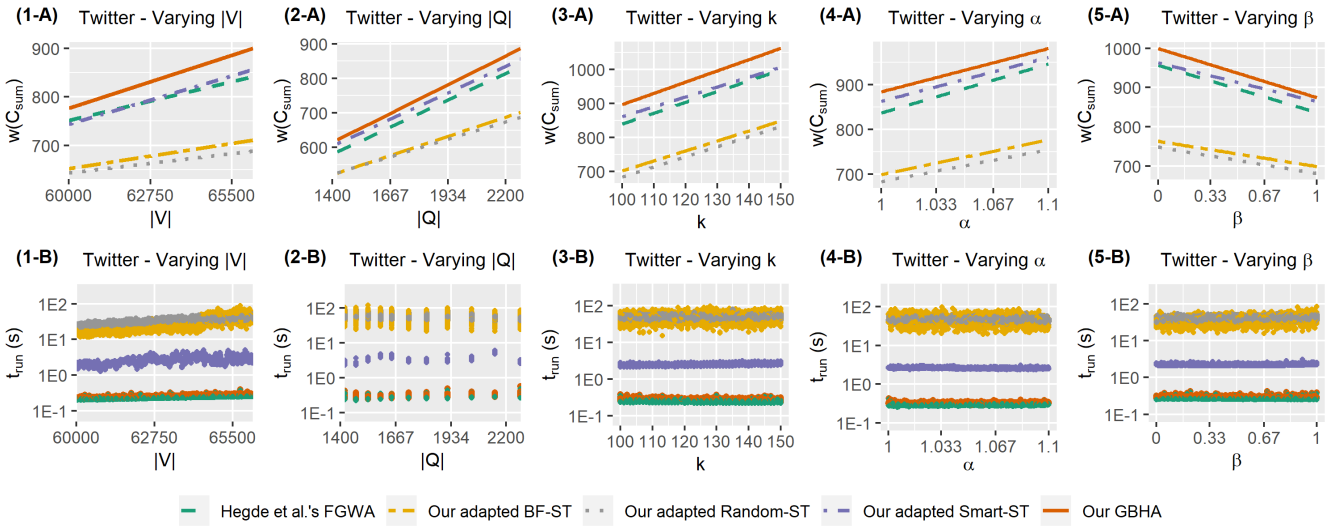
Varying α and β: It is preferable to set α large, as otherwise small vertex prizes turn bumps into singular vertices (see Equation (5)). It is preferable to set β small, as otherwise large edge costs turn bumps into singular vertices (see Equation (6)). In Figures 5a-5b (4-5), we vary α and β in the ranges of $[1, 1.1]$ and $[0, 1]$ respectively for Twitter, while in the ranges of $[1, 1.1]$ and $[0.9, 1]$ respectively for DBLP. We select these ranges in such a way that $w(C_{sum})$ values of different algorithms are visualized clearly.

Generating instances: After the process above, we have raw networks, query nodes and parameters to generate instances. For each instance, we select the first $|V|$ vertices from the raw network, and build a graph $G(V, E, c)$ using these vertices and all the edges between them in the raw network. Since graph connectivity is not specified in BHPG, we do not check the connectivity of G. We update vertex prizes and edge costs to generate an instance graph $G'(V, E, w', c')$ for PCSFP using Equations (5-6).

