Hunting Temporal Bumps in Graphs with Dynamic Vertex Properties

Yahui Sun[1*]
Renmin University of China
Beijing, China
yahuisun@ruc.edu.cn

Shuai Ma
SKLSDE Lab, Beihang University
Beijing, China
mashuai@buaa.edu.cn

Bin Cui[2]
Peking University
Beijing, China
bin.cui@pku.edu.cn

ABSTRACT

Given a time interval and a graph where vertices exhibit a property of interest (PoI) dynamically, an interesting question is: where (*i.e.,* which part of the graph) and when (*i.e.,* which time sub-interval) does the PoI occur frequently? To our knowledge, no work has been done to answer this question to date. We address this issue in this paper. Specifically, given (i) a time interval composed of multiple time slots and (ii) a graph where each vertex either exhibits or does not exhibit the PoI in each time slot, our objective is to find a pair of a connected sub-graph and a time sub-interval (which we refer to as a temporal bump), such that the discrepancy between the numbers of times that vertices in this sub-graph exhibit and do not exhibit the PoI during this time sub-interval is maximized. Due to the NP-hardness of this problem, initially, we propose two approximation algorithms. The first one achieves a tight approximation guarantee, at the cost of a weak scalability to the number of time slots. The second one achieves a strong scalability to the number of time slots, at the price of a loose approximation guarantee. Then, we propose two heuristic algorithms that have no non-trivial approximation guarantee, but produce similar solutions with, and are considerably faster than, the two approximation algorithms. Experiments on real datasets show that, in comparison with baselines built using related existing techniques, our algorithms hunt bumps with significantly higher discrepancies, while scaling well to large graphs, and thus are more suitable for answering the aforementioned question.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**.

KEYWORDS

Data mining; graph mining; bump hunting; Steiner trees

ACM Reference Format:
Yahui Sun[1*], Shuai Ma, and Bin Cui[2]. 2022. Hunting Temporal Bumps in Graphs with Dynamic Vertex Properties. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517859

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517859

1 School of Information & Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education
2 School of CS & National Engineering Laboratory for Big Data Analysis and Applications, Peking University; Institute of Computational Social Science, Peking University (Qingdao)

* Yahui Sun is the corresponding author

1 INTRODUCTION

Graph mining (*e.g.,* [20]), which is about discovering knowledge from graph-structured datasets, plays an important role in the topic of data management. In practice, we often have a graph where some vertices exhibit a property of interest dynamically. For example, in a Wikipedia graph where vertices and edges represent Wikipedia pages and close relations between Wikipedia pages respectively, some pages may be viewed a large number of times in some hours. Given such a graph and a time interval composed of multiple time slots, *e.g.,* multiple hours, we consider a vertex as *queried* in a time slot if this vertex exhibits a property of interest in this time slot. We further consider the number of times that a vertex is queried or not queried in a time interval as the number of time slots in this interval in which this vertex is queried or not queried. Then, an intuitive problem is to find a pair of a connected sub-graph and a time sub-interval that contains as many queried states of vertices as possible, and as few not queried states of vertices as possible, that is to say, the discrepancy between the numbers of times that vertices in this sub-graph are queried and not queried during this time sub-interval is maximized. We refer to such a pair of a connected sub-graph and a time sub-interval as a *temporal bump*, and refer to this problem as the *temporal bump hunting* problem.

Solving this problem is to identify where and when the property of interest occurs frequently. For instance, in the Wikipedia graph where the property of interest is a large number of page views, solving this problem is to identify a cluster of closely related pages and a time sub-interval such that these pages are intensively viewed during this time sub-interval. An example is as follows. Consider the Wikipedia bump in Figure 1, where a Wikipedia page is queried in an hour if this page is viewed a large number of times in this hour; and the queried states of pages are highlighted in green. The connection of sub-graph guarantees that the identified pages are closely related. The discrepancy maximization objective guarantees that these pages are frequently queried, *i.e.,* intensively viewed, during the identified time sub-interval. This bump shows that (i) people are paying intensive attention to the US-Iran conflict in January 2020, of which a major flash point occurred around 1 AM on 3rd January 2020, when US President Donald Trump approved the targeted killing of Iranian Major General Qasem Soleimani in Baghdad [3]; and (ii) except the topic of "Qasem Soleimani" that directly corresponds to the conflict, people are beginning to pay attention to some related topics shortly after this conflict, like "Iran

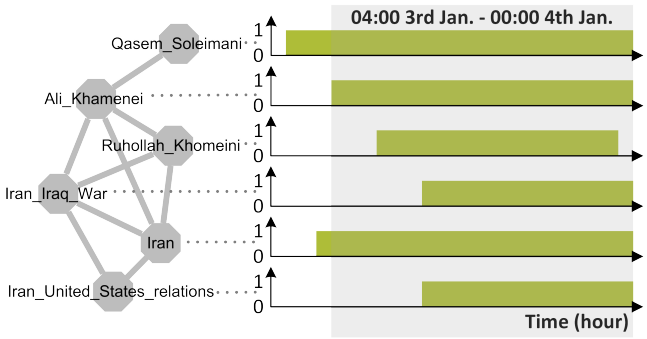


Figure 1: A temporal bump, comprising a sub-graph and a time sub-interval, in Wikipedia. The sub-graph contains 6 Wikipedia pages. The time sub-interval is between 04:00 3rd and 00:00 4th January 2020. These pages are intensively viewed (highlighted in green) during this time sub-interval.

Iraq War". Mining this knowledge could contribute to actionable intelligence, *e.g.,* to help medias make decisions on producing TV shows on Iran Iraq War after talking about the US-Iran conflict.

The graph information is essential in the above case. For example, if we simply identify Wikipedia pages that are viewed intensively during the time sub-interval of the above bump, and do not consider the relations between pages, we may get a list of dozens of pages, with the six pages in Figure 1 scattered in the list, without knowing that these six pages are closely related and collectively correspond to our intensive attention to the US-Iran conflict, and thus could not recommend medias to produce TV shows on Iran Iraq War after talking about the conflict. The temporal information is also essential in the above case, *e.g.,* the fact that the bump in Figure 1 starts at 04:00, shortly after the killing of Qasem Soleimani, helps analyze that this bump corresponds to the US-Iran conflict; and the fact that this bump lasts to the end of the input time interval helps analyze that our intensive attention to the conflict is ongoing, which is a valuable information to medias. The existing work on event detection does not suit the above case, since most such work focuses on non-graph-structured datasets (*e.g.,* [12, 28, 36, 53, 67]), while the other work that focuses on graph-structured datasets either does not consider the temporal information (*e.g.,* [47, 56, 68]), or only suits edge-evolving graphs where event-related activities are associated with edges (*e.g.,* [11, 49, 57]). Thus, solving the temporal bump hunting problem is particularly useful in analyzing graphs with dynamic vertex properties in the above case.

To our knowledge, no work has been done to solve the temporal bump hunting problem to date. The existing work on temporal graphs or dynamic networks performs different tasks, such as spotting anomalous sub-graphs with large dynamic edge weights (*e.g.,* [17, 19, 48, 49]), identifying temporal reach-abilities (*e.g.,* [41, 66]) or shortest paths (*e.g.,* [65, 69]) between vertices, discovering evolving communities and their life cycles (*e.g.,* birth, growth, and death of communities [23, 51, 55]), and temporal motif mining, *i.e.,* detecting sub-graphs with specific temporal patterns (*e.g.,* with specific appearance orders of dynamic edges [37, 46, 52, 64], or with bursting densities [21]). The above work does not identify where and when the property of interest occurs frequently in cases where vertices exhibit the property of interest dynamically, and thus does not rule out the particular usefulness of hunting temporal bumps.

Meanwhile, the existing researches on *static bump hunting* (*e.g.,* [10, 30, 32, 34, 35, 38, 62]) are closely related to our work. The topic of static bump hunting originated in the field of high energy physics half a century ago (*e.g.,* [50, 58]), and is to find regions of static datasets where a property of interest occurs frequently. Most existing researches on static bump hunting target non-graph-structured datasets (*e.g.,* [10, 30, 34, 35, 38]). Only some recent work aims graph-structured datasets (*e.g.,* [32, 62]). In particular, the problem studied by Gionis *et al.* [32] can be seen as a static version of the temporal bump hunting problem. That is to say, they consider a graph where each vertex is either queried or not queried, regardless of the time, and their objective is to find a connected sub-graph, *i.e.,* a *static bump*, where the discrepancy between the numbers of queried and not queried vertices is maximized. We refer to their problem as the static bump hunting problem.

The existing algorithms [32] for solving the static bump hunting problem do not suit hunting temporal bumps. We explain this as follows. Due to the neglect of dynamic query states of vertices, these algorithms can only hunt static bumps in a single time slot. We cannot use these algorithms to hunt temporal bumps by simply hunting static bumps in a time slot by time slot way, since static bumps in different time slots may not share vertices with each other, and as a result we cannot combine static bumps in different time slots together as a temporal bump. An intuitive idea of adapting these algorithms to hunt a temporal bump is to (i) hunt a static bump, *i.e.,* a connected sub-graph, and (ii) find a time sub-interval such that the pair of this sub-graph and this time sub-interval maximizes the objective value of the temporal bump hunting problem. Then, this pair of sub-graph and time sub-interval is the hunted temporal bump. The baselines in the later experiments are such adaptations. Theoretically, these adaptations cannot achieve non-trivial guarantees of solution qualities for hunting temporal bumps. Practically, the later experiments show that these adaptations cannot hunt high-discrepancy temporal bumps. As a result, new algorithms are required to meet the challenge of hunting temporal bumps.

Contributions. Specifically, since it is NP-hard to solve the temporal bump hunting problem to optimality, we develop several non-exact algorithms that find sub-optimal solutions as follows.

- We propose an approximation algorithm: MIRROR (Section 3.2). It solves the temporal bump hunting problem by (i) compressing dynamic query states of vertices during every time sub-interval to static values; and (ii) solving a static Steiner tree problem [42] for these values during every time sub-interval. Since it is too slow to conduct this process, we develop new branch and bound techniques that effectively accelerate this process, via which MIRROR achieves an approximation guarantee of 2 for a minimization objective that is equivalent to the discrepancy maximization objective of the temporal bump hunting problem.
- MIRROR does not scale well to the number of time slots, since it computes all time sub-intervals, while the number of time sub-intervals increases quadratically with the number of time slots. To overcome this weakness, we propose another approximation algorithm: S-MIRROR (Section 3.3), the idea of which is to select and compute a nearly linear number of time sub-intervals. By doing this, S-MIRROR achieves non-trivial approximation guarantees looser than those of MIRROR.

- To further push the limit of algorithmic efficiency while maintaining a practically high solution quality, we ignore non-trivial approximation guarantees, and propose two fast heuristic algorithms: H-MIRROR and H-S-MIRROR (Section 4). Motivated by the observation that real bumps mostly contain queried vertices (*i.e.,* vertices that have been queried at once during the given time interval), the idea of H-MIRROR and H-S-MIRROR is simple and effective: instead of computing all vertices, only compute queried vertices and their close neighbors.

We conduct experiments using real datasets (Section 5). In comparison with baselines built using state-of-the-art static bump hunting algorithms, all the proposed algorithms hunt bumps with significantly (sometimes an order of magnitude) higher discrepancies. Meanwhile, even though MIRROR and S-MIRROR are slower than some baselines, H-MIRROR and H-S-MIRROR are generally faster than the baselines. Some other observations are: (i) MIRROR scales nearly quadratically to the number of time slots, and often hunts bumps with at least one third of the maximum discrepancies; (ii) S-MIRROR scales nearly linearly to the number of time slots, and often hunts bumps with similar discrepancies with MIRROR; and (iii) H-MIRROR and H-S-MIRROR hunt bumps with similar discrepancies with MIRROR and S-MIRROR, while being considerably faster than MIRROR and S-MIRROR in various cases.

