An Efficient Approach to Finding Dense Temporal Subgraphs

Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, Jinpeng Huai

Abstract—Dense subgraph discovery has proven useful in various applications of temporal networks. We focus on a special class of temporal networks whose nodes and edges are kept fixed, but edge weights regularly vary with timestamps. However, finding dense subgraphs in temporal networks is non-trivial, and its state of the art solution uses a filter-and-verification framework that is not scalable on large temporal networks. In this study, we propose a highly efficient approach to finding dense subgraphs in large temporal networks with T timestamps. (1) We first develop a statistics-driven approach that employs hidden statistics to identifying k time intervals, instead of $T(T+1)/2$ ones (k is typically much smaller than T), which strikes a balance between quality and efficiency. (2) After proving that the problem has no constant factor approximation algorithms, we design better heuristic algorithms to attack the problem, by connecting finding dense subgraphs with a variant of the Prize Collecting Steiner Tree problem. (3) Finally, we have conducted an extensive experimental study to verify that our approach is both effective and efficient.

1 Introduction

Dynamic behavior is an essential feature of many data analytic systems and applications that could be modeled as graphs or networks, such as social network analysis, biological data analysis, recommendation systems and route planning [1], [20]. Hence, it is not surprising that dynamic networks have drawn significant attentions from both industry and academic communities. In fact, dynamic networks also arise under many other terms, such as temporal networks, dynamic graphs, evolving networks, evolutionary networks, and graph streams [1], [5]–[7], [9], [10], [12], [14], [18], [21], [22], [27], [29], [30], [33], [37], [38].

Dense subgraph discovery and analysis have been widely studied in static networks [2], [3], [13], [15], [16], [23], [24], [36], such as finding maximal cliques, k-core analysis and the Prize Collecting Steiner Tree problem. It is worth pointing out that dense subgraphs are a very general concept, and their concrete semantics highly depend on the studied problems and applications. How to properly transfer or define their semantics over to temporal networks is still in the early stage, not to mention effective and efficient mining algorithms. Examples of such definitions include heavy subgraphs [6], hotspots [38], anomalies [5] and regions [7].

In this study, we investigate a special class of temporal networks (see a recent survey [20]), where the nodes and edges are fixed, but edge weights regularly vary with timestamps. Essentially, a temporal network with T timestamps can be viewed as T snapshots of a static network such that the network nodes and edges are kept the same among these T snapshots, while the edge weights vary with network snapshots. Road traffic networks typically fall into this category [6], [14], [30], [37], and road traffic analyses are of particular importance for large cities such as Beijing, New York, London and Paris. Indeed, large cities are facing with heavy traffic congestions, one of the great challenges of urban computing [5], [6], [30], [39].

We also focus on a certain form of dense temporal subgraphs, which was initially studied in [6]. Formally speaking, a temporal subgraph corresponds to a connected subgraph measured by the sum of all its edge weights in a time interval, *i.e.,* a continuous sequence of timestamps. Intuitively, such a dense subgraph may correspond to a collection of connected highly slow or jam roads (*i.e.,* a jam area) in road networks, lasting for a continuous sequence of snapshots. We next use an example to illustrate its basic idea, such as the real-time traffic status snapshot report which is regularly updated every couple of minutes. Given a sequence of such snapshots in a day period, dense temporal subgraphs help to analyze which areas during what time are in jam conditions.

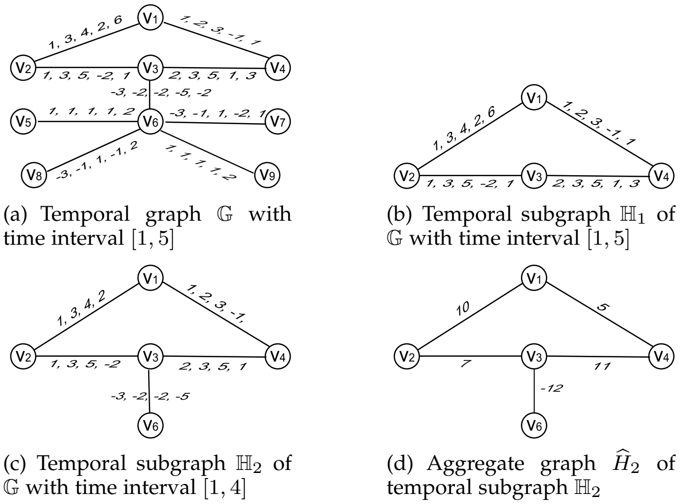
Challenges and limitations. However, the problem of finding dense subgraphs in temporal networks is non-trivial, and it is already NP-complete even for a temporal network with a single snapshot and with $+1$ or -1 edge weights only, as observed in [6]. Even worse, it remains hard to approximate for temporal networks with single snapshots (Section 4). Moreover, given a temporal network with T timestamps, there are a total of $T(T+1)/2$ time intervals to consider, which further aggravates the difficulty. Finally, the state of the art solution MEDEN [6] adopts a filter-and-verification framework so that *there often remain a large number of time intervals to verify even if a large portion of time intervals are filtered*. Hence, MEDEN is not scalable when temporal networks have a large number of nodes/edges or a large number T of timestamps.

Contributions. In this study, we propose a highly efficient and effective statistics-driven approach, instead of filter-and-verification, which employs hidden data statistics to find dense temporal subgraphs in large temporal networks.

(1) We first develop a statistics-driven approach to identify-

- *S. Ma, R. Hu, X. Lin and J. Huai are with the SKLSDE Lab, School of Computer Science and Engineering, Beihang University, China. E-mail: {mashuai, hurenjun, linxl, huaijp}@buaa.edu.cn.*
- *L. Wang is with Google Inc., Mountain View, CA, USA. This work was done when the author was at Beihang University. E-mail: luoshu@google.com.*

Manuscript received October, 2017; revised June, 2018 & November, 2018.



(a) Temporal graph $\mathbb{G}$ with time interval $[1, 5]$

(b) Temporal subgraph $\mathbb{H}_1$ of $\mathbb{G}$ with time interval $[1, 5]$

(c) Temporal subgraph $\mathbb{H}_2$ of $\mathbb{G}$ with time interval $[1, 4]$

(d) Aggregate graph $\widehat{H}_2$ of temporal subgraph $\mathbb{H}_2$

Fig. 1. Running example

ing k time intervals from a total of $T(T+1)/2$ ones (Section 3), where T is the number of snapshots and k is a small constant, *e.g.*, 10. This is achieved by exploring the characteristics of time intervals involved with dense subgraphs based on a novel *evolving convergence phenomenon*. This is also different from filter-and-verification, as it directly finds the most likely solutions, instead of filtering all impossible solutions and verifying the remaining unfiltered solutions one by one.

(2) We then design an algorithm for computing dense subgraphs given a time interval (Section 4). After showing that the problem has no constant factor approximation algorithms, we develop a heuristic algorithm (by proving the equivalence of finding dense subgraphs and finding maximum net worth subtrees, a variant of the Prize Collecting Steiner Tree problem [13], [23]) and three optimization techniques to improve both the effectiveness and efficiency.

(3) Using both real-life and synthetic data, we conduct an extensive experimental study (Section 5). (a) We find that the dense subgraphs found by our method FIDES$^+$ have the best quality, *i.e.*, about 100.20% and 100.15% on average of those found by the state of the art filter-and-verification solution MEDEN [6] on BJDATA and SYNDATA, respectively. (b) Our method FIDES$^+$ is on average 2,980 and 1,486 times faster than MEDEN on BJDATA and SYNDATA, respectively. (c) Finally, MEDEN already ran out of memory for temporal graphs with 150,000 nodes and 2,000 snapshots only.

2 PRELIMINARY

In this section, we introduce the basic definitions of temporal graphs and the problem to be investigated.

2.1 Basic Concepts

We first introduce basic concepts of temporal graphs.

Temporal graphs. A *temporal graph* $\mathbb{G}(V, E, W, L, U)$ is a weighted undirected graph with edge weights varying with timestamps (positive integers), where (1) V is a finite set of nodes, (2) $E \subseteq V \times V$ is a finite set of edges, in which (u, v) or $(v, u) \in E$ denotes an undirected edge between nodes u and v, (3) for each timestamp $t \in [L, U]$, $W^t : E \rightarrow \mathbb{R}$ is a weight function that maps each edge $e \in E$ to a *positive or negative* rational number, and (4) $[L, U]$ is a time interval representing $(U-L+1)$ timestamps, in which $L \leq U$ are the beginning and ending timestamps, respectively. When it is clear from the context, we simply use $\mathbb{G}(V, E, W)$ to denote temporal graphs for clarity.

We consider a special class of temporal networks (see a recent survey [20]) such as road networks and communication networks, where graph nodes and edges are kept fixed, but edge weights vary with respect to timestamps. Intuitively, (1) a temporal graph $\mathbb{G}(V,\ E,\ W)$ essentially denotes a sequence $< G_1(V, E, W^1), \ldots, G_T(V, E, W^T) >$ of $T = U - L + 1$ standard graphs, and (2) the edge weights $W^t(e)$ specify the distances, communication latencies or travelling duration [6], [14], [20], [30], [37], or the affinity or collaborative compatibility [15] between the two corresponding nodes of edges e at timestamps t. Essentially, positive/negative edge weights model opposite relations such as fast/congested traffic and friend/foe relationships.

We also say that $G_i(V, E, W^i)$ $(i \in [1, T])$ is a *snapshot* of temporal graph $\mathbb{G}(V, E, W)$ at timestamp $L + i - 1$.

Temporal subgraphs. A temporal graph $\mathbb{H}(V_s, E_s, W_s, L_s, U_s)$ is a *subgraph* of $\mathbb{G}(V, E, W, L, U)$, denoted by $\mathbb{G}[V_s, E_s, L_s, U_s]$, if $V_s \subseteq V$, $E_s \subseteq V_s \times V_s$, $E_s \subseteq E$, $L_s, U_s \in [L, U]$, and $W^t_s(e) = W^t(e)$ for any $t \in [L_s, U_s]$ and $e \in E_s$.

That is, subgraph $\mathbb{G}[V_s, E_s, L_s, U_s]$ only contains a subset of nodes and edges of graph $\mathbb{G}$, and it is restricted within the time interval $[L_s, U_s]$ that falls into interval $[L, U]$.

When $V_s = V$ and $E_s = E$, we also simply use $\mathbb{G}[L_s, U_s]$ to denote temporal subgraph $\mathbb{G}[V_s, E_s, L_s, U_s]$ for clarity.

Aggregate graphs. Given a temporal graph $\mathbb{G}(V, E, W, L, U)$, its *aggregate graph* $\widehat{G}(V, E, W_a)$ is a standard undirected weighted graph that has the same sets of nodes and edges as $\mathbb{G}$, and for each edge $e \in E$, its weight $W_a(e)$ is the sum of weights $W^t(e)$ with $t \in [L, U]$, *i.e.*, $W_a(e) = \sum_{t=L}^{U} W^t(e)$.

Cohesive and positive cohesive densities. For an aggregate graph $\widehat{G}(V, E, W_a)$, its *cohesive density*, denoted by $\mathsf{cden}(\widehat{G})$, is the sum of all the edge weights, *i.e.*, $\sum_{e \in E} W_a(e)$, and its *positive cohesive density*, denoted by $\mathsf{cden}^+(\widehat{G})$, is the sum of all the positive edge weights, *i.e.*, $\sum_{e \in E, W_a(e)>0} W_a(e)$.

The (positive) cohesive density of a temporal graph $\mathbb{G}$ is simply equal to the (positive) cohesive density of its corresponding aggregate graph $\widehat{G}$.

Dense subgraphs. Given a temporal graph $\mathbb{G}(V, E, W, L, U)$, its *dense subgraph* is a *connected temporal subgraph* $\mathbb{G}[V_s, E_s, L_s, U_s]$ with the *greatest cohesive density*. It is worth pointing out that any edges can be chosen as long as the resulting temporal subgraph is connected.

We next illustrate these concepts with an example.