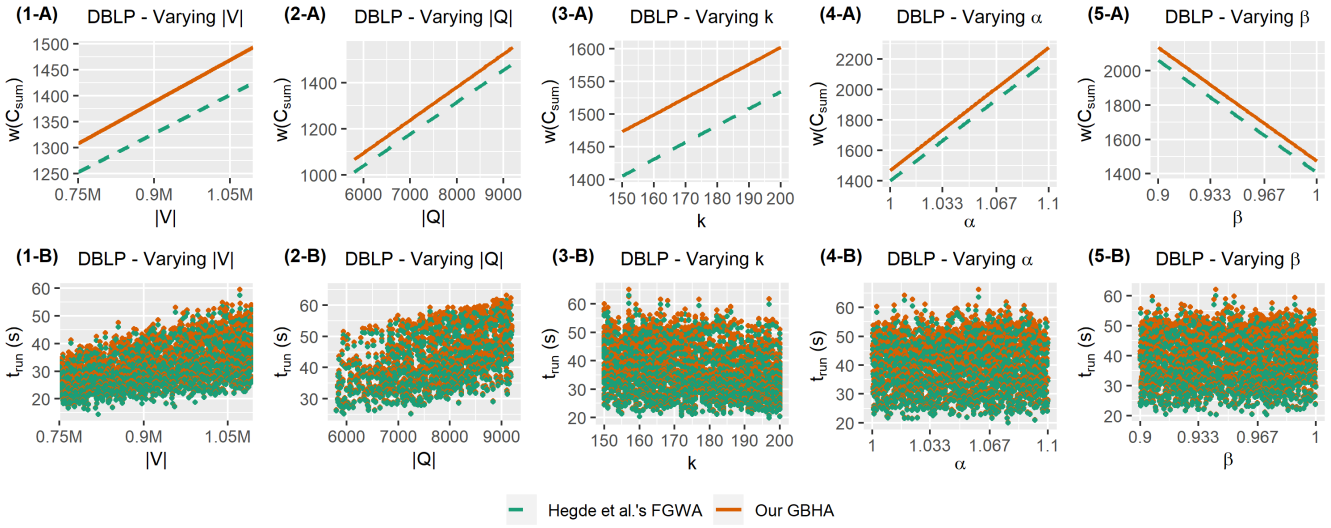
Producing and visualizing experiment results: The experiment results are visualized in Figure 5. In each pair of sub-figures, e.g. Figures 5a (1-A,B), we target a dataset, and randomly vary a parameter 2000 times to generate 2000 instances. Then, we apply algorithms to solve these instances. Since our adapted algorithms do not scale well to large graphs, we only implement our adapted algorithms for Twitter. Since the changes of $w(C_{sum})$ values are roughly linear, we use smoothing lines of the Generalized Additive Model [41] to visualize $w(C_{sum})$ values, while we use scatter plots to visualize t_{run} values. To visualize t_{run} values of different algorithms clearly, we use the exponential notation of t_{run} values in some figures, such as Figure 5a (1-B).

4.3 The experiment results

Varying $|V|$: We vary $|V|$ in Figures 5a-5b (1). We observe that $w(C_{sum})$ values of our GBHA are the largest. This has been observed in the following experiment results as well. We also observe that $w(C_{sum})$ increases with $|V|$. The reason is that bumps with larger net-weights may exist in larger graphs. We further observe that, different from our adapted algorithms, both our GBHA and Hegde *et al.*'s FGWA scale



(a) Twitter (default values: $|V| = 66200$; the LB of prizes is 10 for $|Q|$; $k = 100$; $\alpha = \beta = 1$)



(b) DBLP (default values: $|V| = 1094552$; the number of correlated keywords is 30 for $|Q|$; $k = 150$; $\alpha = \beta = 1$)

Figure 5: The experiment results in Twitter and DBLP (best viewed in color)

well to large graphs. This verifies the usefulness of the fast implementation of the Goemans-Williamson scheme [20] in large graphs.

Varying $|Q|$: We vary $|Q|$ in Figures 5a-5b (2). We observe that $w(C_{sum})$ increases with $|Q|$. The reason is that bumps with larger net-weights may exist in graphs with more query nodes. We also observe that scatter plots for Twitter are discontinuous with respect to $|Q|$. The reason is that vertex prizes in Twitter are integers, and we vary $|Q|$ by randomly selecting an LB of prizes in the range of $[10, 20]$ (details in Section 4.2). We further observe that t_{run} values of our GBHA and Hegde *et al.*'s FGWA increase with $|Q|$. The reason is that the number of initial clusters in the Goemans-Williamson growing scheme equals the number of vertices with positive prizes, which is $|Q|$ (see Theorem 1).

Varying k: We vary k in Figures 5a-5b (3). We observe that $w(C_{sum})$ increases with k. The reason is that a larger number of bumps has a larger net-weight. We also observe that t_{run} values of our GBHA and Hegde *et al.*'s FGWA decrease with k. The reason is that the Goemans-Williamson growing scheme terminates when k out of $|Q|$ active clusters of vertices remain. Nevertheless, it may not be preferable to set k close to the number of concrete bumps in many cases, as 1) doing this requires counting the number of concrete bumps, which is slow; and 2) most concrete bumps only contain a single vertex, and thus are less meaningful to be hunted in practice (details in Section 4.2).

Varying α: We vary α in Figures 5a-5b (4). We observe that $w(C_{sum})$ increases with α. The reason is that node weights increase with α (see Equation (5)). We further observe that t_{run} does not change much with α.