2 PROBLEM FORMULATION

Given a time interval $T = [t_1, t_m]$ of m continuous time slots, we consider an undirected graph $G(V, E, \eta)$, where V is the set of vertices, E is the set of edges, and η is a function which maps each vertex $v \in V$ to a set of m Boolean values $\eta(v) = \{\eta^{t_1}(v), \cdots, \eta^{t_m}(v)\}$ that correspond to m time slots in T, and are referred to as *query states* of v. For time slot $t_x \in T$ ($x \in [1, m]$), if $\eta^{t_x}(v) = 1$, then v is queried in t_x, which means that v exhibits a property of interest in t_x, otherwise v is not queried in t_x, *i.e.*, $\eta^{t_x}(v) = 0$. These dynamic query states of vertices reflect the fact that vertices often exhibit a property of interest dynamically in real graphs.

We refer to a connected sub-graph of G as a *component* of G. In the previous work [32], a *static bump* is defined as a component. Here, due to the involvement of the temporal dimension, we extend the above previous work, and define a *temporal bump* as the pair of a component and a time sub-interval, *e.g.,* the pair of a component $C(V_C, E_C)$ of G and a time sub-interval $T_S \subseteq T$.

Definition 1 (Temporal Bump). *Given a time interval $T = [t_1, t_m]$ and a graph $G(V, E, \eta)$, a temporal bump is the pair of a component $C(V_C, E_C)$ of G and a time sub-interval $T_S \subseteq T$.*

Given a temporal bump $\{C, T_S\}$, we refer to $p_C^{T_S}$ as the number of times that vertices in C have been queried during T_S, *i.e.*,

$$p_C^{T_S} = \sum_{v \in V_C, t_x \in T_S} \eta^{t_x}(v). \tag{1}$$

Similarly, we refer to $n_C^{T_S}$ as the number of times that vertices in C have not been queried during T_S. Since each vertex has $|T_S|$ query states during time sub-interval T_S, we have

$$n_C^{T_S} = |T_S||V_C| - p_C^{T_S}. \tag{2}$$

We define the *temporal discrepancy* of C during T_S, *i.e.,* the temporal discrepancy of the temporal bump $\{C, T_S\}$, as follows.

Definition 2 (Temporal Discrepancy). *Given a time interval $T = [t_1, t_m]$ and a graph $G(V, E, \eta)$, the temporal discrepancy of a component $C(V_C, E_C)$ of G during a time sub-interval $T_S \subseteq T$ is*

$$D_C^{T_S} = p_C^{T_S} - n_C^{T_S}. \tag{3}$$

This temporal discrepancy is a natural extension of the *static discrepancy* of C in the previous work [32], which is the discrepancy between the numbers of statically queried and not queried vertices in C. Like the previous work, we can also use a parameter $\alpha > 0$ to regulate $p_C^{T_S}$ and $n_C^{T_S}$ in $D_C^{T_S}$, *i.e.,* to define $D_C^{T_S}$ as $\alpha p_C^{T_S} - n_C^{T_S}$. Since $p_C^{T_S}$ and $n_C^{T_S}$ have the same measurement unit, it is intuitively preferable to set $\alpha = 1$ in practice [32]. As a result, to reduce the complexity of the problem setting and achieve an easy use of our work, we omit α in this paper. Nevertheless, we show the feasibility of using α in Section S2 in the supplement [6].

A temporal bump with a high temporal discrepancy corresponds to a region of G and a time sub-interval such that the property of interest occurs frequently in this region during this time sub-interval. Like the previous work [32], we define the temporal bump hunting problem as a discrepancy maximization problem as follows.

Problem 1 (Temporal Bump Hunting). *Given a time interval $T = [t_1, t_m]$ and a graph $G(V, E, \eta)$, the temporal bump hunting problem is to find the pair of a component $C(V_C, E_C)$ of G and a time sub-interval $T_S \subseteq T$, i.e., a temporal bump $\{C, T_S\}$, such that its temporal discrepancy $D_C^{T_S}$ is the maximum.*

Solving Problem 1 is to identify *where* and *when* the property of interest occurs frequently. The discrepancy maximization objective is to ensure that the hunted bump contains as many queried states as possible, and as few not queried states as possible, *i.e.,* the hunted bump exhibits the property of interest intensively. The connection of C is to ensure that the entities represented by the vertices in C are closely related, which is meaningful in various cases, *e.g.,* the connection of sub-graph in Figure 1 ensures that the identified Wikipedia pages are closely related and collectively correspond to our intensive attention to a specific hot topic.

The static bump hunting problem [32] is about finding a component C such that the α-regulated static discrepancy of C is maximized. Thus, the static bump hunting problem with the restriction of $\alpha = 1$ is a special case of Problem 1 where $m = 1$. The previous work [32] proves that the static problem is NP-hard when α is not restricted to 1. In the supplement [6], we prove that the static problem is NP-hard even when α is restricted to 1. Since Problem 1 is a generalization of this restricted static case, Problem 1 is NP-hard. This NP-hardness indicates the preference of developing non-exact algorithms in practice. We develop such algorithms as follows.

3 TWO APPROXIMATION ALGORITHMS

In this section, we develop two approximation algorithms, dubbed MIRROR and S-MIRROR respectively, for hunting temporal bumps. First, in Section 3.1, we transform Problem 1 to a temporal Steiner tree problem. Then, in Sections 3.2 and 3.3, we propose MIRROR and S-MIRROR respectively to solve this Steiner tree problem.

3.1 A temporal Steiner tree problem

Here, we first formulate the temporal prize-collecting Steiner tree problem, and then demonstrate the transformation from Problem

1 to this problem. In this problem, we consider a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, where ξ is a function which maps each vertex $v \in V$ to a set of m nonnegative values $\xi(v) = \{\xi^{t_1}(v), \cdots, \xi^{t_m}(v)\}$ that correspond to m time slots in T, and are referred to as *vertex prizes*, and similarly, c is a function which maps each edge $e \in E$ to a set of m nonnegative values $c(e) = \{c^{t_1}(e), \cdots, c^{t_m}(e)\}$ that are referred to as *edge costs*. We present the temporal prize-collecting Steiner tree problem as follows.

Problem 2 (Temporal Prize-Collecting Steiner Tree). *Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, the temporal prize-collecting Steiner tree problem is to find a tree $\Theta(V_\Theta, E_\Theta)$ of G' and a time sub-interval $T_S \subseteq T$, i.e., the combination of $\{\Theta, T_S\}$, such that the weight of this combination, namely,*

$$w_{T_S}(\Theta) = \sum_{v \in V_\Theta, t_x \in T_S} \xi^{t_x}(v) - \sum_{e \in E_\Theta, t_x \in T_S} c^{t_x}(e) - |T_S| \tag{4}$$

is maximized, or the cost of this combination, namely,

$$c_{T_S}(\Theta) = \sum_{e \in E_\Theta, t_x \in T_S} c^{t_x}(e) + \sum_{v \in V \setminus V_\Theta, t_x \in T_S} \xi^{t_x}(v) + \sum_{v \in V, t_x \in T \setminus T_S} \xi^{t_x}(v) + |T_S| \tag{5}$$

is minimized.

For a given time interval T and a given graph $G'(V, E, \xi, c)$, the sum of vertex prizes, *i.e.*, $\sum_{v \in V, t_x \in T} \xi^{t_x}(v)$, is constant. We have

$$w_{T_S}(\Theta) + c_{T_S}(\Theta) = \sum_{v \in V, t_x \in T} \xi^{t_x}(v). \tag{6}$$

Thus, the above two objectives, *i.e.,* the maximization of $w_{T_S}(\Theta)$ and the minimization of $c_{T_S}(\Theta)$, are equivalent. The static prize-collecting Steiner tree problem [42] can be seen as a special case of Problem 2 where $m = 1$. Like the static problem, Problem 2 is NP-hard. We transform Problem 1 to Problem 2 as follows.

Theorem 1. *Consider a time interval $T = [t_1, t_m]$; a graph $G(V, E, \eta)$; and a graph $G'(V, E, \xi, c)$. If*

$$\xi^{t_x}(v) = 2\eta^{t_x}(v) \mid \forall v \in V, t_x \in T, \tag{7}$$

$$c^{t_x}(e) = 1 \mid \forall e \in E, t_x \in T, \tag{8}$$

then, for any time sub-interval $T_S \subseteq T$; any component $C(V_C, E_C)$ of G; and any tree $\Theta(V_\Theta, E_\Theta)$ of G' that has the same set of vertices with C, i.e., $V_\Theta = V_C$, we have

$$D_C^{T_S} = w_{T_S}(\Theta), \tag{9}$$

which means that any approximation guarantee (including the optimal guarantee) that holds for maximizing $w_{T_S}(\Theta)$ also holds for maximizing $D_C^{T_S}$, and vise versa.

We put the detailed proof in Section S1 in the supplement [6]. Based on this theorem, we can solve Problem 1 in G by solving Problem 2 in G'. In the following sub-sections, we develop two approximation algorithms to solve Problem 2 in G'.

3.2 The MIRROR algorithm

In this sub-section, we develop MIRROR, *i.e.,* the time sub-interval enumerating algorithm, for solving Problem 2 in G'.

First, given a time sub-interval T_S and a graph $G'(V, E, \xi, c)$, we define the *aggregated graph* of G' during T_S as a static graph $G'_{T_S}(V, E, w_s, c_s)$, where w_s is a function which maps each vertex $v \in V$ to a nonnegative value $w_s(v)$, and c_s is a function which maps each edge $e \in E$ to a nonnegative value $c_s(e)$, and

$$w_s(v) = \sum_{t_x \in T_S} \xi^{t_x}(v) \mid \forall v \in V, \tag{10}$$

$$c_s(e) = \sum_{t_x \in T_S} c^{t_x}(e) \mid \forall e \in E, \tag{11}$$

i.e., the aggregated vertex prizes and edge costs during T_S.

For a static graph $G'_{T_S}(V, E, w_s, c_s)$, the static prize-collecting Steiner tree problem [42] is to find a tree $\Theta(V_\Theta, E_\Theta)$ in G'_{T_S} such that the net-weight of this tree, namely,

$$w_{G'_{T_S}}(\Theta) = \sum_{v \in V_\Theta} w_s(v) - \sum_{e \in E_\Theta} c_s(e) \tag{12}$$

is maximized. For any time sub-interval $T_S \subseteq T$, we observe that

$$w_{T_S}(\Theta) = w_{G'_{T_S}}(\Theta) - |T_S|. \tag{13}$$

This means that any tree Θ that maximizes $w_{G'_{T_S}}(\Theta)$ also maximizes $w_{T_S}(\Theta)$. Thus, we can solve the temporal prize-collecting Steiner tree problem in G' by (i) enumerating every time sub-interval $T_S \subseteq T$; (ii) finding the solution tree Θ to the static prize-collecting Steiner tree problem in G'_{T_S}; and (iii) returning $\{\Theta, T_S\}$ that maximizes $w_{T_S}(\Theta)$ as the solution to the temporal prize-collecting Steiner tree problem in G' during T. However, it is too slow to conduct the above process in practice. To address this issue, MIRROR incorporates newly developed techniques based on the classic branch and bound idea [44] for accelerating the above process.

Description of MIRROR**.** Algorithm 1 demonstrates the pseudo code of MIRROR. Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, the algorithm first sorts all time sub-intervals $\Phi = \{T_i \mid \forall T_i \subseteq T\}$ from large to small based on their ranges (Line 1). The reason of sorting time sub-intervals in this order is that MIRROR later dynamically prunes un-profitable small time sub-intervals based on the computing results in large time sub-intervals.

Then, the algorithm initializes an empty tree Θ_M, an empty time sub-interval T_M, and consider $w_{T_M}(\Theta_M) = -\infty$, and also initializes an empty hash table P for storing pruned time sub-intervals (Line 2). MIRROR uses $\{\Theta_M, T_M\}$ to store the best found solution.