Example 1: (1) Figure 1(a) depicts a temporal graph $\mathbb{G}$ with 9 nodes, 9 edges and 5 timestamps.
(2) Figure 1(b) shows a temporal subgraph $\mathbb{H}_1$ of $\mathbb{G}$ with time interval $[1, 5]$, and Fig. 1(c) shows a temporal subgraph $\mathbb{H}_2$ of $\mathbb{G}$ with time interval $[1, 4]$, respectively.
(3) Figure 1(d) shows the aggregate graph $\widehat{H}_2$ of temporal subgraph $\mathbb{H}_2$ whose cohesive density $\mathsf{cden}(\widehat{H}_2)$ = 21 and positive cohesive density $\mathsf{cden}^+(\widehat{H}_2)$ = 33, respectively.
(4) One can verify that the temporal subgraph $\mathbb{H}_1$ in Fig. 1(b) is indeed the only dense subgraph of $\mathbb{G}$ with the greatest cohesive density $\mathsf{cden}(\widehat{H}_1)$ = 44. □

```
Input: Temporal graph $\mathbb{G}(V, E, W, L, U)$.
Output: $\mathbb{G}[V_s, E_s, L_s, U_s]$, a solution of FDS.
1. for each time interval $[i, j]$ $(L \le i \le j \le U)$ do
2.     let $\widehat{G}_{[i,j]}(V, E, W_a)$ be the aggregate graph of $\mathbb{G}[i, j]$;
3.     $\mathsf{UB_{sop}[i,j]} := \mathsf{cden}^+(\widehat{G}_{[i,j]})$;
4. Estimate a lower bound LB for the solution of FDS;
5. for each time interval $[i, j]$ $(L \le i \le j \le U)$ do
6.     Prune $[i, j]$ if $\mathsf{UB_{sop}[i,j]} \le \mathsf{LB}$;
7. for each unpruned time interval $[i, j]$ $(L \le i \le j \le U)$ do
8.     $\widehat{G}'_{[i,j]} := \mathsf{topDown}(\widehat{G}_{[i,j]})$;
9. $\mathbb{G}[V_s, E_s, L_s, U_s]$ := a subgraph with the largest $\mathsf{cden}(\mathbb{G})$;
10. return $\mathbb{G}[V_s, E_s, L_s, U_s]$.
```

Fig. 2. Algorithm basicMEDEN

2.2 Finding Dense Subgraphs

We next present the problem statement and its baseline solutions [6]. Given a temporal graph $\mathbb{G}(V, E, W)$, the problem of finding dense subgraphs (referred to as FDS) is to find a *connected temporal subgraph* $\mathbb{H} = \mathbb{G}[V_s, E_s, L_s, U_s]$ that has the *greatest* cohesive density $\mathsf{cden}(\mathbb{H})$.

Intractability. It is already known that the FDS problem is intractable, as observed in [6].

Proposition 1: *The problem of finding dense subgraphs is* NP-*complete, even for a temporal network with a single snapshot and with only* $+1$ *or* -1 *edge weights [6].* □

Baseline solutions. We next present the details of algorithm basicMEDEN developed in [6], shown in Fig. 2. Here procedure topDown aims to find a subgraph of an aggregate graph with a higher cohesive density.

Algorithm basicMEDEN. Given a temporal graph $\mathbb{G}$, it returns a solution of FDS. It first computes an upper bound $\mathsf{UB_{sop}[i,j]}$ for each $[i, j]$ of the $T(T+1)/2$ time intervals $(L \le i \le j \le U)$ (lines 1–3). It then uses procedure topDown to compute the solutions for the k time intervals $[i, j]$ that have the top–k highest $\mathsf{UB_{sop}[i,j]}$, and sets LB to the highest cohesive density of the k computed dense subgraphs (line 4). Using LB and $\mathsf{UB_{sop}[i,j]}$, it prunes time intervals (lines 5–6), and uses topDown again to compute the solutions for all unpruned time intervals (lines 7–8). The subgraph found with the greatest cohesive density is finally returned as the dense subgraph (lines 9-10).

Remarks. (1) The state of the art solution MEDEN [6] adopts a filter-and-verification framework. For clarity, here we only present the basic version basicMEDEN. The sophisticated version MEDEN incorporates a more scalable filtering technique by grouping time intervals, and it was reported that MEDEN achieved an order of magnitude performance improvement over basicMEDEN [6]. (2) We shall compare our approach with the sophisticated version MEDEN in the experimental study (Section 5). (3) It is easy to see that the FDS problem is solvable in polynomial time when all edge weights are non-negative.

3 Identifying Time Intervals

Our efficient statistics-driven approach to finding dense subgraphs in temporal graphs consists of two key components: (1) identifying k time intervals and (2) finding a dense subgraph for a given time interval. We first introduce how to identify k time intervals based on a novel evolving convergence phenomenon. We consider a temporal graph $\mathbb{G}(V, E, W, L, U)$, and assume without loss of generality that $L = 1$ and $U = T$ in the sequel.

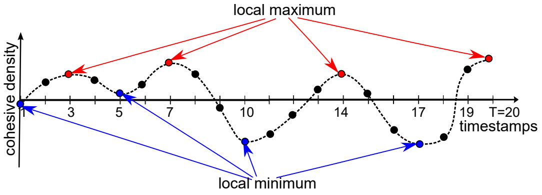


Fig. 3. Local minima and maxima

Observe that there are $T(T+1)/2$ time intervals in total, and even if T is not a large number, *e.g.*, 2,000, there are more than 2×10^6 time intervals to investigate, which involves too much computational cost. Even though MEDEN filters a large portion of unnecessary time intervals, say 99% [6], there remain a large number of time intervals to verify, *e.g.*, 2×10^4 in the above case. Recall that the FDS problem remains NP-complete for single snapshots (Proposition 1, Section 2.2). Instead of filter-and-verification, we develop a statistics-driven approach to exploring k time intervals only, aiming at striking a balance between quality and efficiency. Here k is typically a small constant, *e.g.*, 10 or 15.

3.1 Characteristics of Time Intervals

We first present the key characteristics of the time intervals involved with dense subgraphs.

Cohesive density curves. Given a temporal graph $\mathbb{G}(V, E, W)$, its *cohesive density curve* is a function $y = f(x)$ such that $f(x) = (\mathsf{cden}(\mathbb{G}[x, x]) - \mu)/\sigma$ for each $x \in [1, T]$, where μ and σ are the mean value and standard deviation of $\mathsf{cden}(\mathbb{G}[x, x])$ $(x \in [1, T])$, respectively. That is, we use normalized cohesive densities with mean values 0 and standard deviations 1 in the cohesive density curves.

Local maximum and minimum. We consider a cohesive density curve $y = f(x)$ of temporal graph $\mathbb{G}(V, E, W)$.

The curve y is said to have a *local maximum* at a point x^* for a given positive integer $\delta \ge 1$ if $f(x^*) \ge f(x)$ holds for all x with $|x - x^*| \le \delta$.

Similarly, y is said to have a *local minimum* at a point x^* for a given positive integer $\delta \ge 1$ if $f(x^*) \le f(x)$ holds for all x with $|x - x^*| \le \delta$.

Example 2: Consider the cohesive density curve $y = f(x)$ of temporal graph $\mathbb{G}$ in Fig. 3. Here for $\delta = 1$, the curve y has a local maximum at points $x = 3, 7, 14, 20$ and a local minimum at points $x = 1, 5, 10, 17$, respectively. □

We now present the first key observation inspired by convergent evolution in evolutionary biology. Based on this, we prove a very important characteristic of the time intervals involved with dense subgraphs.

Evolving convergence phenomenon. Consider a sequence $< G_1(V, E, W^1), \ldots, G_T(V, E, W^T) >$ of the T snapshots of temporal graph $\mathbb{G}(V, E, W)$.

The evolving convergence phenomenon asserts that if there exists an edge in G_i $(i \in [1, T-1])$ whose weight is no less (resp. no greater) than its weight in G_{i+1}, then for all edges, their weights in G_i are no less (resp. no greater) than their corresponding weights in G_{i+1}. Intuitively, this says that all edges evolve in a convergent manner, *i.e.*, the increase of one edge weight indicates that all the remaining edge weights do not decrease, and vice versa.

Proposition 2: *To find the dense subgraph, we only need to consider the time intervals $[i, j]$ such that the cohesive density curve has a local maximum at certain point between i and j under the evolving convergence phenomenon.* □

The evolving convergence phenomenon assures the correctness of Proposition 2, which gives a precise characteristic of the time intervals involved with dense subgraphs. It may not completely hold, but remains effective to a large extent in practice. Moreover, it is indeed a statistical characteristic which captures the general tendency of edge weight changes on certain temporal graphs.

Considering the Beijing road network for instance, the phenomenon *almost holds*. There are morning and evening peaks at Beijing, and it is common that a majority of roads become slow/jam during the peak time, and enter a faster traffic condition after the peak time ends, although individual roads may be *physically isolated*. In this case, it is also easy to see that it is very likely that the dense subgraph lies in a time interval containing peaks.

Moreover, it is trivial to verify the following two characteristics, which further help us to identify the time intervals involved with dense subgraphs.

Fact 1: *All dense subgraphs have a non-negative cohesive density, which equals to zero for a subgraph without any edges.* □

Intuitively, *Fact 1* tells us that dense subgraphs are more concerned with the positive weight edges, and the positive cohesive density may give better estimations for the potential of time intervals, compared with the cohesive density.

Heuristic 2: *Temporal subgraph $\mathbb{G}[i, j]$ $(i \leq j \in [1, T])$ with a higher positive cohesive density in general is a good candidate subgraph to contain a dense subgraph.* □

The positive cohesive density essentially provides an upper bound of the cohesive density that any subgraph can achieve. And *Heuristic 2* tells us that the temporal subgraph with the highest positive cohesive density is *very likely to* contain the dense subgraph that we are looking for. Instead of the highest one only, we further compute the time intervals whose corresponding temporal subgraphs have the top–k highest positive cohesive densities from those $[i, j]$ containing a local maximum.

Remarks. As will be shown by the experiments (Section 5), the three characteristics (*i.e.,* Proposition 2, Fact 1 and Heuristic 2) together assure a pretty good estimation of the time intervals involved with dense subgraphs, even when the phenomenon does not completely, but almost holds.

3.2 Computing Top-K Time Intervals

We finally present our methods to estimate the top–k time intervals whose aggregate graphs have the greatest positive cohesive densities, following the analysis in Section 3.1.

Algorithm maxTInterval uses local maxima to identify the time intervals, and is presented in Fig. 4.

Given a temporal graph $\mathbb{G}(V, E, W)$ and two positive integers k and δ, it outputs a set of $k/2$ aggregate graphs with the largest positive cohesive densities. (1) It first computes the h local maxima of the cohesive density curve of $\mathbb{G}$ (lines 1–2). (2) For each local maximum x_i $(i \in [1, h])$, it finds two timestamps, *i.e.,* lower bound $x_i.l$ and upper bound $x_i.u$, such that $x_i.l \leq x_i \leq x_i.u$ and $f(x)$ $(x \in [x_i.l, x_i.u])$ remains relatively high (line 3). More specifically, it finds the largest timestamp $x \leq x_i$ such that $|f(x) - f(x-1)| \geq \xi(x)$ and the smallest timestamp $x \geq x_i$ such that $|f(x) - f(x+1)| \geq \xi(x)$, where $\xi(x) = \Delta(x)e^{f(x)}$ is an *adaptive* smoothing threshold that considers both the normalized cohesive density $f(x)$ and density change. Here $\Delta(x)$ is the smaller of the average density change of the entire curve and the local average change within c timestamps from x, *i.e.,* $\frac{1}{2c}\sum_{i=x-c}^{x+c-1} |f(i) - f(i+1)|$, $\Delta(x)$ is scaled using $e^{f(x)}$ as $f(x)$ can be either positive or negative, and c is a small constant, *e.g.,* 2. (3) It then repeatedly merges overlapped time intervals by removing local maxima (line 4). The intuition behind is that close maxima can be treated as one with wider range. (4) Using $x_i.l$ and $x_i.u$ $(i \in [1, h])$, it generates a set S of $h(h+1)/2$ time intervals (line 5). Note that time interval $[x_i.l, x_j.u]$ $(i \leq j)$ must contain a local maximum x_i. (5) After this, a subset of $2k$ time intervals whose aggregate graphs have the top–2k largest positive cohesive densities are selected and further tuned (lines 6–8). For time interval $[l, u] \in S_{2k}$, it considers four tuning candidates $[l \pm \eta, u \pm \eta]$ simultaneously, and uses the one giving the largest positive cohesive density to replace $[l, u]$ if needed (line 8). The step size η is first fixed to 1, and is doubled every 4 replacements. (6) It finally computes $S_{k/2}$ as the top–k/2 time intervals in S_{2k} and returns the $k/2$ aggregate graphs given time intervals in $S_{k/2}$ (lines 9–11).

```
Input: Temporal graph $\mathbb{G}(V, E, W)$, positive integers $k$ and $\delta$.
Output: $k/2$ aggregate graphs $\widehat{G}$ with the largest $\mathsf{cden}^+(\widehat{G})$.
1. let $y = f(x)$ be the cohesive density curve of $\mathbb{G}$;
2. let $x_1 < \ldots < x_h$ be the local maxima of $y$;
3. let $x_i.l/x_i.u$ be the lower/upper bounds of $x_i$ $(i \in [1, h])$;
4. Merge overlapped time intervals $[x_i.l, x_i.u]$ and $[x_j.l, x_j.u]$;
5. $S := \{[x_i.l, x_j.u] \mid 1 \leq i \leq j \leq h\}$;
6. $S_{2k}$ := the top–2k time intervals in $S$;
7. for each time interval $[l, u] \in S_{2k}$ do
8.    Tune $[l, u]$ to $[l', u']$ s.t. $\mathsf{cden}^+(\mathbb{G}[l', u']) > \mathsf{cden}^+(\mathbb{G}[l, u])$;
9. $S_{k/2}$ := the top–k/2 time intervals in $S_{2k}$;
10. $R := \{\mathbb{G}[l, u] \mid [l, u] \in S_{k/2}\}$;
11. return the aggregate graphs of temporal graphs in $R$.
```