Varying β: We vary β in Figures 5a-5b (5). We observe that $w(C_{sum})$ decreases with β. The reason is that edge costs increase with β (see Equation (6)). To visualize different $w(C_{sum})$ values of our GBHA and Hegde *et al.*'s FGWA clearly, we do not vary β to 0 for DBLP. Nevertheless, we

Table 1: The relative bump qualities (FGWA: 100%)

Datasets	GBHA	BF-ST	Random-ST	Smart-ST
Twitter	105.38%	84.04%	82.27%	101.86%
DBLP	104.41%	N/A	N/A	N/A

note that $w(C_{sum})$ values of our GBHA and Hegde *et al.*'s FGWA for DBLP are close when $\beta = 0$. The reason is that most hunted bumps only contain two query nodes and an edge, and, since edge costs are neglected when $\beta = 0$, the qualities of these bumps are equal when $\beta = 0$. We further observe that t_{run} does not change much with β.

Evaluating algorithms: We present the relative bump qualities of algorithms in Table 1 by averaging their $w(C_{sum})$ values in the above experiment results. We observe that bumps hunted by our GBHA have the highest quality. We also observe that bumps hunted by our adapted Smart-ST have higher qualities than those hunted by Hegde *et al.*'s FGWA for Twitter. We further observe that our adapted BF-ST, Random-ST and Smart-ST do not scale well to large graphs. In comparison, our GBHA and Hegde *et al.*'s FGWA scale well to large graphs. The reason is that the time complexities of our GBHA and Hegde *et al.*'s FGWA can be considered as $O(d|E|log|V| + k|V_{raw\_solu}|)$ and $O(d|E|log|V| + |V_{raw\_solu}|)$ respectively, which are approximately equal in practice (details in Section 2.3). Since our GBHA is better than Hegde *et al.*'s FGWA with respect to bump quality and similar to Hegde *et al.*'s FGWA with respect to scalability, it may be preferable to apply our GBHA for hunting multiple bumps in graphs.

5. APPLICATIONS

In this section, we extend two existing bump hunting applications by hunting multiple bumps in a weighted graph.

5.1 Finding communities of researchers

Gionis *et al.* [12] showed that, given a social network, we can find a community of a property of interest by hunting a bump for a set of query nodes representing persons who exhibit this property of interest. Here, we extend this application by finding multiple communities of a property of interest. We take the DBLP dataset as an example. The step-by-step implementation details are as follows.

Loading the social network: We load the DBLP network in Section 4.1. Recall that vertices and edges represent researchers and collaborations between researchers respectively; each researcher is associated with a list of keywords; and each edge is associated with a cost that indicates the dis-closeness between researchers.

Querying researchers: Given a set of correlated keywords (details in Section 4.2), we consider researchers who are associated with at least one of these keywords as queried.

Generating a Steiner Forest instance: With the DBLP network and the queried researchers, which are query nodes, we generate the Prize-Collecting Steiner Forest instance $G'(V, E, w', c')$ using Theorem 1. Here, the regulating weights incorporated in this process are $\alpha = \beta = 1$.

Finding the top-k communities: We hunt the top-k bumps by applying our GBHA to solve PCSFP in G'. These bumps represent communities of the queried researchers who are closely connected. We visualize the second and third bumps for the single keyword "snake robots" in Figure 1 (notably, communities represented by tree-structured bumps intuitively contain all the edges between vertices in these bumps). Such visualizations help us analyze the development of a specific research area vividly.

Table 2: Statistic results on detected communities

Algorithms	$\lvert V_{in}\rvert$	*Density*	$g(C)$	$\lvert E_{in}\rvert/\lvert E_{out}\rvert$
Our GBHA	380	0.019	378	0.04
Newman [28]	73577	0.00014	-72780	1.22
Blondel [5]	199605	0.000068	-198359	53.22

