Attribute-Driven Backbone Discovery

Sheng Guan[*] Hanchao Ma[*] Yinghui Wu[*†]

Washington State University[*] Pacific Northwest National Laboratory[†]
{sheng.guan,hanchao.ma,yinghui.wu}@wsu.edu

ABSTRACT

Backbones refer to critical tree structures that span a set of nodes of interests in networks. This paper introduces a novel class of *attributed backbones* and detection algorithms in richly attributed networks. Unlike conventional models, attributed backbones capture dynamics in edge cost model: it specifies *affinitive attributes* for each edge, and the cost of each edge is dynamically determined by the selection of its associated affinitive attributes and the closeness of their values at its end nodes. The backbone discovery is to compute an attributed backbone that covers interested nodes with smallest connection cost dynamically determined by selected affinitive attributes. While this problem is hard to approximate, we develop feasible algorithms within practical reach for large attributed networks. (1) We show that this problem is *fixed-parameter approximable* parameterized by the number of affinitive attributes, by providing a Lagrangean-preserving 2-approximation. (2) When the attribute number is large and specifying closeness function is difficult, we provide a fast heuristic, which learns an edge-generative model, and applies the model to infer best backbones, without the need of specifying closeness functions. Using real-world networks, we verify the effectiveness and efficiency of our algorithms and show their applications in collaboration recommendation.

CCS CONCEPTS

• **Information systems** → *Network data models*; *Data mining*; • **Networks** → *Social media networks.*

KEYWORDS

Attribute-driven;backbones;attributed network

ACM Reference Format:
Sheng Guan, Hanchao Ma, and Yinghui Wu. 2019. Attribute-Driven Backbone Discovery. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330934

1 INTRODUCTION

Backbone structures [5, 25] are well established to understand the connectivity and relationships among critical entities in networks. Consider a network G and a set V_I of nodes of interests designated by users. One often wants to understand the relationships among V_I by finding a *backbone* T, *i.e.,* a subtree of G that spans over V_I with a minimized connection cost. Backbone detection has been applied for Pathway identification in biology study [2], information propagation [21], and network flow optimization[20].

Emerging needs in understanding richly attributed networks require the detection of new type of backbones. In these backbones, each edge and its semantic is dynamically characterized by *specific attributes and values* of its end nodes. Moreover, these attributes and their values also provide intuitive interpretation for the backbone structures. Consider the following example.

Example 1: Fig. 1 illustrates a fraction of an attributed geo-social network G. Each node v in G is a landmark with properties such as type or location, as well as attributes from statistics of its past visitors, such as major gender (*e.g.,* 'female'), tour_type (*e.g.,* 'family'), or type of favored attractions attr_type (*e.g.,* 'museum') when stayed at v. Such information can be extracted from photos, check-in or review data [4]. The edges denote routes among the landmarks.

To identify potential tours in G from scratch without knowing specific destination, a user may want to start with a "backbone" (a tree) with a root (*e.g.,* v_1 (hotel)) that spans a set of point of interests (POIs) to further inspect specific trips. Moreover, she may want to pose specific constraints as"tour_type" or "attr_type" to enforce that each route connects landmarks favored by similar type of tours (*e.g.,*'family') or have similar type (*e.g.,*'museum'), respectively. As such, the significance of each route in a tour varies due to different focus of preference reflected by specific attributes and their values.
(1) When a user specifies attribute "location", the connection cost of routes between two landmarks should be measured by geographical distance. For example, a route from restaurant v_4 to attraction v_5 (from coordinates $(150, 200)$ to $(150, 175)$) has smaller cost under "location". The route (v_5, v_{12}) is relatively less favorable due to longer distance ($(150, 175)$ to $(180, 70)$).
(2) Another user may prefer tours that connect attractions featuring similar "family tours" (tour_type). The same route (v_4, v_5) under this specification becomes less desirable compared with route (v_5, v_{12}) that connects two attractions both featuring family tours. Indeed, v_4 and v_5 are not "close" as v_4 is more favored by "couple".