Fig. 4. Algorithm maxTInterval using local maxima

Algorithm minTInterval is the counterpart of maxTInterval, and it uses local minima to identify the remaining top–k/2 time intervals, as two local minima contain a local maximum, another form of time intervals. It is along the same lines as algorithm maxTInterval except the following: It (1) computes h local minima $x_1 < \ldots < x_h$ (line 2), (2) produces a set S of $h(h-1)/2$ time intervals in the form of $\{[x_i.u, x_j.l] \mid 1 \leq i < j \leq h]\}$ (line 5), and (3) sets the smoothing threshold $\xi(x) = \Delta(x)e^{-f(x)}$ to assure that $f(x)$ $(x \in [x_i.l, x_i.u])$ remains relatively low.

We next use an example to illustrate how to generate the top–k time intervals using local maxima and minima.

Example 3: Consider the curve $y = f(x)$ in Fig. 3 again.
(1) Assume without loss of generality that $[x.l, x.u]$ is $[2, 4]$, $[6, 8]$, $[13, 15]$, $[19, 20]$ for the local maxima at points x = 3, 7, 14, 20, respectively. Here no local maxima can be merged. Then the set S at line 5 of maxTInterval is $\{[2, 4]$, $[2, 8]$, $[2, 15]$, $[2, 20]$, $[6, 8]$, $[6, 15]$, $[6, 20]$, $[13, 15]$, $[13, 20]$, $[19, 20]\}$. If $[2, 8] \in S_{2k}$ and $\mathsf{cden}^+(\mathbb{G}[1, 8])$>$\mathsf{cden}^+(\mathbb{G}[2, 8])$, then maxTInterval replaces $[2, 8]$ with $[1, 8]$.
(2) Assume without loss of generality that $[x.l, x.u]$ is $[1, 2]$,

```
Input: Aggregate graph $\widehat{H}(V, E, W_a)$.
Output: Converted graph $\vec{H}(V', E', p, w)$.
1. $\widehat{H}^+ := \widehat{H}$ with non-negative edges only;
2. Compute the connected components $\text{CC}_1, \ldots, \text{CC}_l$ of $\widehat{H}^+$ ;
3. let $V' := \{u_1, \ldots, u_l\}$;
4. for each $i \in [1, l]$ do $p(u_i)$ := the total edge weight of $\text{CC}_i$;
5. if there are negative edges between $\text{CC}_i$ and $\text{CC}_j$ $(i, j \in [1, l])$
6. then $E' := E' \cup \{(u_i, u_j)\}$;
7.     $w(u_i, u_j)$ := |the largest negative edge weight|;
8. return $\vec{H}(V', E', p, w)$.
```

Fig. 5. Procedure convertAG

$[4, 6]$, $[9, 11]$, $[16, 18]$ for the local minima at points x = 1, 5, 10, 17, respectively. Then the set S of algorithm minTInterval is $\{[2, 4], [2, 9], [2, 16], [6, 9], [6, 16], [11, 16]\}$. If time interval $[6, 16] \in S_{2k}$ and $\text{cden}^+(\mathbb{G}[6, 15]) > \text{cden}^+(\mathbb{G}[6, 16])$, then minTInterval replaces $[6, 16]$ with $[6, 15]$. □

Time complexity analysis. Algorithms maxTInterval and minTInterval both run in $O((T + h^2)|E|)$ time, in which h is the number of local maxima or minima.

Observe the following. (1) It takes $O(T|E|)$ time to generate the cohesive density curve (line 1). (2) When selecting the top–2k time intervals based on positive cohesive densities (line 6), it takes $O(h^2|E|)$ time by precomputing $W_\Sigma(e, t) = \sum_{i=1}^{t} W^i(e)$ for each edge e and timestamp t. (3) It takes $O(k|E|\log T)$ time to tune time intervals (lines 7–8) since each one can be updated at most $O(\log T)$ times. Putting these together, we have the conclusion. And, similarly, algorithm minTInterval runs in $O((T + h^2)|E|)$ time as well. Note that here k is a small constant, *e.g.*, 10 or 15, and h is typically much smaller than T.

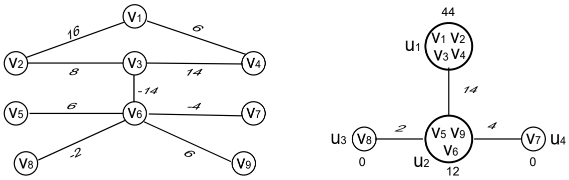
Remarks. Different from our earlier work [26], the cohesive density curve is plotted with the normalized cohesive densities, and the top–k time interval estimation algorithms are refined by: (a) determining more accurate time intervals with an *adaptive* smoothing threshold, instead of a *fixed* one, and (b) revising the tuning strategy. As will be shown in the experimental study (Section 5), these help to identify better time intervals of dense subgraphs.

4 COMPUTING DENSE SUBGRAPHS

We now explain how to compute the dense subgraph for a given time interval. This reduces to the problem of finding the subgraph of an aggregate graph with the highest cohesive density, which remains NP-hard as observed in [6]. We first show that the problem has no constant factor approximation algorithms, and then connect the problem of finding dense subgraphs in an aggregate graph with the *Net Worth Maximization problem* (NWM), a variant of the Prize Collecting Steiner Tree problem [13], [23]. Hence, we develop algorithm heuristics to attack the problem. We finally present our complete solution FIDES$^+$ for finding dense subgraphs in temporal networks.

4.1 Approximation Hardness

The hardness is verified by a reduction from the *Net Worth Maximization* (NWM) problem, a variant of the Prize Collecting Steiner Tree problem [13], [23]. Given an undirected graph $G(V, E)$, a non-negative edge weight $w(e)$ for each $e \in E$ and a non-negative node weight $p(v)$ for each $v \in V$, the NWM problem is to find a subtree $ST(V_{st}, E_{st})$ that maximizes its net worth $NW(ST) = \sum_{v \in V_{st}} p(v) - \sum_{e \in E_{st}} w(e)$. It is already known that the NWM problem is NP-complete, and is hard to approximate: it is NP-hard to approximate the optimal net worth within any constant factor [23]. Finding an optimal dense subgraph of an aggregate graph is also non-trivial, as shown below.



(a) Aggregate graph $\widehat{G}$ (b) Converted graph $\vec{G}$

Fig. 6. Example of converted graphs

Theorem 3: *The cohesive density achieved by an optimal subgraph is* NP*-hard to approximate within any constant factor.* □

We utilize the *approximation factor preserving reduction* (AFP-reduction) that retains approximation bounds [11], [35], and show that there exists an AFP-reduction from the NWM problem to the problem of finding the dense subgraph of an aggregate graph, from which Theorem 3 follows.

4.2 Connections with the NWM Problem

Theorem 3 tells us that heuristic algorithms are essentially the practical solutions on which we should focus, as its counterpart the NWM problem does [23]. We shall reduce the problem of finding dense subgraphs in an aggregate graph with *positive or negative* edge weights to the NWM problem, based on a notion of *converted graphs* that are undirected graphs with *non-negative* node and edge weights.

We next present the details of procedure convertAG in Fig. 5, which takes as input an aggregate graph $\widehat{H}(V, E, W_a)$, and returns its converted graph $\vec{H}(V', E', p, w)$ with non-negative node and edge weights.

Procedure convertAG first generates graph $\widehat{H}^+$ by removing all the edges in $\widehat{H}$ with negative weights (line 1). It then computes the connected components of $\widehat{H}^+$ (line 2). For each connected component CC_i $(i \in [1, l])$, there is a corresponding node u_i in $\vec{H}$, whose weight $p(u_i)$ is equal to the total edge weight of CC_i (lines 3-4). An edge (u_i, u_j) $(i, j \in [1, l])$ is included in $\vec{H}$ if there are negative weight edges between CC_i and CC_j in $\widehat{H}$, and the edge weight $w(u_i, u_j)$ is the absolute value of the *largest negative* edge weight among all edges across CC_i and CC_j (lines 5-7). Finally, the converted graph $\vec{H}$ is returned (line 8).

Example 4: Consider the temporal graph $\mathbb{G}$ in Fig. 1(a) again, and its aggregate graph $\widehat{G}$ in Fig. 6(a). The converted graph $\vec{G}$ of $\widehat{G}$ computed by convertAG is shown in Fig. 6(b). □

With a close look at procedure convertAG, it is easy to verify the following, which connects the dense subgraphs in an aggregate graph with the maximum net worth subtrees in its converted graph.

Proposition 4: *Finding a dense subgraph in an aggregate graph* $\widehat{H}$ *is equivalent to finding a maximum net worth subtree in the converted graph* convertAG$(\widehat{H})$. □

Remarks. (1) It is worth mentioning that convertAG *reduces the size of aggregate graphs*, which further helps to improve

```
Input: Converted graph $\vec{H}$.
Output: Merged converted graph $\vec{H}'$.
1. let $\vec{H}'$ := $\vec{H}$;
2. while there are nodes in $\vec{H}'$ that can be merged do
3.    Merge $u$ and $v$ into a single node $x$;
4.    $p(x) := p(v) + p(u) - w(u, v)$;
5.    Remove edge $(u, v)$ from $\vec{H}'$;
6.    Replace all edges $(u, y)$ and $(v, y)$ with $(x, y)$;
7.       $w(x, y) := \min(w(u, y), w(v, y))$;
8. return $\vec{H}'$.
```

Fig. 7. A basic version of procedure strongMerging

the efficiency of finding dense subgraphs. (2) Different from aggregate graphs, converted graphs have *non-negative* node and edge weights only, as illustrated by Example 4.

4.3 Algorithm Optimizations

Proposition 4 tells us that the algorithm of the NWM problem [23] provides us a basic solution. We next investigate optimization techniques to find the dense subgraphs of aggregate graphs, by finding the maximum net worth subtrees in the corresponding converted graphs.

(I) Strong merging. After having a converted graph, we repeatedly merge two neighboring nodes if the resulting merged node has a weight no less than the two weights before merging. Intuitively, those neighboring nodes belong to the same maximum net worth subtree. This technique is to reduce the size of the converted graph for speeding up the process of identifying dense subgraphs.

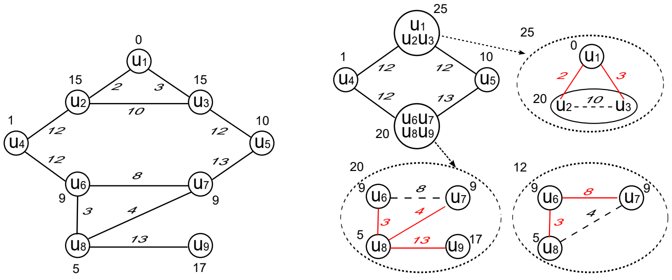
We next present the details of a basic version of procedure strongMerging in Fig. 7. It takes as input a converted graph $\vec{H}$, and returns its merged converted graph $\vec{H}'$. It repeatedly merges two neighboring nodes u and v if both $p(u) \geq w(u, v)$ and $p(v) \geq w(u, v)$ hold (lines 1–7) and then returns the merged converted graph $\vec{H}'$ (line 8).