Subsequently, it reduces G' using the Degree-0-1-2 test as follows (Line 3). It associates each vertex $v \in V$ with a Boolean value $w_r(v)$. If $\xi^{t_x}(v) = 0$ for all $t_x \in T$, then $w_r(v) = 0$. Otherwise, $w_r(v) = 1$. It associates each edge $e \in E$ with a value $c_r(e) = 1$. If vertex v has a degree of 0 or 1 and $w_r(v) = 0$, then it removes v from G', since v is not in an optimal solution. If vertex v has a degree of 2 and $w_r(v) = 0$, then it removes v from G' in the following way. Let j and k be the adjacent vertices of v. If edge (j, k) is not in G', then it adds (j, k) into G', and sets $c_r(j, k) = c_r(v, j) + c_r(v, k)$, and removes v from G', and uses a hash to record that (j, k) is a merge of (v, j) and (v, k). If edge (j, k) is in G' and $c_r(j, k) > c_r(v, j) + c_r(v, k)$, then it updates $c_r(j, k) = c_r(v, j) + c_r(v, k)$, and removes v from G', and uses a hash to record that (j, k) is a merge of (v, j) and (v, k). If edge (j, k) is in G' and $c_r(j, k) \leq c_r(v, j) + c_r(v, k)$, then it simply removes v from G'. It conducts this reduction process until no vertex can be removed any more.

After reducing G', MIRROR conducts a depth first search to mark maximum connected components of G' (Line 4). Then, it

Algorithm 1 The MIRROR algorithm

Input: a time interval $T = [t_1, t_m]$, a graph $G'(V, E, \xi, c)$
Output: a tree Θ_M and a time sub-interval T_M

1: Sort $\Phi = \{T_i \mid \forall\, T_i \subseteq T\}$ from large to small
2: Initialize $\Theta_M = \emptyset$, $T_M = \emptyset$, $w_{T_M}(\Theta_M) = -\infty$, $P = \emptyset$
3: Reduce G' via Degree-0-1-2 test
4: Mark maximum connected components of G'
5: **for** each (sorted) $T_i \in \Phi$ **do**
6: **if** $T_i \notin P$ **then**
7: Build $G'_{T_i}(V, E, w_s, c_s)$
8: **if** $\zeta_{T_i} - |T_i| \leq w_{T_M}(\Theta_M)$ **then**
9: **if** $\zeta_{T_i} \leq w_{T_M}(\Theta_M)$ **then**
10: $P = P \cup \{T_j \mid \forall\, T_j \subseteq T_i\}$
11: **end if**
12: Continue \\ Skip to Line 5
13: **end if**
14: **if** $UB_{T_i} \leq w_{T_M}(\Theta_M)$ **then**
15: Continue \\ Skip to Line 5
16: **end if**
17: $\Theta_{1T_i}(V_{1T_i}, E_{1T_i}) = FastGrowingTree(G'_{T_i})$
18: $\Theta_{2T_i}(V_{2T_i}, E_{2T_i}) = GeneralPrunning(\Theta_{1T_i})$
19: **if** $w_{T_i}(\Theta_{2T_i}) > w_{T_M}(\Theta_M)$ **then**
20: $\Theta_M = \Theta_{2T_i}$, $T_M = T_i$
21: **end if**
22: **end if**
23: **end for**
24: Return Θ_M and T_M

enumerates every time sub-interval $T_i \in \Phi$ from large to small (Line 5). If T_i has not been pruned, *i.e.*, $T_i \notin P$ (Line 6), it builds the aggregated graph $G'_{T_i}(V, E, w_s, c_s)$ (Line 7). Different from $c_s(e)$ in Equation (11), here, for every edge $e \in E$,

$$c_s(e) = c_r(e) \cdot |T_i|, \tag{14}$$

since $c_r(e)$ edges in the original G' are merged together as e during the above Degree-0-1-2 test in Line 3, and by Theorem 1, $c^{t_x}(e) = 1$ for every $e \in E$ and $t_x \in T$ in the temporal bump hunting case.

To enhance the efficiency, MIRROR employs newly developed techniques based on the classic branch and bound idea [44] as follows (Lines 8-16). We refer to ζ_{T_i} as the maximum value of the sum of aggregated vertex prizes (w_s) in a maximum connected component of G'_{T_i}. The best solution we can find in G'_{T_i} has a weight not larger than $\zeta_{T_i} - |T_i|$ (see Equation (4)). If $\zeta_{T_i} - |T_i| \leq w_{T_M}(\Theta_M)$ (Line 8), then MIRROR continues the loop without solving the static prize-collecting Steiner tree problem in G'_{T_i} (Line 12), since it cannot find a combination of a tree and T_i that has a larger weight than the best found solution $\{\Theta_M, T_M\}$. Before continuing the loop, if $\zeta_{T_i} \leq w_{T_M}(\Theta_M)$ (Line 9), then it prunes all time sub-intervals in T_i, *i.e.,* it pushes these time sub-intervals into P (Line 10), since it cannot find a combination of a tree and $T_j \mid \forall\, T_j \subseteq T_i$ that has a larger weight than $\{\Theta_M, T_M\}$.

MIRROR further employs the fact that $c_s(e) \geq |T_i|$ for every edge $e \in E$ to compute a newly discovered upper bound of the best solution weight that it can obtain in G'_{T_i} (Lines 14-16), for further enhancing the efficiency. The details are as follows. For a maximum connected component of G'_{T_i}: $C_x(V_{C_x}, E_{C_x})$, let $SUM_{T_i\_C_x}$ be the sum of aggregated vertex prizes in C_x that are larger than $|T_i|$, *i.e.,*

$$SUM_{T_i\_C_x} = \sum_{v \in V_{C_x}, w_s(v) > |T_i|} w_s(v). \tag{15}$$

Moreover, let NUM_{C_x} be the number of aggregated vertex prizes in C_x that are larger than $|T_i|$, *i.e.,*

$$NUM_{T_i\_C_x} = \sum_{v \in V_{C_x}, w_s(v) > |T_i|} 1. \tag{16}$$

We set

$$UB_{T_i\_C_x} = SUM_{T_i\_C_x} - NUM_{T_i\_C_x} \cdot |T_i|. \tag{17}$$

Let UB_{T_i} be the maximum value of $UB_{T_i\_C_x}$ for any C_x, *i.e.,* any maximum connected component of G'_{T_i}. Since $c^{t_x}(e) = 1$ for every $e \in E$ and $t_x \in T$ in the bump hunting case, we have Theorem 2, the proof of which is in Section S1 in the supplement [6].

Theorem 2. *Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, for any tree $\Theta(V_\Theta, E_\Theta)$ of G' and any time sub-interval $T_i \subseteq T$, we have*

$$UB_{T_i} \geq w_{T_i}(\Theta). \tag{18}$$

Theorem 2 shows that UB_{T_i} is an upper bound of the best solution weight that MIRROR can obtain in G'_{T_i}. If $UB_{T_i} \leq w_{T_M}(\Theta_M)$ (Line 14), then MIRROR continues the loop without solving the static prize-collecting Steiner tree problem in G'_{T_i} (Line 15), since it cannot find a solution in G'_{T_i} that is better than $\{\Theta_M, T_M\}$.

If the above process does not rule out the possibility of finding a better solution than $\{\Theta_M, T_M\}$ in G'_{T_i}, then MIRROR employs the Goemans-Williamson approximation scheme [33] to solve the static prize-collecting Steiner tree problem in G'_{T_i}. Specifically, it first uses the fast implementation of the Goemans-Williamson growing algorithm [39] to produce a raw solution tree Θ_{1T_i} (Line 17), and then uses the general pruning algorithm [61] to prune this raw solution tree as Θ_{2T_i} (Line 18). The pruned tree is an approximate solution to the static prize-collecting Steiner tree problem in G'_{T_i}. If $w_{T_i}(\Theta_{2T_i}) > w_{T_M}(\Theta_M)$ (Line 19), then MIRROR updates Θ_M and T_M to be Θ_{2T_i} and T_i (Line 20). After the loop, MIRROR returns Θ_M and T_M as the final solution (Line 24). Note that, due to the reduction process (Line 3), Θ_M may contain edges that are not in the original G' but are merged by edges in the original G'. Thus, it is required to use the recorded merging information to restore Θ_M, for guaranteeing that Θ_M is a tree in the original G'.

Approximation guarantees of MIRROR**.** MIRROR employs the Goemans-Williamson approximation scheme [33] to solve a static prize-collecting Steiner tree instance for every time sub-interval. Via this process, MIRROR extends the above scheme to temporal scenarios, and achieves the following approximation guarantees.

Theorem 3. MIRROR *has an approximation guarantee of 2 with respect to minimizing $c_{T_S}(\Theta)$ for solving the temporal prize-collecting Steiner tree problem.*

The proof of this theorem is in Section S1 in the supplement [6]. The above 2 ratio does not translate into a constant approximation ratio for maximizing $D_C^{T_S}$, since Theorem 1 shows that maximizing $D_C^{T_S}$ is translated into maximizing $w_{T_S}(\Theta)$, not minimizing $c_{T_S}(\Theta)$.

The previous work [29] indicates that it is NP-hard to approximately maximize $w_{T_S}(\Theta)$ within any constant ratio. With this in mind, we present the following theorem to show the approximation guarantee of MIRROR with respect to maximizing $w_{T_S}(\Theta)$.

Theorem 4. *Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, let $\{\Theta(V_\Theta, E_\Theta), T_S\}$ be an optimal solution to the temporal prize-collecting Steiner tree problem, and let $\{\Theta_M(V_M, E_M), T_M\}$ be the solution of* MIRROR*, then*

$$2w_{T_M}(\Theta_M) + L \geq 2w_{T_S}(\Theta), \tag{19}$$

where

$$L = \max\{\sum_{e \in E_{2T_i}, t_x \in T_i} c^{t_x}(e) \mid \forall\, T_i \in \Phi\}, \tag{20}$$

and E_{2T_i} and Φ are in the process of MIRROR *(if Line 18 is not executed for T_i due to the branch and bound process, then we consider $E_{2T_i} = \emptyset$).*

The proof of Theorem 4 is also in the supplement [6]. Theorem 4 implies an upper bound of the optimal solution weight $w_{T_S}(\Theta)$:

$$w_{T_M}(\Theta_M) + \frac{L}{2}.$$

The ratio of $w_{T_M}(\Theta_M)$ to this upper bound can be seen as a non-constant worst case approximation ratio of $w_{T_M}(\Theta_M)$.

Time complexity of MIRROR**:**

$$O\Big(m^2|V| + m^2 d|E|\log|V| + m^2|\cup v_{pos}| + m^3\Big),$$

where d is the precision of vertex prizes and edge costs (details in [39]), and $|\cup v_{pos}|$ is the number of positive vertex prizes, which is at most $m|V|$. Due to space limitation, we put the details of the above time complexity in Section S4 in the supplement [6].

3.3 The S-MIRROR algorithm

The above MIRROR does not have a strong scalability to the number of time slots: m, since it computes all time sub-intervals, while the total number of time sub-intervals with respect to m is quadratic. To address this issue, here, we develop S-MIRROR, *i.e.,* the selected time sub-interval enumerating algorithm, for solving Problem 2 in G'. Unlike MIRROR that computes all time sub-intervals, S-MIRROR only computes some selected time sub-intervals.