Differentiating from conventional community detection approaches: Our bump hunting approach is different from conventional community detection approaches (e.g. [28, 5, 11, 26, 42]) in its unique objective of discrepancy maximization. The algorithms developed by Newman [28] and Blondel *et al.* [5] are two widely-used conventional community detection algorithms. These two algorithms aim to detect communities with a large number of within-community edges and a small number of between-community edges. In Table 2, we compare the top-5 largest communities detected by each of these two algorithms with the top-5 communities detected by our GBHA for each of 10 randomly queried sets of 10 correlated keywords (details in Section 4.2). Four metrics are used: 1) $|V_{in}|$: the average number of vertices inside communities; 2) *Density* [8]: the average density of communities; 3) $g(C)$: the discrepancy (for communities detected by two conventional algorithms, we randomly select a set of 10 correlated keywords from the keywords associated with researchers inside communities, and then consider researchers who are associated with at least one of these keywords as query nodes); and 4) $|E_{in}|/|E_{out}|$: the average number of edges inside communities divide the average number of edges connecting communities to the outside. We observe that communities detected by our GBHA have a smaller $|V_{in}|$. The reason is that the number of query nodes is limited. Since the number of collaborations of each researcher is often limited in reality, smaller communities are often denser. As a result, communities detected by our GBHA have a larger *Density*. We also observe that communities detected by two conventional algorithms have large negative $g(C)$, which indicates that these communities may not exhibit a property of interest frequently. We further observe that communities detected by our GBHA have a smaller $|E_{in}|/|E_{out}|$. The reason is that queried researchers have a lot of collaborations with not-queried researchers, which indicates that research communities of a property of interest may not be echo chambers of this property of interest in reality. These comparison results verify the unique usefulness of bump hunting for community detection.

5.2 Finding regions with high levels of activity

Rozenshtein *et al.* [32] showed that, given an activity network, we can find a region that exhibits a high level of activity via the Prize-Collecting Steiner Tree approach. Due to the similarity between the Prize-Collecting Steiner approach and the bump hunting approach (see Theorem 1), we consider this application as bump hunting. Here, we extend this application by finding multiple regions that exhibit high levels of activity. We take the Twitter dataset as an example. The step-by-step implementation details are as follows.

Figure 6: Finding regions that exhibit high levels of Twitter activity in Austin. The orange, red and magenta lines are the top-3 bumps, which correspond to the downtown, the South Congress Avenue, and the East Sixth Street.

Loading the activity network: We load the Twitter activity network in Section 4.1. Recall that vertices and edges represent road intersections and roads respectively; each road intersection is associated with an activity level that is the number of nearby Twitter posts; each edge is associated with a cost this is the corresponding road distance.

Querying road intersections: Given a lower bound of activity level, we consider road intersections that have higher or equal activity levels as queried. Here, we set the lower bound to 100.

Generating a Steiner Forest instance: With the activity network and the queried road intersections, which are query nodes, we generate the Prize-Collecting Steiner Forest instance $G'(V, E, w', c')$ using Theorem 1. Here, the regulating weights incorporated in this process are $\alpha = \beta = 1$.

Finding the top-k regions: We hunt the top-k bumps by applying GBHA to solve PCSFP in G'. These bumps represent regions that exhibit high levels of close activity. We visualize the top-3 bumps in Figure 6. Such visualizations help us analyze the behaviors of Twitter users, and allocate governmental resources for building smart cities. Furthermore, techniques of detecting activities (e.g. crowdsensing [18]) can be incorporated to extend this work in the future.

6. RELATED WORK

The origin and development of bump hunting: The term "bump hunting" originated in the field of high energy physics in the middle of the 20th century (e.g. [7, 33]), when it referred to the activity of detecting real bumps in mass spectra in scattering experiments. Orear and Cassel (1971) [30] described bump hunting as "one of the major current activities of high-energy physicists", while Trigg (1970) [38] criticized such a rash in a sarcastic way. Even with this controversy, bump hunting has been continuously studied and become an important data analysis approach (e.g. [14, 19, 3, 22]). The traditional bump hunting methods suit Euclidean datasets, but not graph datasets. Recently, Gionis *et al.* (2017) [12] explored bump hunting for graph datasets. Hunting bumps in graphs is different from the other graph mining approaches (e.g. [6, 4, 27, 17, 40]) in its unique objective of discrepancy maximization. Gionis *et al.* hunted a single bump in an unweighted graph. Here, we extend their work by hunting multiple bumps in a weighted graph.

The prize-collecting Steiner approach to data analytics: The prize-collecting Steiner problems, of which the most well-known one is the Prize-Collecting Steiner Tree Problem (PCSTP) [23], have been studied intensively in the last decade. Given a graph, PCSTP is to find such a tree that edge costs in this tree and positive vertex prizes not in this tree are minimized simultaneously. A lot of work has been done to explore the Prize-Collecting Steiner Tree approach to data analytics (e.g. [39, 32, 34, 12]). Different from PCSTP, Hegde *et al.* (2015) [21] formulated the Prize-Collecting Steiner Forest Problem (PCSFP) for graph-structured sparsity. PCSFP is more general than PCSTP in that, instead of finding a single tree, it finds a set of k non-overlapping trees. Their Prize-Collecting Steiner Forest algorithm [21] is the only existing and thus the state-of-the-art algorithm for solving PCSFP to date. Here, we improve this algorithm for hunting multiple bumps in a weighted graph.