However, there may exist different ways to compute the weights of the merged nodes, and two nodes may be merged under a more complex condition, as shown below.

Example 5: (1) Consider the converted graph $\vec{H}$ in Fig. 8(a). (a) When nodes u_6, u_7, u_8 are merged using edges (u_6, u_7) and (u_6, u_8), $p(\{u_6, u_7, u_8\})$ equals to 12, and node u_9 cannot be further merged; (b) When nodes u_6, u_7, u_8 are merged using edges (u_6, u_8) and (u_7, u_8), $p(\{u_6, u_7, u_8\})$ equals to 16, and node u_9 can further be merged, as shown in Fig. 8(b). That is, the way how nodes are merged has effects on both the node weights and the merging process.
(2) One can easily check that basic strongMerging merges nodes u_2 and u_3 with $p(\{u_2, u_3\}) = 20$. Moreover, as shown in Fig. 8(b), node $\{u_2, u_3\}$ can further be merged with u_1 by using edges (u_1, u_2) and (u_1, u_3) and removing edge (u_2, u_3). The resulting node weight $p(\{u_1, u_2, u_3\}) = 25$ is higher than both $p(\{u_2, u_3\})$ and $p(u_1)$. □

To address these, we maintain a Minimum Spanning Tree (MST) for each merged node in the process of procedure strongMerging, which leads to the need of merging two MSTs (line 3). Hence, procedure strongMerging further uses Sleator–Tarjan dynamic trees [34] to achieve a good performance, as shown below.

Proposition 5: *Given a converted graph $\vec{H}(V', E')$, the extra cost of maintaining* MSTs *for procedure* strongMerging *is bounded by $O(|E'| \log |V'|)$.* □



(a) Converted graph $\vec{H}$ (b) Merged converted graph $\vec{H}'$

Fig. 8. Example for strong merging

For convenience, the edges in a converted graph are classified into (a) *internal edges*, the edges of MSTs associated with merged nodes, (b) *external edges*, those edges not associated with any MST, and (c) *across*-MST edges, the external edges connecting two distinct MSTs.

We first introduce the rules for node merging.

(1) We say that two (merged) nodes u and v are *type-I* mergeable if there exists an external edge (u, v) such that $\min\{p(u), p(v)\} \geq w(u, v)$. That is, connecting u and v with an external edge (u, v) forms a new type-I merged node.

(2) We say that two (merged) nodes u and v are *type-II* mergeable if there exist (a) two external edges (u_1^c, v_1^c) and (u_2^c, v_2^c) connecting the associated MSTs of u and v, respectively, and (b) an internal edge e in the MST associated with u or v such that $p(u) + p(v) + w(e) - w(u_1^c, v_1^c) - w(u_2^c, v_2^c) \geq \max\{p(u), p(v)\}$. That is, connecting u and v with two external edges (u_1^c, v_1^c) and (u_2^c, v_2^c) and removing an internal edge e form a new type-II merged node.

We then explain the complete procedure strongMerging.
(1) It considers type-II mergeable pairs only when no type-I pairs are left, as type-II mergeable needs merged nodes, and it is designed as a complement of type-I mergeable to adjust the edges used for merging. For type-I pairs, if one node belongs to an optimal subtree, the other must as well. This is not true for type-II pairs whose process is order sensitive. For instance, suppose adding node u_1' in Fig. 8(a) with $p(u_1') = 0$ and $w(u_1', u_2) = w(u_1', u_3) = 3$, then u_1' can also be type-II merged with $\{u_2, u_3\}$, which is a sub-optimal solution. However, node u_1 cannot be merged any more.
(2) It maintains across-MST edges for the convenience of merging MSTs. It computes the degree $deg(u) = \sum_{u^c \in u} deg(u^c)$ of (merged) node u, where u^c is a converted graph node of u, *e.g.*, $deg(\{u_1, u_2, u_3\}) = 2 + 3 + 3$ in Fig. 8, and always merges across-MST edges from lower to higher degree nodes. In this way, each edge is merged for at most $\lfloor \log |E'| \rfloor + 1$ times for converted graph $\vec{H}(V', E')$. Moreover, the across-MST edges of each merged node are stored in a balanced binary tree, with a total maintaining cost of $O(|E'| \log^2 |V'|)$.
(3) Once nodes u and v are merged, new type-I pairs are further identified by verifying the across-MST edges of u (resp. v) whose weights fall into the range of $(p(u), p(\{u, v\})]$ (resp. $(p(v), p(\{u, v\})]$).
(4) The process of type-II pairs is order sensitive. After all type-I pairs are processed, procedure strongMerging sorts all across-MST edges by the higher node weights, and then by the lower node weights of the two end nodes. This gives priority to further merging nodes of higher weights. It

```
Input: Minimum spanning tree $T$ of $\vec{H}'$.
Output: An optimal subtree $ST$ of $T$.
1. Randomly select a node as the root of $T$;
2. $nw(u) := p(u)$ $(u \in T)$;
3. for each node $u$ in $T$ in a bottom-up fashion do
4.    for each child node $v$ of $u$ do
5.       if $nw(v) < w(u, v)$ then remove edge $(u, v)$;
6.       else $nw(u) := nw(u) + nw(v) - w(u, v)$;
7. $u_r$ := $\mathrm{argmax_u}\{nw(u)\}$;
8. return the subtree $ST$ rooted at $u_r$.
```

Fig. 9. Procedure strongPruning

then processes each across-MST edge, and merges the two MSTs, if possible, where external edge (u_1^c, v_1^c) is the edge to process, external edge (u_2^c, v_2^c) is set to the lowest-weight edge between the two MSTs, and the removed interval edge e is fixed to the highest-weight edge in the circle formed by the two MSTs and external edges (u_1^c, v_1^c) and (u_2^c, v_2^c). Both the sorting and the processing steps cost $O(|E'| \log |V'|)$ time. Note that this step may also find new type-I pairs.

Combining these, one can easily verify that procedure strongMerging runs in $O(|E'| \log^2 |V'|)$ time.

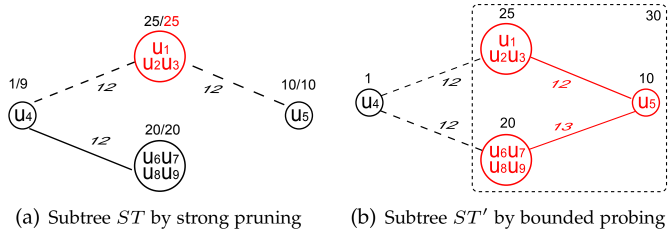
(II) Strong pruning. Strong pruning is an effective technique for solving the NWM problem developed in [23], [28], which finds an optimal net worth subtree containing a specified root node. Hence, we revise and utilize the improved strong pruning technique that eliminates the restriction of containing a specified root node [28]. Given an arbitrary tree T, it produces a subtree ST of T that maximizes its net worth $NW(ST)$ among all subtrees of T. This technique is to identify the dense subgraphs of aggregate graphs.

We next present the details of procedure strongPruning in Fig. 9 that given an MST T of a merged converted graph generated by procedure strongMerging, returns the maximum net worth subtree ST of T. It first reconstructs T by randomly choosing a root node (line 1). It then initializes the $nw(u)$ of each node u as its $p(u)$ (line 2). All nodes u are further updated in a bottom-up manner, *i.e.,* u is updated if and only if all its child nodes have already been updated. When it updates node u, each of the child nodes of u is processed independently: (a) If $nw(v) \geq w(u, v)$, it replaces $nw(u)$ with $nw(u) + nw(v) - w(u, v)$, and (b) otherwise, it removes edge (u, v) from T (lines 3–6). Finally, the subtree rooted at the node u_r with the maximum $nw(u_r)$ in the remaining T is returned (lines 7-8).

In the following, we use an example to illustrate the process of procedure strongPruning.

Example 6: Consider the merged converted graph $\vec{H}'$ in Fig. 8(b). Its MST T and the optimal subtree ST produced by strongPruning are shown in Fig. 10(a), in which x and y of labels x/y denote $p(u)$ and $nw(u)$ of nodes u, respectively, and dashed edges denote removed edges in the process.

Procedure strongPruning first randomly selects $\{u_1, u_2, u_3\}$ as the root of T. For each node u in T, its net worth $nw(u)$ is initialized with $p(u)$. The net worth $nw(\{u_5\})$ and $nw(\{u_6, u_7, u_8, u_9\})$ need not to be updated as both of them have no child nodes. The first modification is on node $\{u_4\}$, whose net worth $nw(\{u_4\})$ is replaced by $nw(\{u_4\}) + nw(\{u_6, u_7, u_8, u_9\}) - w(\{u_4\}, \{u_6, u_7, u_8, u_9\})$. Then for node $\{u_1, u_2, u_3\}$, its attached two edges are removed since $nw(\{u_4\}) < w(\{u_1, u_2, u_3\}, \{u_4\})$ and $nw(\{u_5\}) < w(\{u_1, u_2, u_3\}, \{u_5\})$. Finally, the subtree rooted at $\{u_1, u_2, u_3\}$ is returned, which is indeed $\{u_1, u_2, u_3\}$ itself, as its net worth $nw(\{u_1, u_2, u_3\})$ is the maximum. □



(a) Subtree ST by strong pruning (b) Subtree ST' by bounded probing

Fig. 10. Examples for strong pruning and bounded probing

Based on [28], one can easily check the following result.

Corollary 6: *Given a tree* $T(V_T, E_T)$*, procedure* strongPruning *takes* $O(|V_T|)$ *time to find the maximum net worth subtree.* □

(III) Bounded probing. For the subtree found by procedure strongPruning, we propose a bounded probing technique to find another subtree with a higher net worth. This technique is to further optimize the dense subgraphs with a limited extra cost. We first illustrate this with an example below.

Example 7: Consider the subtree ST, *i.e.,* the single node $\{u_1, u_2, u_3\}$, found by procedure strongPruning in Fig. 10(a), where all nodes can no longer be merged. But, as shown in Fig. 10(b), after adding nodes $\{u_5\}$ and $\{u_6, u_7, u_8, u_9\}$ along the path connecting them, we indeed have a subtree with a higher net worth $25 + 10 + 20 - 12 - 13 = 30$. □

We next present the details of boundedProbing, shown in Fig. 11. Given a merged converted graph $\vec{H}'$ and the subtree ST of $\vec{H}'$ found by strongPruning, it returns another subtree ST' of $\vec{H}'$ with a higher net worth. It first probes the set B of nodes in the merged converted graph $\vec{H}'$ (but not in the subtree ST) that can reach certain nodes in ST within r hops, where r is a small constant, *e.g.,* 4 (lines 2–3). For each node $u \in B$, it greedily chooses a path that connects it with ST and gives the highest increment of the net worth if merging the path into ST (lines 4–6). For each node $u \in B$ in the descending order of $NW^+(u)$, it merges path $P(u)$ into ST if $NW^+(u) \geq 0$ and nodes in $V^+(u)$ are not in ST yet (lines 7–9). The above process is repeated r times, and it finally returns the subtree ST as ST'.

Remarks. An alternative way for bounded probing is to compute several paths for each node $u \in B$, and to extend ST using these paths. Here we choose to compute the best paths and repeat the process r times. In this way, it not only potentially finds better paths after new nodes are included in a subtree, but also finds more extending paths starting from the newly included nodes in a subtree.

4.4 Algorithm for Aggregate Graphs

We now present the algorithm to compute the *Aggregate graph's Dense Subgraphs*, referred to as compADS$^+$. A basic version of algorithm compADS$^+$ is shown in Fig. 12, which takes as input an aggregate graph $\widehat{H}$, and returns the dense subgraph of $\widehat{H}$, using the three optimizations in Section 4.3.