Description of S-MIRROR**.** Algorithm 2 shows the pseudo code of S-MIRROR. Like MIRROR, S-MIRROR inputs a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$. When $m = 1$, it selects the single time sub-interval. When $m \geq 2$, it selects time sub-intervals $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$, where Ω_1, Ω_2 and Ω_3 are defined as follows.

$$\begin{aligned}\Omega_1 = \{&[t_1, t_{2^i}], [t_{2^i}, t_{2^{i+1}-1}], \cdots, [t_\tau, t_m]\\ , &[t_{m-(2^i-1)}, t_m], [t_{m-2(2^i-1)}, t_{m-(2^i-1)}], \cdots, [t_1, t_\varphi]\\ &\mid \forall\, 1 \leq i \leq \log_2 m, i \in \mathbb{N}\},\end{aligned} \tag{21}$$

where τ and φ are such integers that $m - \tau \leq 2^i - 1, \varphi - 1 \leq 2^i - 1$. That is to say, for each integer $i \in [1, \log_2 m]$, Ω_1 enumerates and includes adjacent time sub-intervals with a length of 2^i both from t_1 to t_m (*i.e.,* from $[t_1, t_{2^i}]$ to $[t_\tau, t_m]$) and from t_m to t_1 (*i.e.,* from $[t_{m-(2^i-1)}, t_m]$ to $[t_1, t_\varphi]$). To fully cover T in the above enumerations, the lengths of $[t_\tau, t_m]$ and $[t_1, t_\varphi]$ may be smaller than 2^i.

$$\begin{aligned}\Omega_2 = \{&[t_x, t_x], [t_x, t_{x+1}], \cdots, [t_x, t_y]\\ , &[t_y, t_y], [t_{y-1}, t_y], \cdots, [t_{x+1}, t_y] \mid \forall\, [t_x, t_y] \in \Omega_1\}.\end{aligned} \tag{22}$$

Algorithm 2 The S-MIRROR algorithm

Input: a time interval $T = [t_1, t_m]$, a graph $G'(V, E, \xi, c)$
Output: a tree Θ_{SM} and a time sub-interval T_{SM}

1: Select time sub-intervals $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$
2: Sort $\Phi = \Omega$ from large to small
3: $\{\Theta_M, T_M\}$ = Implement Lines 2-23 in MIRROR
4: Return $\Theta_{SM} = \Theta_M$ and $T_{SM} = T_M$

That is to say, for every time sub-interval $[t_x, t_y] \in \Omega_1, \Omega_2$ contains every time sub-interval that (i) is a part of $[t_x, t_y]$ and (ii) starts at t_x or ends at t_y. Let Υ be the set of time sub-intervals that are not in Ω_1 or Ω_2. Subsequently, Ω_3 contains $\min\{\overline{m\log_2 m}, |\Upsilon|\}$ time sub-intervals that are selected from Υ uniformly at random, where $\overline{m\log_2 m}$ is the smallest integer larger than or equal to $m\log_2 m$.

The number of selected time sub-intervals is

$$O\Big(|\Omega|\Big) = O\Big(m\log m\Big). \tag{23}$$

The deduction details of Equation (23) are simple and omitted. After selecting time sub-intervals in the above way (Line 1), S-MIRROR sorts the selected time sub-intervals from large to small based on their ranges (Line 2). It implements Lines 2-23 in MIRROR to produce tree Θ_M and time sub-interval T_M (Line 3). It returns Θ_M and T_M as the final solution (Line 4). It is possible to add a parameter $h \in \mathbb{N}$ into Ω_3, *i.e.,* to let Ω_3 contain $\min\{h \cdot \overline{m\log_2 m}, |\Upsilon|\}$ time sub-intervals that are selected from Υ uniformly at random. The experiment results in Section S5 in the the supplement [6] show that setting $h = 1$ gives S-MIRROR a high performance. As a result, for the easy use of S-MIRROR, we omit h in this paper.

Approximation guarantees of S-MIRROR**.** S-MIRROR select time sub-intervals in such a way that, for any time sub-interval $T_S = [t_a, t_c] \subseteq T$, there are two selected time sub-intervals $[t_a, t_b]$ and $[t_b, t_c]$ in Ω_2 such that $t_a \leq t_b \leq t_c$ (see Lemma 1 in Section S1 in the supplement [6]). By computing both $[t_a, t_b]$ and $[t_b, t_c]$ for any $T_S = [t_a, t_c] \subseteq T$, S-MIRROR achieves the following approximation guarantees.

Theorem 5. *Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$, let $\{\Theta(V_\Theta, E_\Theta), T_S\}$ be an optimal solution to the temporal prize-collecting Steiner tree problem, and let $\{\Theta_{SM}(V_{SM}, E_{SM}), T_{SM}\}$ be the solution of* S-MIRROR*, then*

$$2w_{T_{SM}}(\Theta_{SM}) + H + 1 \geq w_{T_S}(\Theta), \tag{24}$$

$$2c_{T_{SM}}(\Theta_{SM}) - H - 1 \leq \sum_{v \in V, t_x \in T} \xi^{t_x}(v) + c_{T_S}(\Theta), \tag{25}$$

where

$$\begin{aligned}H = \quad &\max\{\sum_{e \in E_\Theta} c^{t_b}(e) - \sum_{v \in V_\Theta} \xi^{t_b}(v) \mid \forall t_b \in T\}\\ &+ \max\{\kappa_1, \kappa_2\},\end{aligned} \tag{26}$$

where κ_1 is the maximum value of $\sum_{e \in E_{2T_i}, t_x \in T_i} c^{t_x}(e)$ for such $T_i \in \Omega_2$ that Line 18 of MIRROR *is executed, and κ_2 is the maximum value of ζ_{T_i} for such $T_i \in \Omega_2$ that Line 12 or 15 of* MIRROR *is executed.*

We put the proof of Theorem 5 in Section S1 in the supplement [6]. Based on Theorem 5, we can use the solution of S-MIRROR to produce an upper bound of the optimal solution weight $w_{T_S}(\Theta)$.

Before showing this upper bound, we propose a theorem as follows, which is based on the fact that each transformed vertex prize is either 0 or 2 in the temporal bump hunting case (see Equation (7)).

Theorem 6. *Given a time interval $T = [t_1, t_m]$ and a graph $G'(V, E, \xi, c)$ built via Theorem 1, let $\{\Theta(V_\Theta, E_\Theta), T_S\}$ be an optimal solution to the temporal prize-collecting Steiner tree problem. If there is at least one positive vertex prize in G' during T, then*

$$|E_\Theta| \leq \min\{|V| - 1, 2|V_{pos}| - 1 - \frac{1}{|T|}\}, \quad (27)$$

where V_{pos} is the set of vertices that have at least one positive prize during T, i.e., $V_{pos} = \{v \mid \forall v \in V, \sum_{t_x \in T} \xi^{t_x}(v) > 0\}$.

We put the proof of this theorem in Section S1 in the supplement [6]. Based on this theorem, when G' is built via Theorem 1, we have

$$\sum_{e \in E_\Theta} c^{t_b}(e) - \sum_{v \in V_\Theta} \xi^{t_b}(v) \leq \min\{|V| - 1, 2|V_{pos}| - 1 - \frac{1}{|T|}\}. \quad (28)$$

Then, by Equation (24), we can use the solution of S-MIRROR to produce an upper bound of $w_{T_S}(\Theta)$:

$$2w_{T_{SM}}(\Theta_{SM}) + \max\{\kappa_1, \kappa_2\} + \min\{|V|, 2|V_{pos}| - \frac{1}{|T|}\}.$$

Notably, $|V|$ in this upper bound can be the number of vertices in the graph reduced by Degree-0-1-2 test (see Line 3 of MIRROR). Like MIRROR, the ratio of $w_{T_{SM}}(\Theta_{SM})$ to the above bound can be seen as a non-constant worst case approximation ratio of $w_{T_{SM}}(\Theta_{SM})$.

Time complexity of S-MIRROR**:**

$$O\Big(m \log m \cdot |V| + m \log m \cdot d|E| \log |V| + m \log m \cdot |\cup v_{pos}| + m^3\Big).$$

The details of this time complexity are similar to those of MIRROR, since the only difference between MIRROR and S-MIRROR is that S-MIRROR selects and computes $O(m \log m)$ time sub-intervals.

4 TWO FAST HEURISTIC ALGORITHMS

To push the limit of algorithmic efficiency, here, we propose two fast heuristic algorithms for solving the temporal prize-collecting Steiner tree problem in G'. These two algorithms are dubbed H-MIRROR and H-S-MIRROR, respectively. H-MIRROR and H-S-MIRROR are different from MIRROR and S-MIRROR in that H-MIRROR and H-S-MIRROR only hunt bumps from sub-graphs constructed by vertices that have been queried at least once during T and close neighbors of these vertices, while ignoring the other parts of the input graph. That is to say, different from MIRROR and S-MIRROR that compute all vertices, H-MIRROR and H-S-MIRROR only compute queried vertices and their close neighbors.

This change is motivated by two observations: (i) most vertices in real high-discrepancy bumps have been queried at least once during the time interval T, as otherwise the discrepancies of these bumps would be low; and (ii) only a small part of vertices have ever exhibited a property of interest and been queried in many real scenarios (*e.g.*, in the Wikipedia graph, pages that are being viewed intensively are often pages that are related to hot topics, which are a fraction of all Wikipedia pages). Based on these observations, by only computing queried vertices and their close neighbors, we could enhance the algorithmic efficiency, while not sacrificing the practical solution quality. H-MIRROR and H-S-MIRROR are based on this simple but effective idea.

Algorithm 3 The H-MIRROR algorithm

Input: a time interval $T = [t_1, t_m]$, a graph $G'(V, E, \xi, c)$
Output: a tree Θ_{HM} and a time sub-interval T_{HM}

1: $G' = BreadthFirstSearch(G')$
2: $\{\Theta_M, T_M\} = \text{MIRROR}(T, G')$
3: Return $\Theta_{HM} = \Theta_M$ and $T_{HM} = T_M$

Algorithm 4 The H-S-MIRROR algorithm

Input: a time interval $T = [t_1, t_m]$, a graph $G'(V, E, \xi, c)$
Output: a tree Θ_{HSM} and a time sub-interval T_{HSM}

1: $G' = BreadthFirstSearch(G')$
2: $\{\Theta_{SM}, T_{SM}\} = \text{S} - \text{MIRROR}(T, G')$
3: Return $\Theta_{HSM} = \Theta_{SM}$ and $T_{HSM} = T_{SM}$

Description of H-MIRROR **and** H-S-MIRROR**.** Algorithm 3 shows the pseudo code of H-MIRROR. The algorithm first updates G' as follows (Line 1). For each vertex $v \in V$ such that $\sum_{t_x \in T} \xi^{t_x}(v) > 0$, H-MIRROR conducts a breadth first search starting from v, with a maximum searching depth of b (the depth of v is 0, and the depth of adjacent vertices of v is 1, *etc.*), where b is the minimum possible number of vertices between two queried vertices. After conducting the above breadth first searches, H-MIRROR updates G' to be the sub-graph that is constructed by all the searched vertices and all the edges between these vertices (Line 1). That is to say, the updated G' is constructed by queried vertices and their close neighbors. Then, the algorithm employs MIRROR to produce a feasible solution $\{\Theta_M, T_M\}$ (Line 2), and returns this solution (Line 3).

Algorithm 4 shows the pseudo code of H-S-MIRROR, which is different from H-MIRROR in that it employs S-MIRROR to produce a feasible solution $\{\Theta_{SM}, T_{SM}\}$ in the updated G' (Line 2).

Solution qualities and time complexities of H-MIRROR **and** H-S-MIRROR**.** Let $\{\Theta(V_\Theta, E_\Theta), T_S\}$ be an optimal solution. Let $V'_\Theta \in V_\Theta$ be the set of vertices in V_Θ such that, for each vertex $v \in V'_\Theta$, $\sum_{t_x \in T} \xi^{t_x}(v) > 0$. For any path in Θ that contains no vertex in V'_Θ, if there are at most $2b$ vertices in this path, then Θ is in the updated G' in H-MIRROR and H-S-MIRROR. In this case, H-MIRROR and H-S-MIRROR provide the same approximation guarantees with MIRROR and S-MIRROR, respectively. Otherwise, H-MIRROR and H-S-MIRROR do not provide non-trivial approximation guarantees. Notably, when G' is built via Theorem 1, H-MIRROR and H-S-MIRROR, as well as MIRROR and S-MIRROR, have a trivial approximation guarantee of $\frac{1}{m|V|}$ for maximizing $w_{T_S}(\Theta)$ (see Theorem 7 in Section S1 in the supplement [6]).

The updated G' in H-MIRROR and H-S-MIRROR may contain all vertices. As a result, the time complexities of H-MIRROR and H-S-MIRROR are the same with MIRROR and S-MIRROR, respectively.

5 EXPERIMENTS

In this section, we conduct experiments on a server with 64 ARM-architecture computing cores and 191 GB RAM[1].