A recommended tour that carries attributes on its routes indicate the rationale of the recommendation. For the first user, a trip plan (colored in blue) suggests that "*Park v_2, Museum v_3 and restaurant v_4 are closer as they are favored by the same gender group "female", while*

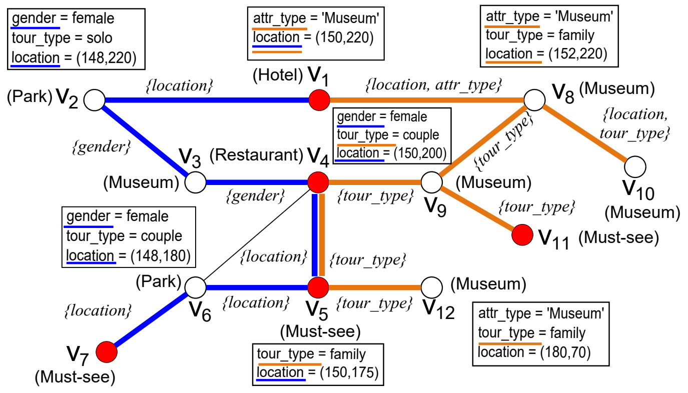


Figure 1: Suggesting tours with geo-social attributes.

the rest suggest closer trips. The second plan (colored in orange) are more favorable to family tours. Moreover, it is desirable to suggest trips along with the geo-social attributes that determine their cost, when users are not familiar with underlying data. □

The above scenario is a departure from conventional trip recommendation that assumes a fixed cost model and known destinations. The backbone structures in the example, although desirable, cannot be characterized by conventional characterization such as Steiner trees [6]. Indeed, these backbones are *jointly characterized* by interesting nodes and their relationships as well as the connectivity cost determined by specific node attributes of interests.

We call such backbone structures *attribute-driven backbones*. Attribute-driven backbones can be characterized by a weighted tree $T = (V_T, E_T)$ that connects a set of attributed nodes V_T, where each edge e=(v, v') in T is associated with a set of *affinitive attributes* A, such that both v and v' carry non-empty values for each attribute in A. The cost of an edge e in E_T should be dynamically determined by the selection of its affinitive attributes and their values.

We study a new problem called *attribute-driven backbone discovery* (ABD), characterized as follows.

- **Input**: Attributed graph G, a set of interested nodes V_I in G, an interestingness function I, and an edge cost model C that assigns a weight to an edge given its affinitive attributes;
- **Output**: an attributed backbone T that connects the nodes in V_I with maximized interestingness in terms of I, and minimized edge cost in terms of C.

Applications. Attribute-driven backbones readily provide enriched models for cascade structures enhanced by affinity attributes that may "explain" the connectivity between entities. (1) Attribute-driven backbones in geo-social networks can be used to further suggest personalized trips (paths) in trip recommendations [4]. (2) Signaling cascades such as pathways among proteins [2] can be modeled as attribute-driven backbones starting at receptor proteins and transmitted through protein interactions to effector proteins. The affinity attributes indicate interaction patterns between proteins with high confidence. (3) Adding affinity attributes to backbones in academic networks provide more expressive and interpretable collaboration patterns [24] among scholars, where affinity attributes such as "topics" or "affiliation" indicate potential collaboration that benefits from similar research areas or institutes. Attribute-driven backbones also suggest potential community in sparse networks, where density-based detection may become overkill [6], as verified by our experimental study (Section 5).

Although desirable, computing ABD is nontrivial. Instead of using predefined functions, the edge cost C in ABD (thus the quality of backbones) is dynamically determined by the selection of affinity attributes. Conventional community detection or Steiner-tree based algorithms [6] that rely on static cost model and labeled graphs can not be directly applied.

Contributions. We study feasible models and algorithms to compute *attribute-driven backbones* in large attributed networks.

(1) We formulate attribute-driven backbones to enrich the connectivity model among the nodes of interests (Section 2). We develop a bi-criteria cost model to characterize good backbones. Unlike conventional models, the cost model incorporates both node interestingness and edge cost, and is dynamically determined by the specification of affinity attributes and the topology of the backbone.

(2) Based on the cost model, we formulate the problem of *attributed backbone discovery* problem (ABD) (Section 2.2). This problem is NP-complete, and is also hard to approximate (APX-hard). This result is robust for polynomial-time computable cost models.

(3) Despite the hardness, we show that ABD is within reach by feasible approximation algorithms. We show that ABD is *fixed-parameter approximable*, under a practical case that the number of node attributes is small. We develop a *Lagrangian preserving* 2*-approximation*. This algorithm specializes a general Goemans-Williamson optimization scheme of cut problems for ABD. To reduce the cost, we devise a dynamically maintain *frontier* structure that prioritizes and explores promising edges in the neighbors of interested nodes in a bi-directional manner (Section 3).