Algorithm compADS$^+$ first transforms aggregate graph $\widehat{H}$ into its converted graph $\vec{H}$ (line 1), and then produces a merged converted graph $\vec{H}'$ using the strong merging technique (line 2). An MST T of $\vec{H}'$ is computed (line 3), and a subtree ST of T is produced using the strong pruning technique (line 4), which is further optimized to ST'

```
Input: Merged converted graph $\vec{H}'$, subtree $ST$ of $\vec{H}'$.
Output: Subtree $ST'$ of $\vec{H}'$ with a higher net worth.
1.  repeat for $r$ times:
2.    let $V_H$ and $V_{ST}$ be the node sets of $\vec{H}'$ and $ST$;
3.    $B := \{u \mid u \in V_H \setminus V_{ST}, \exists v \in V_{ST}\ \mathsf{dist}(u,v) \le r\}$;
4.    for each $u \in B$ do
5.      let $P(u)$ be the simple path $v_0/v_1/\ldots/v_L$ such that
        $L \le r$, $v_0 \in V_{ST}$, $v_L = u$, $\{v_1,\ldots,v_L\} \cap V_{ST} = \emptyset$, and
        $NW^+(u) := \sum_{i=1}^{L} \big(p(v_i) - w(v_{i-1}, v_i)\big)$ is maximized;
6.      $V^+(u) := \{v_1,\ldots,v_L\}$;
7.    for each $u \in B$ in the descending order of $NW^+(u)$ do
8.      if $NW^+(u) \ge 0$ and $V^+(u) \cap V_{ST} = \emptyset$
9.      then Merge path $P(u)$ into ST; $V_{ST} := V_{ST} \cup V^+(u)$;
10. return the subtree $ST$ as $ST'$.
```

Fig. 11. Procedure boundedProbing

with the bounded probing technique (line 5). A minimum spanning subtree ST'' of $\vec{H}$ is computed that has the same set of nodes as ST' (line 6). Finally, the subgraph of $\widehat{H}$ corresponding to the ST'' is returned (line 7).

We next show how algorithm compADS$^+$ finds the dense subgraph of an aggregate graph with an example.

Example 8: Consider the converted graph $\vec{H}$ shown in Fig. 8(a). Algorithm compADS$^+$ computes its merged converted graph $\vec{H}'$ shown in Fig. 8(b). The MST T of $\vec{H}'$ is then computed, as shown in Fig. 10(a). With strongPruning, it finds the optimal subtree of T, *i.e.*, the single $\{u_1, u_2, u_3\}$ in T. Using boundedProbing, it finds another subtree ST', having nodes $\{u_1, u_2, u_3\}$, $\{u_5\}$ and $\{u_6, u_7, u_8, u_9\}$, with a higher net worth, as shown in Fig. 10(b). This tree is indeed the MST in converted graph $\vec{H}$, and, finally, the subgraph corresponding to ST' is returned. □

Recall that the strong merging technique directly merges two neighboring nodes using the edge between them. This might give worse results when there exist paths that are better than the incident edges for merging. Hence, algorithm compADS$^+$ further computes dense subgraphs without using the strong merging technique (lines 1, 3–5 and 7), and finally returns the better solution found.

Time complexity analysis. Algorithm compADS$^+$ runs in $O(|V| + |E| + |V'|\log|V'| + |E'|\log^2|V'|)$ time, in which $|E'|$ and $|V'|$ are the numbers of edges and nodes in the converted graph $\vec{H}$, respectively.

Observe the following. (1) Given an aggregate graph $\widehat{H}(V, E, W_a)$, it takes procedure convertAG $O(|V|+|E|)$ time to produce its converted graph $\vec{H}(V', E', p, w)$, as finding all connected components can be done in linear time [8] (line 1). (2) Procedure strongMerging runs in $O(|E'|\log^2|V'|)$ time to generate the merged converted graph $\vec{H}'$ of $\vec{H}$ (line 2). (3) An MST can be computed in $O(|E'|\log|V'|)$ time [8] (lines 3, 6). (4) Strong pruning can be done in $O(|V'|)$ time [23] (line 4). Finally, (5) bounded probing takes $O(r^2|E'| + r|V'|\log|V'|)$ time (line 5). Note that here $\vec{H}$ is typically much smaller than $\mathbb{G}$, $\vec{H}'$ is smaller than $\vec{H}$, and r is a small constant, *e.g.*, 3 or 4.

Remarks. Different from our earlier work compADS in [26], here compADS$^+$ is equipped with enhanced strong merging.

4.5 FIDES$^+$: The Complete Solution

We finally present the complete statistics-driven approach to FInding DEnse Subgraphs in temporal graphs, referred to as FIDES$^+$, which combines algorithm compADS$^+$ above and the algorithms of identifying time intervals in Section 3.

```
Input: Aggregate graph $\widehat{H}(V, E, W_a)$.
Output: Subgraph of $\widehat{H}$ with a large cohesive density.
1. $\vec{H}(V', E', p, w) := \mathsf{convertAG}(\widehat{H})$;
2. $\vec{H}' := \mathsf{strongMerging}(\vec{H})$;
3. $T$ := a minimum spanning tree of $\vec{H}'$;
4. $ST := \mathsf{strongPruning}\ (T)$;
5. $ST' := \mathsf{boundedProbing}\ (\vec{H}', ST)$;
6. $ST''$ := a minimum spanning subtree of $\vec{H}$ using $ST'$;
   /*$ST''$ and $ST'$ have the same set of converted graph nodes*/
7. return the subgraph of $\widehat{H}$ corresponding to $ST''$.
```

Fig. 12. Algorithm compADS$^+$

Algorithm FIDES$^+$ takes as input a temporal graph $\mathbb{G}(V, E, W)$ and positive integers k and δ, and outputs the dense subgraph of $\mathbb{G}$ with the largest possible cohesive density. It first computes k time intervals using algorithms maxTInterval and minTInterval. Among these k time intervals, it finds and returns the subgraph of $\mathbb{G}$ with the largest possible cohesive density, using algorithm compADS$^+$.

Time complexity analysis. By the analyses of algorithms maxTInterval, minTInterval and compADS$^+$, it is easy to know that given a temporal graph $\mathbb{G}(V, E, W)$ and a positive integer k, algorithm FIDES$^+$ runs in $O((T + h^2)|E| + k(|V| + |E| + |V'|\log|V'| + |E'|\log^2|V'| + |E|\log T))$ time.

Space complexity analysis. The space complexity of algorithm FIDES$^+$ is $O(2T|E|)$: (1) the storage of the temporal graph costs $O(T|E|)$ space, (2) we compute $W_\Sigma(e, t)$ for each edge e and timestamp t, which costs another $O(T|E|)$ space, and (3) each step of compADS$^+$ is basically based on the converted graph $\vec{H}$, with the space complexity being the size of $\vec{H}$, *i.e.*, $O(|V'| + |E'|)$.

Note that here (1) h is the number of local maxima or minima, and (2) $|E'|$ and $|V'|$ are the largest numbers of edges and nodes in all converted graphs $\vec{H}$, which are typically much smaller than $|E|$ and $|V|$, respectively.

Remarks. (1) For procedure strongMerging, we also develop a new design that, given a converted graph $\vec{H}(V', E')$, improves its time complexity from $O(|E'|^2)$ [26] to $O(|E'|\log^2|V'|)$. (2) Different from our earlier work FIDES [26], FIDES$^+$ is equipped with the enhanced methods for finding the top–k time intervals (Section 3) and for computing dense subgraphs from aggregate graphs (Section 4).

5 Experimental Study

Using both real-life and synthetic data, we conduct an extensive experimental study of our efficient statistics-driven approach FIDES$^+$ to finding dense subgraphs in large temporal networks, compared with the state of the art method MEDEN [6] and our earlier work FIDES [26].

5.1 Experimental Settings

We first introduce the settings of our experimental study.

Datasets. We chose three datasets to test our approach.

(1) BJData is a real-life dataset that records the dynamic traffic condition of the road network in Beijing. Its road traffic conditions (+2: congested, +1: slow, and −1: fast)

were collected by taxies with GPS sensors, and were updated every 5 minutes. Here we consider day level data with 289 snapshots in total. Hence, BJDATA is very large, and has 23,724,877 nodes and 31,280,782 edges.

(2) SYNDATA is produced by the synthetic data generator developed in [6]. The generator first produces a temporal graph with n nodes, m edges and T snapshots using the random graph model, where all edge weights are first set to -1. It then activates a seed edge at random whose weight is set to +1. After this, its neighboring edges and its copy in the next snapshot are activated based on probabilities np_r and tp_r, respectively. All activated edges will perform the same process, with decayed np_r and tp_r. The process is repeated until the graph reaches a fixed activation density ad_r, the percentage of activated edges in all snapshots. Rates np_r, tp_r and ad_r are fixed to 0.3, 0.9 and 0.3 by default, and the number m of edges is fixed to $2n$.

(3) BENCHDATA contains four groups of 114 benchmark small graphs, *i.e.*, aggregate graphs, whose optimal dense subgraphs are known in advance [25].

Algorithms and implementation. Algorithms were all implemented with Java, including the state of the art algorithm MEDEN and the synthetic data generator [6]. Parameter δ for computing local maxima/minima was selected in a way such that local maxima/minima caused by slight fluctuations in the cohesive density curve were filtered, which was fixed to 4. Parameter r in both FIDES$^+$ and FIDES was fixed to 4 to reach stable results. Both δ and r are small constants for a balance between effectiveness and efficiency.

All experiments were run on a PC with 2 Intel Xeon E5–2630 2.4GHz CPUs and 64 GB of memory. When quantity measures are evaluated, the test was repeated over 5 times and the average is reported here.

5.2 Experimental Results

We next present our findings.

Exp-1. Verification of the evolving convergence phenomenon. In the first set of tests, we show the rationale of the evolving convergence phenomenon, which justifies the way how we identify the top–k time intervals.

Given a temporal graph $\mathbb{G}(V, E, W)$, we define a metric $p_{EC} = \sum_{t=2}^{T} \max\{|E^{\geq}(t)|, |E^{\leq}(t)|\}/(|E|(T-1))$ (*i.e.*, the proportion of edges that satisfy the evolving convergence phenomenon) to measure to what degree temporal graph $\mathbb{G}$ obeys the phenomenon, in which $|E^{\geq}(t)|$ and $|E^{\leq}(t)|$ denote the numbers of edges $e \in E$ with $W^t(e) \geq W^{t-1}(e)$ and $W^t(e) \leq W^{t-1}(e)$, respectively.

The p_{EC} are 96% on BJDATA and 90% on average on all tested SYNDATA, respectively, which justifies our observation of the evolving convergence phenomenon.

Exp-2. Algorithms compADS$^+$ **vs.** topDown **and** compADS. In the second set of tests, we compare the effectiveness and efficiency of algorithm compADS$^+$ with topDown and compADS, which compute the dense subgraphs on aggregate graphs given time intervals, and are called by FIDES$^+$, MEDEN and FIDES, respectively.

Exp-2.1. To evaluate the impacts of the number T_{ti} of snapshots in the time intervals of aggregate graphs, we varied T_{ti} from 50 to 289 for BJDATA and from 200 to 2,000 for SYNDATA, respectively. We used the entire BJDATA, and fixed SYNDATA with n = 100,000, T = 2,000 and $ad_r = 0.3$. For fairness, we computed the average results of 20 aggregate graphs with distinct time intervals for each T_{ti}, except the largest T_{ti} for BJDATA, shown in Figs. 13(a)–13(d).

When varying T_{ti}, the cohesive density scores of the subgraphs found increase with the increment of T_{ti}, as the data has temporal contiguity of positive weight edges. Further, the dense subgraphs found by compADS$^+$ are consistently better than those by topDown and compADS, *i.e.*, (+0.37%, +0.20%) and (+0.04%, +0.16%) on average better on (BJDATA, SYNDATA), respectively.

The running time of all algorithms is insensitive to the number T_{ti}, as the aggregate graphs are basically the same, in terms of both their sizes and structures. However, algorithms compADS$^+$ is much more efficient than topDown, and is (76, 26) and (1.17, 1.11) times faster than topDown and compADS on (BJDATA, SYNDATA), respectively. Recall that converted graphs and the strong merging technique reduce the sizes of aggregate graphs, which speeds up the computation. Further compADS$^+$ has a time complexity better than compADS, and is faster than compADS.