7. CONCLUSIONS

In this paper, we extend the existing bump hunting researches by hunting multiple bumps in a weighted graph. Initially, we prove that this extended bump hunting problem can be transformed to PCSFP. Then, we prove that PCSFP is NP-hard even in trees. Subsequently, we propose a fast approximation algorithm for solving PCSFP in trees. By incorporating this algorithm, we improve the state-of-the-art algorithm for solving PCSFP in graphs, and prove that the solutions of our improvement are always better than or equal to those of the state-of-the-art algorithm. Moreover, we adapt the existing bump hunting algorithms for solving our extended bump hunting problem. We evaluate our methodology via real datasets, and show that 1) our improvement scales well to large graphs, while producing solutions that dominate those of the state-of-the-art algorithm; and 2) our adaptation of an existing bump hunting algorithm can also produce solutions that are better than those of the state-of-the-art algorithm in some cases.

Acknowledgment: This work is funded by MOE2016-T2-2-022 from the Singapore Ministry of Education.

APPENDIX

The proof of Theorem 1:

PROOF. Let $C^B_{sum}(V_{sum}, E^B_{sum}) = \cup_{m=1}^{k} C_m(V_{C_m}, E_{C_m})$ be the optimal solution to BHPG in G. Subsequently, let $C_{sum}(V_{sum}, E_{sum}) \subseteq C^B_{sum}(V_{sum}, E^B_{sum})$ be a set of k non-overlapping trees. We have $|V_{sum}| = k + |E_{sum}|$. Since C^B_{sum} is a set of k non-overlapping trees when $\beta > 0$, we have $E^B_{sum} = E_{sum}$ when $\beta > 0$. Thus,

$$\begin{aligned}
& \textstyle\sum_{m=1}^{k} Qua(C_m) \\
= \; & \textstyle\sum_{m=1}^{k} [g(C_m) - \beta c_{C_m}] \\
= \; & \alpha |V_{sum} \cap Q| - |V_{sum} \setminus Q| - \beta \textstyle\sum_{e \in E_{sum}} c(e) \\
= \; & (\alpha + 1)|V_{sum} \cap Q| - (|V_{sum} \cap Q| + |V_{sum} \setminus Q|) \\
& - \beta \textstyle\sum_{e \in E_{sum}} c(e) \\
= \; & (\alpha + 1)|V_{sum} \cap Q| - |V_{sum}| - \beta \textstyle\sum_{e \in E_{sum}} c(e) \\
= \; & \textstyle\sum_{v \in V_{sum} \cap Q} (\alpha + 1) - (k + |E_{sum}|) \\
& - \textstyle\sum_{e \in E_{sum}} \beta c(e) \\
= \; & \textstyle\sum_{v \in V_{sum} \cap Q} (\alpha + 1) - k - \sum_{e \in E_{sum}} (\beta c(e) + 1) \\
= \; & \textstyle\sum_{v \in V_{sum}} w'(v) - k - \sum_{e \in E_{sum}} c'(e)
\end{aligned} \tag{14}$$

Therefore, finding $C^B_{sum}(V_{sum}, E^B_{sum})$ that maximizes $\sum_{m=1}^{k} Qua(C_m)$ in G and finding $C_{sum}(V_{sum}, E_{sum})$ that maximizes $w(C_{sum}) = \sum_{v \in V_{sum}} w'(v) - \sum_{e \in E_{sum}} c'(e)$ in G' are equivalent. Hence, this theorem holds. □

REFERENCES

[1] 11th DIMACS Implementation Challenge. http://dimacs11.zib.de/.

[2] DBLP: computer science bibliography. https://dblp.uni-trier.de/.

[3] D. Agarwal, J. M. Phillips, and S. Venkatasubramanian. The hunting of the bump: on maximizing statistical discrepancy. In *Proceedings of the seventeenth annual ACM-SIAM Symposium on Discrete Algorithm*, pages 1137–1146. Society for Industrial and Applied Mathematics, 2006.

[4] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys*, 41(3):15, 2009.

[7] A. Clark, G. Conforto, A. Key, H. Neal, and J. Pine. Eexperiments to study meson resonances. *eConf*, 690609(NAL-1969-147):147, 1969.