5.1 Datasets

We use three real datasets as follows.

[1] **Our codes and datasets:** https://github.com/rucdatascience/temporal_bh

New York. We collect this dataset from the NYC OpenData website [1] and the New York City Taxi and Limousine Commission website [7]. We use it to build the New York graph, where vertices and edges represent road junctions and road segments, respectively. There are 77,580 vertices and 119,228 edges in total.

Each road junction is associated with nearby taxi requests in January 2015. We consider each natural hour as a time slot. For vertex v and time slot t_x, we query v in t_x if v is associated with a top $p\%$ largest number of taxi requests among all vertices in t_x, where p is a parameter. In this scenario, a temporal bump in the New York graph represents a pair of a geographical region and a time sub-interval such that large numbers of taxi requests are frequently detected in this region during this time sub-interval.

Reddit. We collect this dataset from the Reddit dump [2], which contains comments at the Reddit website [4]. These comments are written in various Reddit communities, *e.g.,* the financial community: r/wallstreetbets [5]. We build the Reddit graph, where each vertex represents one of three types of entities: (i) communities, (ii) keywords in comments, and (iii) pairs of communities and keywords (if there is a comment that is written in community a and contains keyword b, then there is a vertex representing the pair of a and b). For each vertex representing a pair of a community and a keyword, there are two edges that link this vertex to the corresponding community and keyword. There are 1,763,279 vertices and 2,046,668 edges in total.

Each vertex representing a pair of a community and a keyword is associated with corresponding comment activities (*e.g.,* the vertex representing the pair of community a and keyword b is associated with comments that are written in community a and contain keyword b) in September 2019. We consider each natural hour as a time slot. Like New York, for time slot t_x and vertex v that represents a pair of a community and a keyword, we query v in t_x if v is associated with a top $p\%$ largest number of comments in t_x. Then, a temporal bump in the Reddit graph corresponds to a time sub-interval and a sub-graph such that the vertices representing pairs of communities and keywords in this sub-graph are frequently queried during this time sub-interval, which means that large numbers of comments with the corresponding keywords are frequently written in the corresponding communities during this time sub-interval.

Wikipedia. We collect this dataset from the Wikipedia dump [9]. We use it to build the Wikipedia graph, where vertices represent Wikipedia pages. There is an edge between two vertices if the two corresponding pages are linked to each other, which indicates that these two pages are closely related. There are 1,176,192 vertices and 11,124,449 edges in total.

Each page is associated with page views in January 2020. We consider each natural hour during this period as a time slot. Like New York, for vertex v and time slot t_x, we query v in t_x if v is associated with a top $p\%$ largest number of page views in t_x. In this case, a temporal bump in the Wikipedia graph corresponds to a time sub-interval and a cluster of closely related pages such that these pages are intensively viewed during this time sub-interval.

5.2 Experiment settings

Baseline algorithms. We adapt four state-of-the-art static bump hunting algorithms [32, 62] to hunt temporal bumps as follows.

- BF-ST [32]: The main idea of BF-ST is to perform breadth first searches from queried vertices to obtain spanning trees of the graph, and then find the sub-tree that maximizes the static discrepancy as the hunted static bump. We apply BF-ST to hunt a temporal bump by first hunting a static bump, *i.e.,* a tree, in the above way, and then finding the pair of a sub-tree and a time sub-interval that maximizes the temporal discrepancy as the hunted temporal bump. We find multiple breadth first search trees to produce different solutions in the above way, and return the best solution. Specifically, we perform breadth first searches from randomly selected queried vertices s times to obtain s spanning trees of each maximum connected component of the graph that contains queried vertices, where s is a parameter.
- Random-ST [32]: The main idea of Random-ST is to find multiple random spanning trees of the graph, and then find the sub-tree that maximizes the static discrepancy as the hunted static bump. We apply Random-ST to hunt a temporal bump by first hunting a static bump in the above way, and then finding the pair of a sub-tree and a time sub-interval that maximizes the temporal discrepancy as the hunted temporal bump. Like BF-ST, we employ the parameter s, and find s random spanning trees of each maximum connected component of the graph.
- Smart-ST [32]: The main idea of Smart-ST is to find a minimum spanning tree of the graph for updated edge costs (the updated edge cost between two queried vertices is 0, between one queried and one not queried vertex is 1, and between two not queried vertices is 2), and then find the sub-tree that maximizes the static discrepancy as the hunted static bump. We apply Smart-ST to hunt a temporal bump by first hunting a static bump in the above way, and then finding the pair of a sub-tree and a time sub-interval that maximizes the temporal discrepancy as the hunted temporal bump.
- PCST [32, 62]: The main idea of PCST in [32] is to hunt a static bump by solving the static prize-collecting Steiner tree problem. We apply it to hunt a temporal bump by first hunting a static bump in the above way, and then finding the pair of a sub-tree and a time sub-interval that maximizes the temporal discrepancy as the hunted temporal bump. The algorithm in [62] hunts k static bumps by solving a Steiner forest problem [40] that becomes the static prize-collecting Steiner tree problem when $k = 1$. Hence, we also consider PCST as the adaptation of the algorithm in [62] for hunting temporal bumps.

Parameters. We vary three parameters as follows.

- m: the number of time slots. We randomly extract m continuous hours in the dataset as the time interval T.
- p: the percentage of queried vertices (details in Section 5.1).
- s: the parameter in BF-ST and Random-ST.

We set the default values of parameters as: for New York, $m = 72$, $p = 1, s = 10$; for Reddit, $m = 40$, $p = 1, s = 4$; for Wikipedia, $m = 40$, $p = 1, s = 7$. We vary these parameters in Figure 3. When we vary one parameter, we set the other parameters to default values.

Notably, H-MIRROR and H-S-MIRROR employ a value b that is defined as the minimum possible number of vertices between two queried vertices. For New York and Wikipedia, since two queried vertices may be adjacent, $b = 0$. For Reddit, since vertices representing pairs of communities and keywords are queried, there is at

Figure 2: Comparison of solution quality and speed.

least one vertex between two queried vertices, and $b = 1$. In Section S6 in the supplement [6], we vary b, and show that defining b in the above way gives H-MIRROR and H-S-MIRROR a high performance. As a result, for the easy use of these two algorithms, we define b in the above way, and do not treat b as a parameter in this paper.

Metrics. We evaluate two metrics as follows.

- $D_C^{T_S}$: the temporal discrepancy (see Equation (3)), which equals $w_{T_S}(\Theta)$ (see Theorem 1). A larger value of $D_C^{T_S}$ is better.
- t: the running time of algorithms (unit: second).

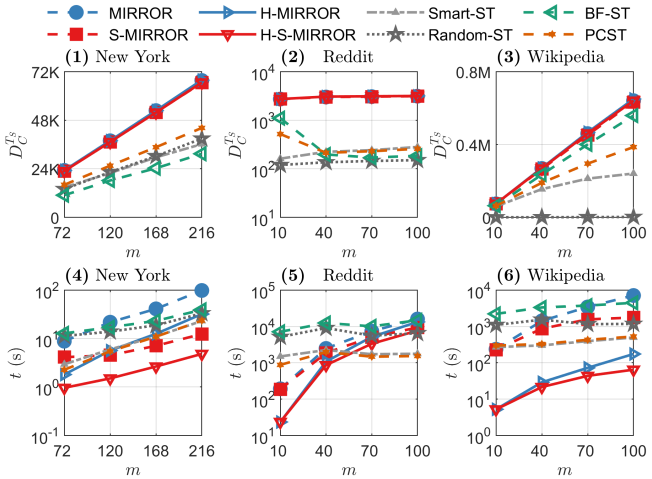
5.3 Quantitative experiment results

Here, for each set of parameters, we randomly generate 50 instances, and then visualize and compare the average metric values.

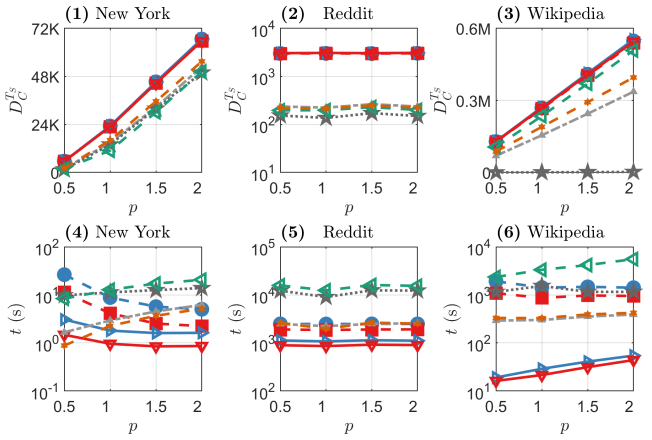
Comparison of solution quality and speed. We compare the solution quality and speed of algorithms in Figure 2. First, we observe that, in Figures 2 (1-3), $D_C^{T_S}$ values of the proposed algorithms are considerably larger than those of the baseline algorithms. Particularly, in Figure 2 (2), $D_C^{T_S}$ values of the proposed algorithms are an order of magnitude larger than those of the baseline algorithms. This shows that the baseline algorithms are not as effective as the proposed ones for hunting high-discrepancy temporal bumps, as discussed in Section 1. Nevertheless, in Figure 2 (3), the $D_C^{T_S}$ value of BF-ST almost matches those of the proposed algorithms, and is higher than those of the other baseline algorithms. This indicates that, when BF-ST performs breadth first searches from queried vertices to obtain spanning trees of the Wikipedia graph, queried vertices are often close to each other in these trees, which makes it possible to hunt high-discrepancy bumps from these trees.

For speed, we observe that, in Figures 2 (4-6), MIRROR and S-MIRROR are often slower than the baseline algorithms (especially, slower than Smart-ST and PCST). We consider this as the price of achieving non-trivial approximation guarantees by these two algorithms. In comparison, H-MIRROR and H-S-MIRROR are generally faster than the baseline algorithms. In particular, in Figure 2 (6), H-MIRROR and H-S-MIRROR are an order of magnitude faster than MIRROR and S-MIRROR, and also significantly faster than the baseline algorithms. Thus, H-MIRROR and H-S-MIRROR have a high efficiency for hunting temporal bumps.

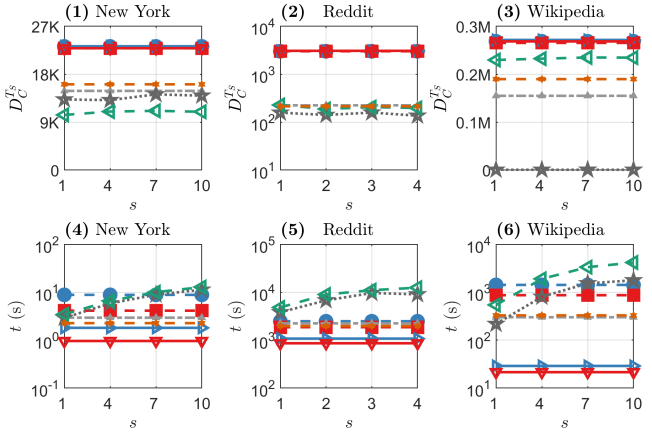
Moreover, in Figures 2 (1-3), we observe that the proposed algorithms hunt bumps with similar discrepancies with each other, even though MIRROR and S-MIRROR provide different approximation



(a) Experiment results of varying m



(b) Experiment results of varying p



(c) Experiment results of varying s

Figure 3: Experiment results of varying m, p, and s.

guarantees, and H-MIRROR and H-S-MIRROR do not provide any non-trivial approximation guarantee. This shows the usefulness of H-MIRROR and H-S-MIRROR in practice, considering the high efficiency of these two algorithms as discussed above.