Exp-2.2. To evaluate the impacts of the graph size n, we varied n from 50,000 to 400,000 on SYNDATA, while fixed $T = 2{,}000$ and $ad_r = 0.3$. For fairness, we used the average result of 100 aggregate graphs by randomly generating 100 time intervals for each graph size. The results are reported in Figs. 13(e) & 13(f). We did not report topDown on graphs with $n \geq$ 150,000, as it ran out of memory.

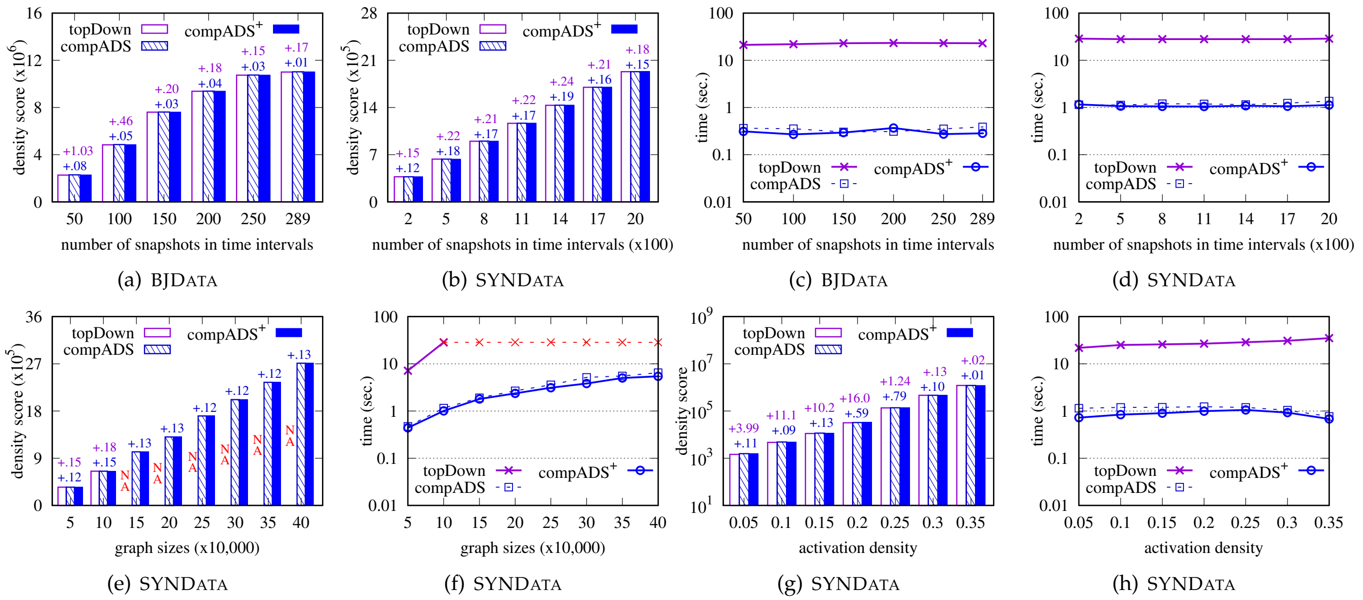
When varying n, the cohesive density scores of the subgraphs found by all algorithms obviously increase with the increment of n, and compADS$^+$ is consistently better than topDown (+0.17% on average) and compADS (+0.13% on average), respectively.

When varying n, the running time of all algorithms increases with the increment of n. Algorithm compADS$^+$ is much faster than topDown, and is 22 and 1.15 times faster than topDown and compADS, respectively.

Exp-2.3. To evaluate the impacts of the activation density ad_r, we varied ad_r from 0.05 to 0.35 on SYNDATA, while fixed n = 100,000 and T = 2,000. Due to the way that the synthetic generator works, it is already relatively dense in terms of positive weight edges even when ad_r is 0.35. Note that ad_r was fixed to 0.1 in [6]. For fairness, we also used the average results for each ad_r, the same as *Exp-2.2*, which are reported in Figs. 13(g) & 13(h).

When varying ad_r, the cohesive density scores of the subgraphs found by all algorithms obviously increase with the increment of ad_r, and compADS$^+$ is consistently better than topDown (+6.10% on average) and compADS (+0.26% on average), respectively. Algorithm compADS$^+$ performs significantly better than topDown when $ad_r \leq 0.2$, which is due to our three well-tuned optimization techniques.

When varying ad_r, the running time of both compADS$^+$ and compADS first increases and then decreases, while the one of topDown increases. This is because (a) there are more positive weight edges for larger ad_r, and (b) procedure convertAG and the strong merging technique become more effective on reducing the graph sizes when there are more positive weight edges. Finally, compADS$^+$ is 33 and 1.28 times faster than topDown and compADS, respectively.



(a) BJDATA (b) SYNDATA (c) BJDATA (d) SYNDATA

(e) SYNDATA (f) SYNDATA (g) SYNDATA (h) SYNDATA

Fig. 13. compADS+ vs. topDown and compADS, where numbers represent the improvement (%) over topDown (top) and compADS (bottom).

Exp-3. Algorithms FIDES+ vs. MEDEN and FIDES. In the third set of tests, we compare the effectiveness and efficiency of our approach FIDES+ with the state of the art method MEDEN and FIDES. In addition to the three factors evaluated in *Exp-2*, we further test the impacts of the number k of time intervals used in both FIDES+ and FIDES. By default, k is set to 10. Due to the superiority of compADS+ over topDown as shown in *Exp-2*, FIDES+ may produce denser subgraphs than MEDEN, despite the limited number of verified time intervals.

Exp-3.1. To evaluate the impacts of the number T of snapshots of temporal graphs, we varied T from 50 to 289 for BJDATA and from 200 to 2,000 for SYNDATA, respectively. We fixed $k = 10$, and used the same setting as *Exp-2.1*. The results are reported in Figs. 14(a)–14(d).

When varying T, the cohesive density scores of the subgraphs found by all algorithms increase with the increment of T. Moreover, the dense subgraphs found by FIDES+ are consistently better than those by MEDEN and FIDES, *i.e.*, (+0.37%, +0.16%) and (+0.07%, +0.13%) on average better on (BJDATA, SYNDATA), respectively.

When varying T, the running time of all algorithms obviously increases with the increment of T. Moreover, FIDES+ is consistently much faster than MEDEN, and is (2,978, 1,861) and (2.39, 1.18) times on average faster than MEDEN and FIDES on (BJDATA, SYNDATA), respectively. Algorithm FIDES+ is much faster than FIDES on BJDATA when $T \leq 150$ since the number of returned time intervals is less than the one of the previous FIDES in [26].

Exp-3.2. To evaluate the impacts of the parameter k, we varied k from 2 to 22. We used the same setting as *Exp-2.1* for T, n and ad_r. Algorithm MEDEN uses k time intervals to estimate a lower bound for pruning (line 4, Fig. 2), and k has impacts on the running time, but not the quality of the dense subgraphs found. The results are reported in Figs. 14(e)–14(h). We simply plotted red markers $*$ in Fig. 14(h) when MEDEN could not finish the tests in 2 days.

When varying k, the dense subgraphs found by FIDES+ and FIDES are insensitive to k when k is no less than 2 and 10, respectively. The dense subgraphs found by FIDES+ are (+0.20%, +0.15%) and (+0.05%, +0.11%) better than those by MEDEN and FIDES on (BJDATA, SYNDATA) when $k \geq 10$.

When varying k, the running time of MEDEN decreases, while the one of FIDES+ and FIDES increases, with the increment of k. Algorithm MEDEN could not finish the test in 2 days on SYNDATA for $k = 2$ as it used a non-effective lower bound. Moreover, FIDES+ is consistently much faster than MEDEN. Indeed, FIDES+ is (15,699, 3,791) and (-0.02, 1.29) times faster than MEDEN and FIDES on (BJDATA, SYNDATA) on average, respectively. Algorithm FIDES+ takes more time for identifying top–k time intervals than FIDES, and is slower when k is small.

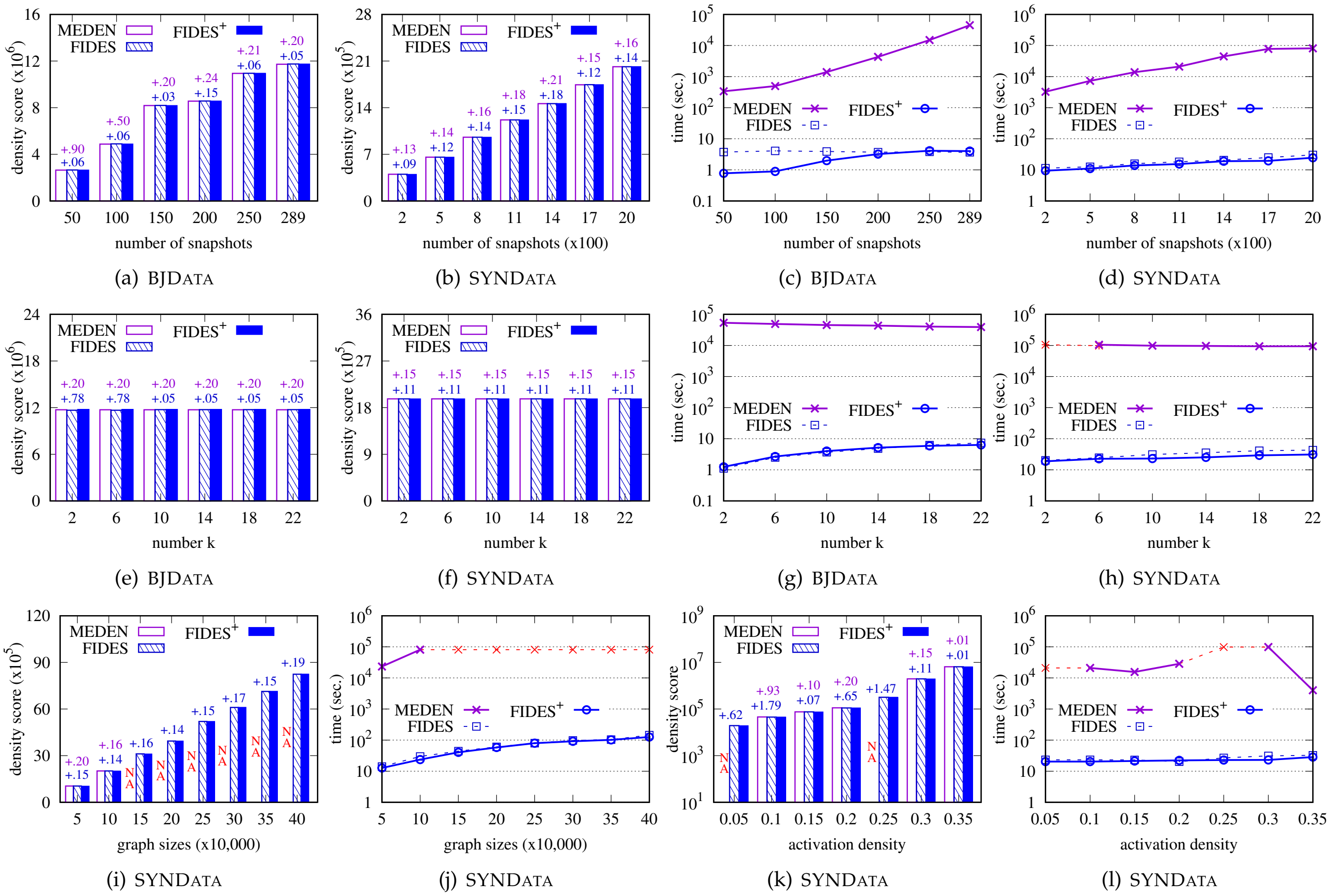
Exp-3.3. To evaluate the impacts of the graph size n, we used the same setting as *Exp-2.2* and fixed $k = 10$. We did not report MEDEN on graphs with size 150,000 or larger, as it ran out of memory, and could not finish the tests. The results are reported in Figs. 14(i) & 14(j).

When varying n, the cohesive density scores of subgraphs found by all algorithms increase with the increment of n, and FIDES+ is better than MEDEN (+0.18% on average) and FIDES (+0.16% on average) in our tests.

When varying n, the running time of all algorithms obviously increases with the increment of n. Moreover, FIDES+ is consistently much faster than MEDEN, and is 2,626 and 1.06 times faster than MEDEN and FIDES on average, in our tests. Moreover, FIDES+ could finish in 126 seconds when the graph size reaches 400,000, while it already took MEDEN 23,180 seconds on graphs with 50,000 nodes only.