[8] T. F. Coleman and J. J. Moré. Estimation of sparse jacobian matrices and graph coloring blems. *SIAM journal on Numerical Analysis*, 20(1):187–209, 1983.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[10] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.

[11] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[12] A. Gionis, M. Mathioudakis, and A. Ukkonen. Bump hunting in the dark: Local discrepancy maximization on graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):529–542, 2017.

[13] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[14] I. Good and R. Gaskins. Density estimation and bump-hunting by the penalized likelihood method exemplified by scattering and meteorite data. *Journal of the American Statistical Association*, 75(369):42–56, 1980.

[15] A. Gupta, J. Könemann, S. Leonardi, R. Ravi, and G. Schäfer. An efficient cost-sharing mechanism for the prize-collecting Steiner forest problem. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1153–1162. Society for Industrial and Applied Mathematics, 2007.

[16] M. T. Hajiaghayi and K. Jain. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete Algorithm*, pages 631–640. Society for Industrial and Applied Mathematics, 2006.

[17] K. Han, Y. He, X. Xiao, S. Tang, F. Gui, C. Xu, and J. Luo. Budget-constrained organization of influential social events. In *2018 IEEE 34th International Conference on Data Engineering*, pages 917–928. IEEE, 2018.

[18] K. Han, C. Zhang, and J. Luo. Taming the uncertainty: Budget limited robust crowdsensing through online learning. *IEEE/ACM Transactions on Networking*, 24(3):1462–1475, 2016.

[19] N. E. Heckman. Bump hunting in regression analysis. *Statistics & probability letters*, 14(2):141–152, 1992.

[20] C. Hegde, P. Indyk, and L. Schmidt. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem. In *Workshop of the 11th DIMACS Implementation Challenge*, 2014.

[21] C. Hegde, P. Indyk, and L. Schmidt. A nearly-linear time framework for graph-structured sparsity. In *International Conference on Machine Learning*, pages 928–937, 2015.

[22] A. E. Jaffe, P. Murakami, H. Lee, J. T. Leek, M. D. Fallin, A. P. Feinberg, and R. A. Irizarry. Bump hunting to identify differentially methylated regions in epigenetic epidemiology studies. *International journal of epidemiology*, 41(1):200–209, 2012.

[23] D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. In *Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

[24] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[25] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 467–476. ACM, 2009.

[26] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[27] P. Lin, Q. Song, and Y. Wu. Fact checking in knowledge graphs with ontological subgraph patterns. *Data Science and Engineering*, 3(4):341–358, 2018.

[28] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[29] S. M. Nikolakaki, C. Mavroforakis, A. Ene, and E. Terzi. Mining tours and paths in activity networks. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 459–468. International World Wide Web Conferences Steering Committee, 2018.

[30] J. Orear and D. Cassel. Applications of statistical inference to physics. *Foundations of Statistical inference*, pages 280–288, 1971.

[31] R. C. Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

[32] P. Rozenshtein, A. Anagnostopoulos, A. Gionis, and N. Tatti. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1176–1185, 2014.

[33] N. Samios. Current problems in experimental boson spectroscopy. In *AIP Conference Proceedings*, volume 8, pages 432–459. AIP, 1972.

[34] L. Schmidt, C. Hegde, P. Indyk, L. Lu, X. Chi, and D. Hohl. Seismic feature extraction using Steiner tree methods. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1647–1651. IEEE, 2015.

[35] Y. Sharma, C. Swamy, and D. P. Williamson. Approximation algorithms for prize collecting forest problems with submodular penalty functions. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete Algorithms*, pages 1275–1284. Society for Industrial and Applied Mathematics, 2007.

[36] Y. Sun, M. Brazil, D. Thomas, and S. Halgamuge. The fast heuristic algorithms and post-processing techniques to design large and low-cost communication networks. *IEEE/ACM Transactions on Networking*, 27(1):375–388, 2019.

[37] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[38] G. L. Trigg. Rules for "bump hunting". *Physical Review Letters*, 25(12):783, 1970.

[39] S. Vijayanarasimhan and K. Grauman. Efficient region search for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1401–1408, 2011.

[40] Y. Wang, Y. Yuan, Y. Ma, and G. Wang. Time-dependent

graphs: Definitions, applications, and algorithms. *Data Science and Engineering*, 4(4):352–366, 2019.

[41] S. N. Wood. *Generalized additive models: an introduction with R*. Chapman and Hall/CRC, 2017.

[42] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: on free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.