Variation of the length of the input time interval: m. We vary m in Figure 3a. In Figures 3a (1) and (3), $D_C^{T_S}$ increases with m for

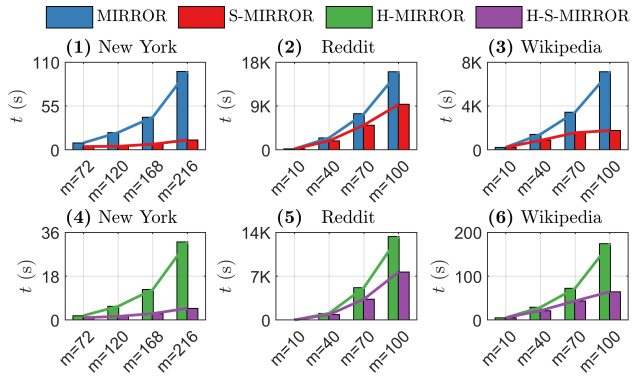


Figure 4: Nearly quadratic and linear scalabilities to m.

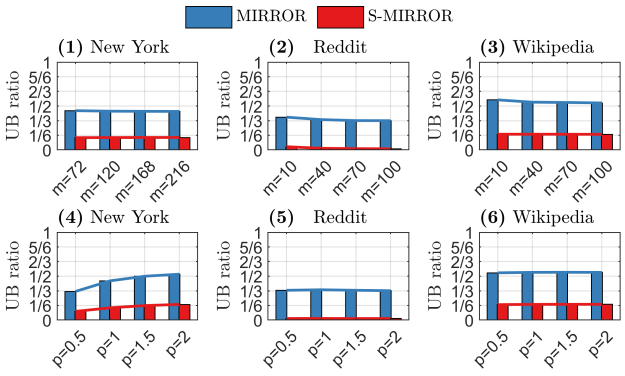


Figure 5: Upper bound ratios of MIRROR and S-MIRROR.

New York and Wikipedia. The reason is that bumps with higher discrepancies often exist in larger time intervals. However, in Figure 3a (2), D_C^{Ts} values of the proposed algorithms do not change much with m for Reddit. The reason is that queried vertices in the Reddit graph in different time slots are often different and far away from each other, which means that bumps with higher discrepancies may not exist in larger time intervals. Notably, the baseline algorithms may not be able to hunt high-discrepancy temporal bumps as m varies. For example, in Figure 3a (2), D_C^{Ts} values of BF-ST and PCST decrease with m for Reddit, and in Figure 3a (3), D_C^{Ts} values of Random-ST are always negligible when comparing to the other algorithms. On the other hand, the superior solution qualities of the proposed algorithms hold well as m varies.

In Figures 3a (4-6), S-MIRROR and H-S-MIRROR scale better to m than MIRROR and H-MIRROR, respectively, *i.e.,* t values of S-MIRROR and H-S-MIRROR increase with m at a lower rate than MIRROR and H-MIRROR. The reason is that S-MIRROR and H-S-MIRROR compute a nearly linear number of time sub-intervals with respect to m, while MIRROR and H-MIRROR compute a quadratic number of time sub-intervals with respect to m. In Figures 3a (4-6), we show t values using logarithmic scales, for visualizing t values of different algorithms clearly. In Figure 4, we present t values of the proposed algorithms using non-logarithmic scales, for clearly showing that MIRROR and H-MIRROR scale nearly quadratically, and S-MIRROR and H-S-MIRROR scale nearly linearly, to m.

In Figures 3a (4-6), the baseline algorithms often scale well to m. The reason is as follows. Each baseline algorithm first hunts a static bump, *i.e.,* a tree, and then finds the pair of a sub-tree and a time sub-interval that maximizes the temporal discrepancy as a temporal bump. Since the hunted static bump is often small, it is often fast to hunt the above temporal bump. As a result, the baseline algorithms may be faster than the proposed ones when m is large, *e.g.,* in Figure 3a (5), Smart-ST and PC-ST are faster than the proposed algorithms when $m = 100$. Nonetheless, as discussed above, since the baseline algorithms often cannot hunt high-discrepancy temporal bumps in large time intervals, it may not be recommended to use the baseline algorithms when m is large, *e.g.,* in Figure 3a (2), D_C^{Ts} values of Smart-ST and PC-ST are more than an order of magnitude smaller than those of the proposed algorithms when $m = 100$, which indicates that it is not recommended to use these two algorithms when $m = 100$, even after considering their high speed here.

Notably, as shown in Figures 3a (4-6), MIRROR has a low efficiency when m is large, *e.g.,* MIRROR consumes nearly 10^4 seconds to produce a solution when $m = 100$ in Figure 3a (6). This low efficiency shows the need of worrying about the scalability of hunting bumps in temporal cases, and justifies the value of the other faster proposed algorithms, especially H-S-MIRROR, which is the fastest proposed one. An example is as follows. In Figures 3a (3) and (6), BF-ST hunts bumps with similar discrepancies with MIRROR, and are slightly faster than MIRROR, when $m = 100$, *i.e.,* BF-ST is as good as MIRROR here. In comparison, the other faster proposed algorithms still performs better than BF-ST here, *e.g.,* in Figures 3a (3) and (6), H-S-MIRROR hunts bumps with slightly higher discrepancies than BF-ST, and is much faster than BF-ST, when $m = 100$. Note that, as shown in Figures 3a (4-6), for New York and Wikipedia, H-S-MIRROR is more than an order of magnitude faster than MIRROR, while for Reddit, H-S-MIRROR is not so significantly faster than MIRROR when m is large (see a clear comparison in Figures 4 (2) and (5)). The major reason is that, different from New York and Wikipedia, a large percentage of vertices have been queried at least once during the input time interval when m is large for Reddit, and as a result, the acceleration strategy of only computing queried vertices and their close neighbors does not work well for Reddit. Nevertheless, since this strategy works well for New York and Wikipedia, we consider this strategy as useful in practice.

Variation of the percentage of queried vertices: p. We vary p in Figure 3b. We observe that, in Figures 3b (1) and (3), D_C^{Ts} increases with p for New York and Wikipedia. The reason is that vertices that are associated with large amounts of activities in the New York and Wikipedia graphs are often close to each other. As p increases, more vertices are queried and included in solutions. For this reason, in Figures 3b (4) and (6), t values of MIRROR and S-MIRROR decrease with p, since these two algorithms use higher-discrepancy solutions to prune more time sub-intervals in the branch and bound process, *i.e.,* Lines 8-16 in MIRROR. In comparison, in Figure 3b (6), t values of H-MIRROR and H-S-MIRROR increase with p, since these two algorithms compute more breadth first searched vertices as p increases. In Figure 3b (2), D_C^{Ts} does not change much with p for Reddit. The reason is that vertices that are associated with large amounts of activities in the Reddit graph are often not close to each other. As a result, solutions that include more queried vertices may not be found as p increases. For this reason, in Figure 3b (5), t values of the proposed algorithms do not change much with p.

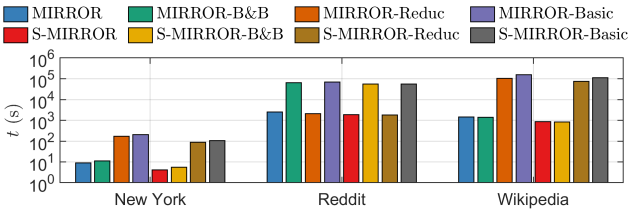
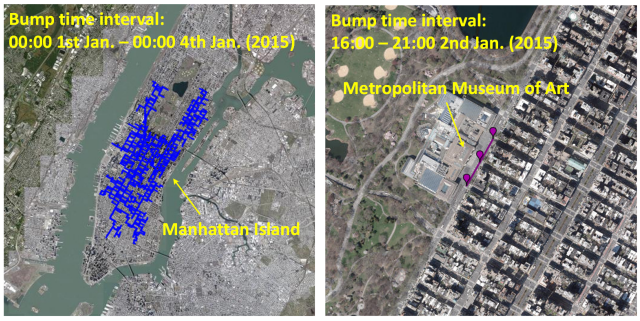


Figure 6: The effectiveness of acceleration techniques.

Variation of the parameter s in BF-ST **and** Random-ST**.** We vary s in Figure 3c. Each of BF-ST and Random-ST computes s feasible solutions, and returns the one with the highest discrepancy (details in Section 5.2). As a result, t values of these two algorithms increase with s in Figures 3c (4-6). Nevertheless, in Figures 3c (1-3), $D_C^{T_S}$ values of these two algorithms do not increase much with s. In particular, by setting s to such large values that these two algorithms are slower than MIRROR and S-MIRROR, the solutions of these two algorithms are still worse than those of MIRROR and S-MIRROR. Thus, these two algorithms are not as effective as the proposed algorithms for hunting temporal bumps.

Upper bound ratios of MIRROR **and** S-MIRROR**.** MIRROR and S-MIRROR have theoretical guarantees on solution qualities, and can compute upper bounds of the maximum solution weight: $w_{T_S}(\Theta)$, which equals $D_C^{T_S}$. We refer to the ratio of the solution weight of MIRROR to the upper bound computed by MIRROR as the upper bound ratio of MIRROR for maximizing the solution weight (similarly, the upper bound ratio of S-MIRROR). Recall that, it is NP-hard to approximately maximize the solution weight within any constant ratio. We illustrate the upper bound ratios of MIRROR and S-MIRROR for varying m and p in Figure 5 (notably, since these two algorithms do not contain the parameter s, we do not vary s here). Note that, the upper bound ratios of MIRROR are often above $\frac{1}{3}$, which means that the solution weight of MIRROR is often at least a third of the maximum solution weight. In particular, for New York and Wikipedia, the upper bound ratios of MIRROR are often around $\frac{1}{2}$. In comparison, the upper bound ratios of S-MIRROR are often around $\frac{1}{6}$ for New York and Wikipedia, and are negligible for Reddit. The reason why S-MIRROR has smaller upper bound ratios than MIRROR is that S-MIRROR has looser approximation guarantees than MIRROR. We consider this as the price of achieving a nearly linear scalability with respect to m. The previous experiment results show that S-MIRROR produces similar solutions with MIRROR in practice. This does not mean that the approximation guarantees of S-MIRROR could be improved to those of MIRROR, since these guarantees are for theoretically worst cases, not practical cases.

Acceleration techniques in MIRROR **and** S-MIRROR**.** We use reduction and branch and bound techniques to accelerate MIRROR and S-MIRROR. Specifically, the Degree-0-1-2 test in Line 3 of MIRROR is a reduction technique adapted from the existing work on classical Steiner tree problems (*e.g.,* [25, 26, 54]), while Theorem 2 and Lines 8-16 of MIRROR are newly developed techniques based on the classic branch and bound idea [44]. We evaluate the effectiveness of these techniques in Figure 6, where MIRROR-B&B is MIRROR with branch and bound, but not reduction; MIRROR-Reduc is MIRROR with reduction, but not branch and bound; and MIRROR-Basic is MIRROR with neither reduction nor branch and bound (similar are S-MIRROR-B&B *etc.*). We observe that MIRROR and S-MIRROR are at least an order of magnitude faster than MIRROR-Basic and S-MIRROR-Basic, respectively. This shows the effectiveness of the above techniques for accelerating MIRROR and S-MIRROR. For New York and Wikipedia, MIRROR-B&B and S-MIRROR-B&B are much faster than MIRROR-Reduc and S-MIRROR-Reduc, respectively. This indicates that the branch and bound technique is more effective than the reduction technique for New York and Wikipedia. In comparison, for Reddit, MIRROR-B&B and S-MIRROR-B&B are much slower than MIRROR-Reduc and S-MIRROR-Reduc, respectively. This means that the reduction technique is more effective than the branch and bound technique for Reddit. The reason is that the Degree-2 test in Line 3 of MIRROR removes a large number of not queried vertices that represent pairs of communities and keywords in the Reddit graph.



(a) Top 1% **intensive taxi requests (b)** 500% **increased taxi requests**

Figure 7: Case studies using the New York **dataset.**

5.4 Case studies

Here, we conduct New York, Reddit and Wikipedia case studies.