Exp-3.4. To evaluate the impacts of the activation density ad_r, we used the same setting as *Exp-2.3* and fixed $k = 10$. The results are reported in Figs. 14(k) & 14(l). Note that MEDEN ran out of memory when ad_r was 0.05 or 0.25, as in these cases there were too many unpruned time intervals to verify, and too much space to store the aggregate graphs.

When varying ad_r, the cohesive density scores of subgraphs found by all algorithms increase with the increment of ad_r, and FIDES+ is consistently better than MEDEN (+0.28% on average) and FIDES (+0.67% on average) in our tests. Algorithm FIDES+ is more robust to ad_r than FIDES, which is worse than MEDEN when ad_r is 0.1 or 0.2.



(a) BJDATA (b) SYNDATA (c) BJDATA (d) SYNDATA

(e) BJDATA (f) SYNDATA (g) BJDATA (h) SYNDATA

(i) SYNDATA (j) SYNDATA (k) SYNDATA (l) SYNDATA

Fig. 14. FIDES$^+$ vs. MEDEN and FIDES, where numbers represent the improvement (%) over MEDEN (top) and FIDES (bottom).

When varying ad_r, the running time of MEDEN first increases and then decreases. This is due to the impacts of the pruning technique of MEDEN. Note that here $ad_r = 0.3$ is a turning point for MEDEN, as it happens that the estimated bounds of MEDEN are not very effective when $ad_r = 0.3$. The running time of FIDES$^+$ and FIDES is quite stable *w.r.t.* ad_r. Further, FIDES$^+$ is much faster than MEDEN, and is 1,486 and 1.13 times faster than MEDEN and FIDES on average, in our tests, respectively.

Exp-4. Performance of optimization techniques. In the fourth set of experiments, we test the performance of the three optimization techniques. We implement four alternatives to algorithm compADS$^+$ using different sets of optimization techniques. More specifically, algorithm SP only uses the strong pruning technique, SPBP uses strong pruning and bounded probing, SMSP uses strong merging and strong pruning, and, finally, OptAll uses all optimization techniques. Note that strong pruning is a basic technique and is used in all cases, and algorithm compADS$^+$ returns the better result of algorithms SPBP and OptAll. We tested the effectiveness and efficiency using the same settings as *Exp-2*. The results are reported in Fig. 15.

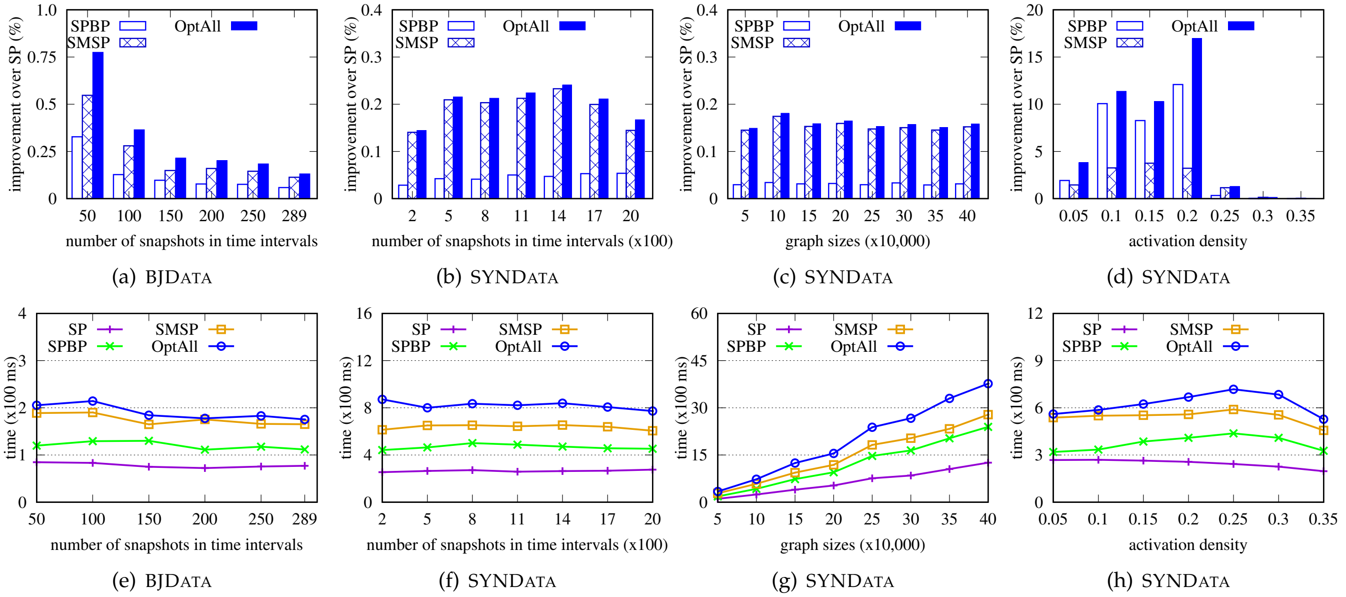
When varying number T_i of snapshots in the time intervals, graph size n and activation density ad_r, algorithm OptAll consistently performs the best in all our tests. The cohesive density scores of subgraphs found by (SPBP, SMSP, OptAll) are (+0.13%, +0.23%, +0.31%) and (+0.05%, +0.19%, +0.20%) better than the ones by SP on BJDATA and SYNDATA on average when varying T_i, and are (+0.03%, +0.15%, +0.16%) and (+4.67%, +1.86%, +6.27%) better than SP on SYNDATA on average when varying n and ad_r. On the other hand, algorithm OptAll costs the most time, followed by algorithms SMSP, SPBP and SP, respectively. The running time of (SP, SPBP, SMSP, OptAll) is (78ms, 120ms, 175ms, 190ms) and (265ms, 468ms, 637ms, 821ms) on BJDATA and SYNDATA on average when varying T_i, and is (649ms, 1,229ms, 1,495ms, 1,997ms) and (246ms, 375ms, 543ms, 624ms) on SYNDATA on average when varying n and ad_r, respectively. That is, the more optimization techniques are used, the better dense subgraphs are found, at an affordable extra time cost.

Exp-5. Closeness to optimality. (1) One may want to know the closeness to the optimal solutions of compADS$^+$ and FIDES$^+$. However, as shown by Proposition 1, computing the optimal dense subgraphs is infeasible even for aggregate graphs. Hence, we used the small graphs of BENCHDATA with known optimal dense subgraphs [25]. The cohesive density scores of subgraphs found by compADS$^+$ are (92%, 94%, 94%, 91%) of the optima of groups (K, P, C, D) on average, respectively. In contrast, topDown and compADS obtain an average performance of (83%, 87%, 87%, 84%) and (92%, 94%, 92%, 90%) on the four groups. (2) Roughly speaking, the solutions of FIDES$^+$ are around 85% of the optima on average, as FIDES$^+$ is slightly better than MEDEN, which achieves a performance of 85% of the optima [6].

Summary. From these tests we find the following.

(1) The evolving convergence phenomenon is quite common. Indeed, there are 96% and 90% of edges satisfying the phenomenon on BJDATA and SYNDATA, respectively.

(2) The quality of the dense subgraphs found by compADS$^+$



(a) BJDATA (b) SYNDATA (c) SYNDATA (d) SYNDATA

(e) BJDATA (f) SYNDATA (g) SYNDATA (h) SYNDATA

Fig. 15. Performance of optimization techniques

is consistently better than those by topDown and compADS, *i.e.*, (+0.28%, +0.17%) and (+0.04%, +0.13%) on (BJDATA, SYNDATA) on average, respectively. Further, compADS$^+$ is (76, 22) times faster than topDown on (BJDATA, SYNDATA), and is comparable to compADS in terms of efficiency. Finally, topDown already ran out of memory for graphs with 150,000 nodes and 2,000 snapshots.

(3) The quality of the dense subgraphs found by FIDES$^+$ is consistently better than those by MEDEN and FIDES, *i.e.*, (+0.20%, +0.15%) and (+0.05%, +0.11%) on (BJDATA, SYNDATA) on average, respectively, while FIDES is worse than MEDEN in some cases. Further, FIDES$^+$ is (2,978, 1,486) times faster than MEDEN on (BJDATA, SYNDATA) on average, and is comparable to FIDES in terms of efficiency. Finally, MEDEN already ran out of memory for graphs with 150,000 nodes and 2,000 snapshots.

(4) The three characteristics of time intervals (*i.e.*, Proposition 2, Fact 1 and Heuristic 2 in Section 3) together assure a pretty good estimation of the time intervals involved with dense subgraphs. Indeed, a small number of intervals, *e.g.*, 10, already suffice for FIDES$^+$ to find a good solution.

6 RELATED WORK

This study extends our earlier work [26] as follows. (1) We have improved our method by (a) enhancing the top–k time interval estimation algorithms with an *adaptive* smoothing threshold to identify more accurate time intervals (Section 3.2), (b) enhancing procedure strongMerging with type-II node merging and (c) improving the complexity of strongMerging with a better data structure (Section 4.3). (2) We have added (a) a comparison with our previous approach [26] and (b) new tests on our optimization techniques (Section 5). (3) We have also provided the details of procedure boundedProbing (Section 4.3) and proofs (Appendix).

Dense subgraphs in static networks. Dense subgraphs are a general concept and have been widely studied. The concrete semantics highly depend on the problems and applications, such as cohesive subgraphs like maximal cliques, n-clique, k-core and n-clan [32], [36], the prize collecting Steiner tree [13], [23], bump hunting on graphs [17], and densities defined in terms of the numbers or weights of edges and nodes [2], [3], [15]–[17], [24].

Our work adopts the strong pruning technique introduced in [23] for finding a better subgraph in aggregate graphs, by building the connection between finding the subgraph of an aggregate graph with the highest cohesive density and finding the maximum net worth subtree [23].

Dense subgraphs in dynamic networks. Dense subgraphs have also been recently investigated in temporal networks under various terms, such as anomalies [5], [7], heavy subgraphs [6], dense subgraphs [4], [9], [31] and network processes [30]. However, they typically refer to connected subgraphs with higher scores, defined in terms of average degree [31] or the weights of edges and nodes in a continuous time interval. Our work adopts the definition of dense subgraphs in [6], and is different from those in [4], [5], [7], [9], [30], [31]. Further, the study in [4], [9] focuses on dynamic graphs with node and/or edge updates, and, hence, is different from ours.

Close to our work is [6] that proposed and studied the FDS problem. We develop an efficient statistics-driven solution, totally different from the filter-and-verification method in [6]. Moreover, the connection between the FDS and NWM problems has never been employed in the algorithm of [6], not to mention the approximation hardness result of the FDS problem. Further, statistics-driven solutions using hidden data statistics also shed light on large graph processing.

Other studies in dynamic networks. Temporal network analysis has recently attracted more and more attentions [1], [20], [40], such as temporal shortest paths [14], [19], [37], temporal minimum spanning trees [22], incremental graph pattern matching [12], graph stream analysis [38] and continuous aggregate queries [29]. Different from these, we study dense temporal subgraphs in this work.

7 CONCLUSIONS

We have proposed FIDES$^+$, a highly efficient approach employing hidden data statistics to finding dense subgraphs in large temporal networks. First, we have employed the data characteristics to effectively identify k time intervals from a total of $T(T+1)/2$ ones, in which T is the number of snapshots and k is typically much smaller than T. Second, we have developed better algorithm heuristics to solve the problem. Finally, we have experimentally verified that FIDES$^+$ finds better dense subgraphs than the state of the art method MEDEN [6], and is much more scalable.

A couple of issues need further study. We are to apply our approach to temporal graphs with node/edge updates, and extend our techniques to find top–k dense subgraphs.

Appendix: Proofs

Proof of Proposition 2: Assume without loss of generality, the time interval $[i, j]$ has a dense subgraph. We only need to consider the case when there are no local maxima between i and j. Let i' be the largest point with $i' \leq i$ and j' be the least point with $j' \geq j$, if there exist, such that the cohesive density curve has a local maximum at points i' and j'. Otherwise, we simply let $i' = i$ or $j' = j$. Note that there must exist at least one of i' and j' in this case.

By the evolving convergence phenomenon and the definition of dense subgraphs, time interval $[i', j']$ has a subgraph whose cohesive density is no less than the dense subgraph of interval $[i, j]$. We thus have the conclusion. □

Proof of Theorem 3: We first present the definition of AFP-reduction and then show that there exists an AFP-reduction (h, g) from the NWM problem to the FDS problem. The conclusion follows since the NWM problem is NP-hard to approximate within any constant factor [13], [23].

(1) Let Π_1 and Π_2 be two maximization optimization problems. An AFP-reduction from Π_1 to Π_2 is a pair of PTIME functions (h, g) that satisfies the following:

(a) for any instance I_1 of Π_1, $I_2 = h(I_1)$ is an instance of Π_2 such that $\mathsf{opt}_2(I_2) \geq \mathsf{opt}_1(I_1)$, where $\mathsf{opt}_1(I_1)$ (respectively $\mathsf{opt}_2(I_2)$) is the value of an optimal solution to I_1 (respectively I_2), and

(b) for any feasible solution s_2 to I_2, $s_1 = g(s_2)$ is a feasible solution to I_1 such that $\mathsf{obj}_1(s_1) \geq \mathsf{obj}_2(s_2)$, where $\mathsf{obj}_1()$ (respectively $\mathsf{obj}_2()$) is a function measuring the value of a solution to I_1 (respectively I_2).

(2) We next construct algorithm h. Given an instance I_1 of NWM as its input, algorithm h outputs an instance I_2 of FDS on an aggregate graph. The instance I_1 consists of a graph $G(V, E)$ with a non-negative node weight $p(v)$ for each node $v \in V$ and a non-negative edge weight $w(e)$ for each edge $e \in E$. Algorithm h constructs the instance I_2 consisting of an aggregate graph $\widehat{G}(V \cup V', E \cup E', W_a)$ such that (a) for each node $v \in V$ of G, it adds a new node v' and a new edge $e_v = (v, v')$ to $\widehat{G}$ with $W_a(e_v) = p(v)$, and (b) for each edge $e \in E$ of G, it sets $W_a(e) = -w(e)$ in $\widehat{G}$. That is, aggregate graph $\widehat{G}$ has $2 \times |V|$ nodes and $|E| + |V|$ edges, and, hence, algorithm h runs in PTIME.

(3) We then construct algorithm g. Given a feasible solution s_2 to I_2, *i.e.*, a subgraph $\widehat{H}(V_s, E_s, W_a)$ of $\widehat{G}$, algorithm g outputs a solution s_1 of the NWM instance I_1. Algorithm g first produces a subgraph $H(V_s', E_s')$ of G from $\widehat{H}(V_s, E_s, W_a)$ by (a) removing all the nodes v' not in G and their incident edges (v, v'), (b) assigning all the remaining nodes with the same node weights as in G, and (c) setting $w(e) = -W_a(e)$ for each remaining edge e. Then it finds and returns a minimum spanning tree of H, which is a feasible solution to I_1. Given these, algorithm g is in PTIME as well.

(4) By the constructions of algorithms h and g above, it is easy to see that $\mathsf{opt}_2(I_2) = \mathsf{opt}_1(I_1)$ and $\mathsf{obj}_1(s_1) \geq \mathsf{obj}_2(s_2)$. Hence, (h, g) is indeed an AFP-reduction from the NWM problem to the FDS problem.

Putting these together, we have the conclusion. □

Proof of Proposition 4: The equivalence can be proved by showing that the dense subgraph in an aggregate graph can be used to construct the maximum net worth subtree in the corresponding converted graph, and vice versa.

Observe that the dense subgraph SG of an aggregate graph $\widehat{H}$ contains the entire $\textsc{cc}_i$ (line 2, Fig. 5) if any edge of $\textsc{cc}_i$ is included in SG, and these components are further linked by edges of $\mathsf{convertAG}(\widehat{H})$ (lines 6–7, Fig. 5). Hence, $\mathsf{convertAG}(SG)$ is a subtree of $\mathsf{convertAG}(\widehat{H})$ whose net worth equals to $\mathsf{cden}(SG)$. Moreover, any subtree ST of $\mathsf{convertAG}(\widehat{H})$ can construct a subgraph of $\widehat{H}$ whose cohesive density equals to $NW(ST)$.

Given the dense subgraph SG of an aggregate graph $\widehat{H}$, subtree $\mathsf{convertAG}(SG)$ must be the maximum net worth subtree of $\mathsf{convertAG}(\widehat{H})$. Otherwise, a subtree ST^* with a higher net worth exists and can construct subgraph SG^* of $\widehat{H}$ with a higher cohesive density. Similarly we can prove that given the maximum net worth subtree ST of $\mathsf{convertAG}(\widehat{H})$, subgraph SG corresponding to ST must be the dense subgraph of $\widehat{H}$. We thus have the conclusion. □

Proof of Proposition 5: Let T be the merged MST of two MSTs $T_1(V_1, E_1)$ and $T_2(V_2, E_2)$ in the process of strongMerging. We first show that it suffices to consider a T consisting of edges of T_1 and T_2 and those between T_1 and T_2 only. Without loss of generality, assume that T contains an edge $e = (u, v)$ with $u, v \in V_1$ and $e \notin E_1$. Adding e into T_1 forms a cycle $u/\cdots/v/u$ in T_1. Given the fact that T_1 is an MST, the weight of e must be the maximum among all edges in the cycle. And, hence, edge e of T can be replaced by an edge in the cycle without increasing the total weight of T. Similarly, there is no need to consider edges $e = (u, v)$ with $u, v \in V_2$ and $e \notin E_2$.

From above, it is easy to see that merging T_1 and T_2 is equivalent to maintaining an MST in a graph $T_1 \cup T_2$ after inserting the edges between V_1 and V_2. Using dynamic trees, it takes $O(\log(|V_1| + |V_2|))$ time to deal with an edge [34]. Hence, the extra cost is bounded by $O(|E'| \log |V'|)$. □

Proof of Corollary 6: Procedure strongPruning is essentially the same as the BEST-SUBTREE pruning procedure (Fig. 2-5) in [28], except that the latter recursively finds the subtree. Note that Theorem 2.8 in [28] proves that the returned subtree ST obtains the optimal net worth among all possible subtrees of T. We next show that procedure strongPruning runs in $O(|V_T|)$ time. It first traverses the tree from the root in a breath-first manner, and updates $nw(u)$ in the reverse traversal order. Finally, strongPruning extracts and returns the subtree rooted at the node with the maximun $nw(u)$. All these operations together can be done in $O(|V_T|)$ time. □

Acknowledgments

This work is supported in part by National Key R&D Program of China 2016YFB1000103, NSFC U1636210&61421003,

Beijing Advanced Innovation Center for Big Data and Brain Computing, and MSRA Collaborative Research Program.

References

[1] C. C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM CSUR*, 47(1):10:1–10:36, 2014.

[2] R. Andersen. A local algorithm for finding dense subgraphs. *ACM Transactions on Algorithms*, 6(4), 2010.

[3] O. D. Balalau, F. Bonchi, T. H. Chan, F. Gullo, and M. Sozio. Finding subgraphs with maximum total density and limited overlap. In *WSDM*, 2015.

[4] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC*, 2015.

[5] P. Bogdanov, C. Faloutsos, M. Mongiovì, E. E. Papalexakis, R. Ranca, and A. K. Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *SDM*, 2013.

[6] P. Bogdanov, M. Mongiovì, and A. K. Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, 2011.

[7] J. Chan, J. Bailey, C. Leckie, and M. Houle. ciForager: Incrementally discovering regions of correlated change in evolving graphs. *TKDD*, 6(3):11, 2012.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[9] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, 2015.

[10] W. Fan and C. Hu. Big graph analyses: From queries to dependencies and association rules. *Data Science and Engineering*, 2(1):36–55, 2017.

[11] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Graph homomorphism revisited for graph matching. *PVLDB*, 3(1), 2010.

[12] W. Fan, X. Wang, and Y. Wu. Incremental graph pattern matching. *TODS*, 38(3):18, 2013.

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

[14] L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

[15] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, 2012.

[16] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

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

[18] W. Han, Y. Miao, K. Li, M. Wu, F. Yang, L. Zhou, V. Prabhakaran, W. Chen, and E. Chen. Chronos: a graph engine for temporal graph analysis. In *Eurosys*, 2014.

[19] M. S. Hassan, W. G. Aref, and A. M. Aly. Graph indexing for shortest-path finding over dynamic sub-graphs. In *SIGMOD*, 2016.

[20] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97 – 125, 2012.

[21] H. Huang, J. Song, X. Lin, S. Ma, and J. Huai. TGraph: A temporal graph data management system. In *CIKM*, 2016.

[22] S. Huang, A. W. Fu, and R. Liu. Minimum spanning trees in temporal graphs. In *SIGMOD*, 2015.

[23] D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: theory and practice. In *SODA*, 2000.

[24] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, 2009.

[25] I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Math. Program.*, 105(2-3):427–449, 2006.

[26] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai. Fast computation of dense temporal subgraphs. In *ICDE*, 2017.

[27] S. Ma, J. Li, C. Hu, X. Lin, and J. Huai. Big graph search: challenges and techniques. *Frontiers of Computer Science*, 10(3):387–398, 2016.

[28] M. Minkoff. The prize collecting steiner tree problem. Master's thesis, MIT, 2000.

[29] J. Mondal and A. Deshpande. EAGr: supporting continuous egocentric aggregate queries over large dynamic graphs. In *SIGMOD*, 2014.

[30] M. Mongiovì, P. Bogdanov, and A. K. Singh. Mining evolving network processes. In *ICDM*, 2013.

[31] P. Rozenshtein, N. Tatti, and A. Gionis. Finding dynamic dense subgraphs. *ACM Trans. Knowl. Discov. Data*, 11(3):27:1–27:30, 2017.

[32] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *PVLDB*, 10(3):97–108, 2016.

[33] K. Semertzidis and E. Pitoura. Durable graph pattern queries on historical graphs. In *ICDE*, 2016.

[34] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *JCSS*, 26(3):362–391, 1983.

[35] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[36] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[37] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *PVLDB*, 7(9):721–732, 2014.

[38] W. Yu, C. C. Aggarwal, S. Ma, and H. Wang. On anomalous hotspot discovery in graph streams. In *ICDM*, 2013.

[39] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *ACM TIST*, 5(3):38:1–38:55, 2014.

[40] B. Zong, X. Xiao, Z. Li, Z. Wu, Z. Qian, X. Yan, A. K. Singh, and G. Jiang. Behavior query discovery in system-generated temporal graphs. *PVLDB*, 9(4):240–251, 2015.

Shuai Ma is a professor at the School of Computer Science and Engineering, Beihang University, China. He obtained his PhD degrees from University of Edinburgh in 2010, and from Peking University in 2004, respectively. He was a postdoctoral research fellow in the database group, University of Edinburgh, a summer intern at Bell labs, Murray Hill, USA and a visiting researcher of MSRA. He is a recipient of the best paper award for VLDB 2010 and the best challenge paper award for WISE 2013. He is an Associate Editor of VLDB Journal since 2017. His current research interests include database theory and systems, and big data.

Renjun Hu is a PhD student at the School of Computer Science and Engineering, Beihang University, supervised by Prof. Shuai Ma. He obtained his BS degree in computer science and technology from Beihang University in 2014. He was a visiting student at Rutgers, The State University of New Jersey. His research focuses on mining massive social networks and graph data.

Luoshu Wang is currently a software engineer at Google Inc., Mountain View, CA. She obtained her M.S. degree in Software Engineering from Carnegie Mellon University in 2016 and B.S. degree in Computer Science from Beihang University in 2015. Her current research interests include content ranking and recommendation, and data mining.

Xuelian Lin is an assistant professor at the School of Computer Science and Engineering, Beihang University. He received his PhD and Master degrees from Beihang University in 2013 and 2002, respectively. He was a visiting researcher of Baidu. His current research interests include large scale data management systems, data intensive computing, mobile computing and time series analysis.

Jinpeng Huai is a professor at the School of Computer Science and Engineering, Beihang University, China. He received his Ph.D. degree in computer science from Beihang University, China, in 1993. Prof. Huai is an academician of Chinese Academy of Sciences and the vice honorary chairman of China Computer Federation (CCF). His research interests include big data computing, distributed systems, virtual computing, service-oriented computing, trustworthiness and security.