New York case studies. We load the New York graph in Section 5.1, where each vertex, *i.e.,* road junction, is associated with nearby taxi requests in January 2015. We consider each natural hour as a time slot. Given the time interval T from 00:00 1st to 00:00 4th January 2015, for each pair of vertex v and time slot $t_x \in T$, we query v in t_x if v is associated with a top 1% largest number of taxi requests among all vertices in t_x. We use H-MIRROR to hunt a temporal bump, and visualize this bump in Figure 7a. This bump is located at the central region of the Manhattan island, and spans T, which means that taxis are intensively requested in Manhattan during T. This bump verifies the fact that Manhattan is the urban core of the New York metropolitan area. Notably, this bump spans the whole input time interval T. This shows that queried vertices, *i.e.,* road junctions with intensive taxi requests, often do not change from time slots to time slots. This stability does not happen when we query vertices differently as follows.

We query vertices associated with abnormally large numbers of taxi requests as follows. Given T from 01:00 2nd to 01:00 3rd January 2015, for each pair of vertex v and time slot $t_x \in T$, we query v in t_x if the number of taxi requests associated with v in t_x has increased more than 500% from the corresponding number in the last day. We use H-MIRROR to hunt a temporal bump, and visualize it in Figure 7b. This bump is located at the front of the Metropolitan Museum of Art, and lasts from 16:00 to 21:00 2nd January 2015. Intuitively,

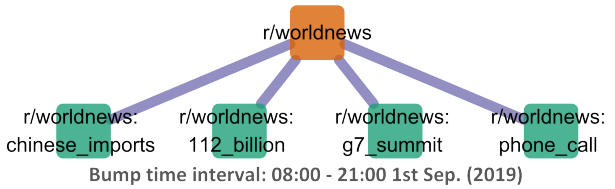


Figure 8: A case study using the Reddit **dataset.**

abnormally large numbers of taxi requests are induced by events, *e.g.,* special exhibitions. This bump indicates that an event may be hold in the above museum during the above period. The connection of sub-graph in the temporal bump hunting problem helps discover this knowledge. We explain this as follows. Let the three locations in the bump in Figure 7b be $L1$, $L2$ and $L3$, respectively. Assume that there are two other locations $L4$ and $L5$ that are also queried during the time of this bump. If we just find which location has a peak of the number of taxi requests, we could return the list of "$L1$, $L4$, $L2$, $L5$ and $L3$" without knowing that $L1$, $L2$ and $L3$ may collectively correspond to an event. Thus, connecting $L1$, $L2$ and $L3$ together helps discover the above event. The discrepancy maximization objective also helps discover the location and time of this event, as this objective ensures that peaked numbers of taxi requests are frequently detected in the above sub-graph during the above time. After discovering this event, taxi companies could contact the museum to ask if there will be more events during the same time in the next few days. If there are more events, then they could allocate taxis to prepare for these events.

A Reddit case study. Initially, we load the Reddit graph in Section 5.1, where each vertex representing a pair of a community and a keyword is associated with corresponding comment activities in September 2019. We consider each natural hour as a time slot. Given the time interval T from 00:00 1st to 00:00 2nd September 2019, for each pair of time slot $t_x \in T$ and vertex v representing a pair of a community and a keyword, we query v in t_x if v is associated with a top 0.006% largest number of comment activities. The reason why we use a small percentage here is that a large percentage induces a bump that is too large to be visualized. We use H-MIRROR to hunt a temporal bump, and visualize this bump in Figure 8, where each green vertex represents a pair of a community and a keyword, *e.g.,* "r/worldnews: chinese_imports" represents the pair of community "r/worldnews" and keyword "chinese_imports". The single orange vertex represents the community "r/worldnews". This bumps exists from 08:00 to 21:00 1st September 2019, and shows the great interest of people in commenting on some topics on world news during this time, particularly the topic that US President Donald Trump's 15% tariffs on $112 billion in Chinese goods took effect on 1st September 2019, just after the G7 summit [8]. Knowing this could help medias make attractive contents, such as making a world-news-focused TV show on how G7 reacts to the $112 billion tariffs, and allowing common viewers to send interactive comments during the show.

A Wikipedia case study. First, we load the Wikipedia graph in Section 5.1, where each vertex, *i.e.,* Wikipedia page, is associated with page views in January 2020. We consider each natural hour as a time slot. Given the time interval T from 00:00 3rd to 00:00 4th January 2020, for each pair of vertex v and time slot $t_x \in T$, we query v in t_x if v is associated with a top 0.002% largest number of page views among all vertices in t_x. We use H-MIRROR to hunt a temporal bump, and visualize it in Figure 1. This bump spans the time sub-interval between 04:00 3rd and 00:00 4th January 2020, contains 6 Wikipedia pages, and has a discrepancy of 88. As discussed in Section 1, this bump shows that, except the topic of "Qasem Soleimani" that directly corresponds to the US-Iran conflict, people are beginning to pay attention to the related topic of "Iran Iraq War" shortly after this conflict, and mining this knowledge could help medias make decisions on producing TV shows on Iran Iraq War after talking about the US-Iran conflict. Except H-MIRROR, we also apply PCST to hunt a bump under the same settings. Different from the bump hunted by H-MIRROR, the bump hunted by PCST only contains three Wikipedia pages: "Qasem_Soleimani", "Ali_Khamenei" and "Ruhollah_Khomeini" (*i.e.,* the three above pages in Figure 1), and has a smaller discrepancy of 52. Thus, PCST cannot discover the knowledge that people are beginning to pay attention to "Iran Iraq War". This shows that, in comparison with the adaptation of the existing static bump hunting techniques, the proposed techniques could mine additional useful knowledge by hunting temporal bumps with higher discrepancies. Moreover, as discussed in Section 1, the existing work on event detection cannot play the same role with the proposed techniques, *e.g.,* in Section S7 in the supplement [6], we show that an existing event detection technique [56] cannot detect a cluster of closely related Wikipedia pages and a time sub-interval that correspond to our intensive attention during a certain period of time, and thus does not rule out the particular usefulness of the proposed techniques in this case.

5.5 Key observations in experiments

To help analyze the above experiment results, we summarize the key observations as follows.

- The proposed algorithms hunt bumps with significantly (sometimes an order of magnitude) higher discrepancies than the baseline algorithms (*e.g.,* Figure 2 (2)). Meanwhile, MIRROR and S-MIRROR are often slower than, while H-MIRROR and H-S-MIRROR are generally faster than, the baseline algorithms (*e.g.,* Figures 2 (4-6)).
- MIRROR and H-MIRROR scale nearly quadratically, while S-MIRROR and H-S-MIRROR scale nearly linearly, to the length of the input time interval: m (see Figure 4).
- Although MIRROR and S-MIRROR generally produce similar solutions in practice (*e.g.,* Figures 2 (1-3)), MIRROR achieves tighter guarantees than S-MIRROR (see Figure 5). Specifically, the solution weight of MIRROR is often projected to be at least one third of the maximum solution weight (see Figure 5).
- H-MIRROR and H-S-MIRROR generally produce similar solutions with MIRROR and S-MIRROR (*e.g.,* Figures 2 (1-3)), while being considerably faster than MIRROR and S-MIRROR in various cases (*e.g.,* Figure 2 (6)).
- Hunting temporal bumps can retrieve information from different types of graphs with dynamic vertex properties, and helps throw light upon the dynamics of a city (*e.g.,* Figure 7), as well as analyze our attention on the Internet (*e.g.,* Figures 1 and 8).

6 RELATED WORK

Bump hunting. Originated as the activity of detecting real bumps in mass spectra in the field of high energy physics (*e.g.,* [50, 58, 63]),

bump hunting has been continuously studied and become an increasingly important data analysis approach (*e.g.,* [10, 30, 32, 34, 35, 38, 62]). Traditional bump hunting techniques apply geometric knowledge to find regions of Euclidean datasets where a property of interest occurs frequently (*e.g.,* [10, 30, 34, 35, 38]). Recently, Gionis *et al.* [32] advance these techniques by applying graph theory knowledge to find such regions of graph datasets. They develop several heuristic algorithms to hunt a static bump, *i.e.*, a connected sub-graph, with the maximum discrepancy between the numbers of queried and not queried vertices. The more recent work in [62] employs this discrepancy maximization idea, and finds k non-overlapping static bumps such that the sum of discrepancies of these bumps is maximized. They prove that this multiple bump hunting problem is NP-hard even when the graph is a set of non-overlapping trees. The above work is different from community search with queried vertices (*e.g.,* [14, 15, 27, 31, 43, 45, 60]). Specifically, bump hunting finds a sub-graph that contains as many queried vertices as possible, and as few not queried vertices as possible. In comparison, community search finds a sub-graph with particular structural properties (*e.g.,* a high density) to connect queried vertices, and does not minimize the number of not queried vertices in the sub-graph. Thus, the above work on bump hunting is useful in finding regions of graphs where the property of interest occurs frequently. However, the above work assumes that vertices exhibit the property of interest statically. As discussed in Section 1, due to the neglect of dynamic vertex properties, the above work [32] cannot hunt temporal bumps directly, and the adaptation of the above work cannot achieve non-trivial guarantees of solution qualities for hunting temporal bumps, or hunt high-discrepancy temporal bumps in practice. Moreover, as discussed in Section 1, the existing work on temporal graphs or dynamic networks performs different tasks from bump hunting, and does not address the above issue. The proposed approach in this paper is different from the above previous work in that (i) it computes dynamic vertex properties in such a way that non-trivial approximation guarantees of hunting temporal bumps are achieved; and (ii) it uses newly developed techniques to accelerate the computing process to a practically satisfiable degree.

Prize-collecting Steiner trees. The static prize-collecting Steiner tree problem was first studied by Segev [59], while the term "prize-collecting Steiner tree" was first used by Bienstock *et al.* [16], who develop the first approximation algorithm for solving this problem, which is NP-hard. Their algorithm has an approximation guarantee of 3 for minimizing the solution cost. Goemans and Williamson [33] develop another algorithm (which we refer to as the GW algorithm) using the linear programming relaxation model of Bienstock *et al.*, and achieve an improved guarantee of $2 - 1/(|V| - 1)$. Recently, Archer *et al.* [13] achieve a guarantee below 1.9672 (for minimizing the solution cost) by combining the GW algorithm with a Mixed Integer Programming (MIP) algorithm [18] for solving the classical Steiner tree problem in graphs [24]. To our knowledge, this is the tightest approximation guarantee for the static prize-collecting Steiner tree problem to date. However, it is too slow to achieve this guarantee, since doing this requires running the GW algorithm twice and the MIP algorithm once for every possible root of the optimal solution tree. Thus, most work about approximating prize-collecting Steiner trees focuses on accelerating the GW algorithm (*e.g.,* [22, 39, 42, 61]). There are two phases in the GW algorithm: growing and pruning. The fastest implementation of the growing phase has a time complexity of $O(d|E| \log |V|)$ [39], while the fastest implementation of the pruning phase has a time complexity of $O(|V|)$ in both rooted and unrooted scenarios [61]. We incorporate these fast implementations into the process of solving the temporal prize-collecting Steiner tree problem (see Lines 17-18 in MIRROR).

7 CONCLUSIONS AND FUTURE WORK

Given a time interval and a graph where vertices exhibit a property of interest dynamically, an interesting question is: where and when does the property of interest occur frequently? We answer this question by solving the temporal bump hunting problem. Initially, we propose two approximation algorithms, dubbed MIRROR and S-MIRROR respectively. MIRROR achieves a tight approximation guarantee, at the cost of a weak scalability to the number of time slots. In comparison, S-MIRROR achieves a strong scalability to the number of time slots, at the price of a loose approximation guarantee. We further propose two heuristic algorithms, dubbed H-MIRROR and H-S-MIRROR, respectively. These two algorithms do not provide non-trivial approximation guarantees, but produce similar solutions with, and are much faster than the two approximation algorithms. Experiments on real datasets show that our algorithms considerably outperform the state of the art for hunting temporal bumps in graphs with dynamic vertex properties.

Some future work can be done based on the work in this paper. Recall that S-MIRROR samples and computes time sub-intervals in such a way that both non-trivial approximation guarantees and a nearly linear scalability to m are achieved. It is recommended to explore new sampling methods in the future, so that tighter guarantees or stronger scalabilities than S-MIRROR may be achieved.

Moreover, we can modify the proposed algorithms to hunt temporal bumps under some different problem settings. First, for the problem of finding a component C_{max} for a given time sub-interval T_S such that the pair of $\{C_{max}, T_S\}$ has the maximum discrepancy, we can modify the proposed algorithms to solve this problem by only computing T_S in Line 5 of MIRROR (in comparison, the problem of finding a time sub-interval T_{max} for a given component C such that $\{C, T_{max}\}$ has the maximum discrepancy can be solved to optimality in polynomial time by enumerating every time sub-interval T_i and computing the discrepancy of $\{C, T_i\}$). Second, consider two temporal bumps $\{C_1, T_1\}$ and $\{C_2, T_2\}$ as non-overlapping if $C_1 \cap C_2 = \emptyset$ or $T_1 \cap T_2 = \emptyset$, or both, then, we can modify the proposed algorithms to heuristically hunt top-k non-overlapping temporal bumps in the following iterative way: when enumerating a time sub-interval T_i in Line 5 of MIRROR for hunting a bump that spans T_i, only compute vertices such that a bump that spans T_i and contains these vertices do not overlap with previously hunted bumps. Nevertheless, applying such modifications to hunt multiple bumps is slow. Some future work can be done to address this issue.

ACKNOWLEDGMENTS

This work is funded by (i) NSFC 61925203; (ii) PKU-Baidu Fund 2019BD006; and (iii) two start up grants from Renmin University of China and National University of Singapore, respectively.

REFERENCES

[1] 2021. NYC OpenData. https://opendata.cityofnewyork.us.
[2] 2021. pushshift.io. https://pushshift.io.
[3] 2021. Qasem Soleimani: US kills top Iranian general in Baghdad air strike. https://www.bbc.com/news/world-middle-east-50979463.
[4] 2021. reddit: the front page of the internet. https://www.reddit.com.
[5] 2021. r/wallstreetbets. https://www.reddit.com/r/wallstreetbets.
[6] 2021. Supplement. https://github.com/rucdatascience/temporal_bh/blob/main/Supplement.pdf.
[7] 2021. The New York City Taxi and Limousine Commission. https://www1.nyc.gov/site/tlc/about/about-tlc.page.
[8] 2021. Trump's 15% tariffs on $112 billion in Chinese goods take effect. https://www.cnbc.com/2019/09/01/trumps-15percent-tariffs-on-112-billion-in-chinese-goods-take-effect.html.
[9] 2021. Wikimedia Dump. https://dumps.wikimedia.org.
[10] Deepak Agarwal, Jeff M Phillips, and Suresh Venkatasubramanian. 2006. The hunting of the bump: on maximizing statistical discrepancy. In *Proceedings of the seventeenth annual ACM-SIAM Symposium on Discrete Algorithm*. Society for Industrial and Applied Mathematics, 1137–1146.
[11] Charu C Aggarwal and Karthik Subbian. 2012. Event detection in social streams. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 624–635.
[12] Xiang Ao, Haoran Shi, Jin Wang, Luo Zuo, Hongwei Li, and Qing He. 2019. Large-scale frequent episode mining from complex event sequences with hierarchies. *ACM Transactions on Intelligent Systems and Technology* 10, 4 (2019), 1–26.
[13] Aaron Archer, Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi, and Howard Karloff. 2011. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing* 40, 2 (2011), 309–332.
[14] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data mining and knowledge discovery* 29, 5 (2015), 1406–1433.
[15] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *Proceedings of the VLDB Endowment* 11, 9 (2018).
[16] Daniel Bienstock, Michel X Goemans, David Simchi-Levi, and David Williamson. 1993. A note on the prize collecting traveling salesman problem. *Mathematical programming* 59, 1-3 (1993), 413–420.
[17] Petko Bogdanov, Misael Mongiovì, and Ambuj K Singh. 2011. Mining heavy subgraphs in time-evolving networks. In *International Conference on Data Mining*. IEEE, 81–90.
[18] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2010. An improved LP-based approximation for Steiner tree. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 583–592.
[19] Jose Cadena and Anil Vullikanti. 2018. Mining heavy temporal subgraphs: Fast algorithms and applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[20] Deepayan Chakrabarti and Christos Faloutsos. 2006. Graph mining: Laws, generators, and algorithms. *ACM computing surveys* 38, 1 (2006).
[21] Lingyang Chu, Yanyan Zhang, Yu Yang, Lanjun Wang, and Jian Pei. 2019. Online density bursting subgraph detection from temporal graphs. *Proceedings of the VLDB Endowment* 12, 13 (2019), 2353–2365.
[22] Richard Cole, Ramesh Hariharan, Moshe Lewenstein, and Ely Porat. 2001. A faster implementation of the Goemans-Williamson clustering algorithm. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms*. 17–25.
[23] Mário Cordeiro, Rui Portocarrero Sarmento, and Joao Gama. 2016. Dynamic community detection in evolving networks using locality modularity optimization. *Social Network Analysis and Mining* 6, 1 (2016), 15.
[24] Stuart E Dreyfus and Robert A Wagner. 1971. The Steiner problem in graphs. *Networks* 1, 3 (1971), 195–207.
[25] Cees Duin. 2000. Preprocessing the Steiner problem in graphs. In *Advances in Steiner Trees*. Springer, 175–233.
[26] Cees W Duin and Anton Volgenant. 1989. Reduction tests for the Steiner problem in graphs. *Networks* 19, 5 (1989), 549–567.
[27] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *The VLDB Journal* 26, 6 (2017), 803–828.
[28] Mateusz Fedoryszak, Brent Frederick, Vijay Rajaram, and Changtao Zhong. 2019. Real-time event detection on social data streams. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining*. 2774–2782.
[29] Joan Feigenbaum, Christos H Papadimitriou, and Scott Shenker. 2001. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences* 63, 1 (2001), 21–41.
[30] Jerome H Friedman and Nicholas I Fisher. 1999. Bump hunting in high-dimensional data. *Statistics and Computing* 9, 2 (1999), 123–143.
[31] Edoardo Galimberti, Martino Ciaperoni, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2020. Span-core Decomposition for Temporal Networks: Algorithms and Applications. *ACM Transactions on Knowledge Discovery from Data* 15, 1 (2020), 1–44.
[32] Aristides Gionis, Michael Mathioudakis, and Antti Ukkonen. 2017. Bump hunting in the dark: Local discrepancy maximization on graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 3 (2017), 529–542.
[33] Michel X Goemans and David P Williamson. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24, 2 (1995), 296–317.
[34] IJ Good and ML Deaton. 1981. Recent advances in bump hunting. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*. Springer, 92–104.
[35] IJ Good and RA Gaskins. 1980. Density estimation and bump-hunting by the penalized likelihood method exemplified by scattering and meteorite data. *Journal of the American Statistical Association* 75, 369 (1980), 42–56.
[36] Valery Guralnik and Jaideep Srivastava. 1999. Event detection from time series data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 33–42.
[37] Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. 2015. Commit: A scalable approach to mining communication motifs from dynamic networks. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 475–489.
[38] Nancy E Heckman. 1992. Bump hunting in regression analysis. *Statistics & probability letters* 14, 2 (1992), 141–152.
[39] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2014. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem. In *Workshop of the 11th DIMACS Implementation Challenge*.
[40] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. 2015. A nearly-linear time framework for graph-structured sparsity. In *International Conference on Machine Learning*. 928–937.
[41] Silu Huang, Ada Wai-Chee Fu, and Ruifeng Liu. 2015. Minimum spanning trees in temporal graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 419–430.
[42] David S Johnson, Maria Minkoff, and Steven Phillips. 2000. The prize collecting Steiner tree problem: theory and practice. In *Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*. 760–769.
[43] Isabel M Kloumann and Jon M Kleinberg. 2014. Community membership identification from small seed sets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1366–1375.
[44] Eugene L Lawler and David E Wood. 1966. Branch-and-bound methods: A survey. *Operations research* 14, 4 (1966), 699–719.
[45] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential community search in large networks. *Proceedings of the VLDB Endowment* 8, 5 (2015), 509–520.
[46] Paul Liu, Austin R Benson, and Moses Charikar. 2019. Sampling methods for counting temporal motifs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 294–302.
[47] Yu Liu, Baojian Zhou, Feng Chen, and David W Cheung. 2016. Graph topic scan statistic for spatial event detection. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 489–498.
[48] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jin-Peng Huai. 2020. An efficient approach to finding dense temporal subgraphs. *IEEE Transactions on Knowledge and Data Engineering* 32, 4 (2020), 645–658.
[49] Misael Mongiovi, Petko Bogdanov, Razvan Ranca, Evangelos E Papalexakis, Christos Faloutsos, and Ambuj K Singh. 2013. Netspot: Spotting significant anomalous regions on dynamic networks. In *Proceedings of the 2013 SIAM international conference on data mining*. SIAM, 28–36.
[50] J Orear and D Cassel. 1971. Applications of statistical inference to physics. *Foundations of Statistical inference* (1971), 280–288.
[51] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. 2007. Quantifying social group evolution. *Nature* 446, 7136 (2007), 664–667.
[52] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 601–610.
[53] Adam Perer and Fei Wang. 2014. Frequence: Interactive mining and visualization of temporal frequent event sequences. In *Proceedings of the 19th international conference on Intelligent User Interfaces*. 153–162.
[54] Daniel Rehfeldt, Thorsten Koch, and Stephen J Maher. 2019. Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem. *Networks* 73, 2 (2019), 206–233.
[55] Giulio Rossetti and Rémy Cazabet. 2018. Community discovery in dynamic networks: a survey. *Comput. Surveys* 51, 2 (2018), 1–37.
[56] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. 2014. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1176–1185.
[57] Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. 2018. Finding events in temporal networks: Segmentation meets densest-subgraph discovery. In *2018 IEEE International Conference on Data Mining*.
[58] NP Samios. 1972. Current problems in experimental boson spectroscopy. In *AIP Conference Proceedings*, Vol. 8. AIP, 432–459.

[59] Arie Segev. 1987. The node-weighted Steiner tree problem. *Networks* 17, 1 (1987), 1–17.
[60] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining*. 939–948.
[61] Yahui Sun, Marcus Brazil, Doreen Thomas, and Saman Halgamuge. 2019. The fast heuristic algorithms and post-processing techniques to design large and low-cost communication networks. *IEEE/ACM Transactions on Networking* 27, 1 (2019), 375–388.
[62] Yahui Sun, Jun Luo, Theodoros Lappas, Xiaokui Xiao, and Bin Cui. 2020. Hunting multiple bumps in graphs. *Proceedings of the VLDB Endowment* 13, 5 (2020), 656–669.
[63] George L Trigg. 1970. Rules for "Bump Hunting". *Physical Review Letters* 25, 12 (1970), 783.
[64] Jingjing Wang, Yanhao Wang, Wenjun Jiang, Yuchen Li, and Kian-Lee Tan. 2020. Efficient sampling algorithms for approximate temporal motif counting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1505–1514.
[65] Yong Wang, Guoliang Li, and Nan Tang. 2019. Querying shortest paths on time dependent road networks. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1249–1261.
[66] Dong Wen, Yilun Huang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2020. Efficiently Answering Span-Reachability Queries in Large Temporal Graphs. In *2020 IEEE 36th International Conference on Data Engineering*. IEEE, 1153–1164.
[67] Jianshu Weng and Bu-Sung Lee. 2011. Event detection in twitter. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 5.
[68] Nannan Wu, Feng Chen, Jianxin Li, Jinpeng Huai, Baojian Zhou, Naren Ramakrishnan, et al. 2018. A nonparametric approach to uncovering connected anomalies by tree shaped priors. *IEEE Transactions on Knowledge and Data Engineering* 31, 10 (2018), 1849–1862.
[69] Ye Yuan, Xiang Lian, Guoren Wang, Yuliang Ma, and Yishu Wang. 2019. Constrained shortest path query in a large time-dependent graph. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1058–1070.