(4) When the attribute number is large, we develop a faster heuristic to compute backbones and relevant attributes. The algorithm performs a *once-for-all* Expectation Maximization (EM)-based learning process to identify a generative model for given attributes, and grows backbones by iteratively selecting attributes that can minimize an expected edge cost for the backbone, for any given interested nodes (Section 4).

(5) Using real-world graphs, we experimentally verify the efficiency and effectiveness of our approximation and heuristic algorithms. We show that our algorithms can efficiently identify meaningful structures along with relevant attributes that cannot be expressed by existing backbone models. They also benefit the discovery of potential community structures in sparse and richly attributed networks (Section 5).

Related Work. We categorize the related work as follows.

Attributed community search. Clustering-based community detection has been studied for attributed networks [3, 10]. I-Louvain method [8] exploits (categorical and numerical) attributes and topological information simultaneously. Index structures are developed [16] to further reduce detection cost. ACQ with input k [12] is a subgraph that satisfies both structure cohesiveness (k core) and keyword cohesiveness. CESNA [30] models the dependence and interaction between the network topology and the node attributes and identify overlapping communities in networks. While prior work identifies communities (node sets) such that all the nodes in a community share same set of attributes, we compute backbones that allow each edge to have its own affinitive attributes. We consider backbones with cost that are dynamically determined by specific selection of affinitive attributes. Prior work typically assumes predefined quality measures that are independent with attributes. We provide approximation algorithms with provable guarantees on the attribute-driven backbones. On the other hand, the backbones either reports critical structures in sparse networks where dense counterparts may not exist, or benefit the detection of dense communities [16].

Connection subgraph detection. Connection subgraph detection [11] aims to compute connected (dense) subgraphs that cover source query nodes and incurs a cost under a budget. For example, [11] computes a subgraph that connects two input terminal nodes s and t with at most b nodes, which maximizes the total network flow from s to t. Center-piece subgraph discovery [26] extends the problem to computing subgraphs that cover a set of nodes. Detecting dense subgraphs in dual networks [29] aims to find a subset of nodes S, where two graphs G_a (unweighted) and G_b (weighted) represent physical and conceptual world separately but with complementary information, such that the induced subgraph $G_a[S]$ is connected and the density of $G_b[S]$ is maximized. (k, r)-core in [31] represents a cohesive subgraph on social network considering both user engagement and similarity perspectives simultaneously, which contains a k-core and an induced clique based on similarity measurement. These work do not consider node attributes and assumes predefined edge weights. In contrast, the attribute-driven backbones explicitly carry affinitive attributes that suggests the possible "reason" of the connection between network entities. Moreover, it does not require user to provide rigid parameters for topological constraints such as (k, r)-core. Posing such constraints can be a daunting task for users and an overkill for detecting community in a sparse network. In addition, we develop approximation algorithms to discover backbone structures characterized by attributes.

Keyword search in graphs. Keyword search over a graph finds a substructure of the graph containing the nodes carrying some or all input keywords (see [28] for a survey). Common models include Steiner trees [17] or r-cliques with bounded diameter [9]. These methods typically compute a minimum weighted Steiner tree, which is a special case of ABD with single label, unit node weight and predefined edge weight. We provide approximation algorithms to compute more general ABDs, by generalizing Goemans-Williamson scheme [14] that computes prize collecting Steiner trees in weighted graphs to attributed networks.

2 ATTRIBUTE-DRIVEN BACKBONES

2.1 Graphs and Attributed Backbones

Graphs. We consider an attributed graph $G = (V, E, F_A)$ with a finite set of nodes V, and a finite set of edges $E \subseteq V \times V$. Each node $v \in V$ has a *node tuple* $F_A(v) = \{(A_1, a_1), \ldots, (A_n, a_n)\}$ defined on a set of node attributes $\mathcal{A}$, where a pair $(A_i, a_i) \in F_A(v)$ states that the attribute $v.A_i \in \mathcal{A}$ has a value $a_i \in \text{adom}(A_i)$. Here (a) $\mathcal{A}$ refers to a set of all the node attributes seen in G; and (b) $\text{adom}(A_i)$ is a finite *active domain* of attribute A_i in G, and contains all the values of $v.A_i$, where v ranges over all the nodes in V.

We do not assume that all the nodes in G have the same set of attributes. Specifically, we denote the set of all the attributes in $F_A(v)$ as $\mathcal{A}(v)$ ($\mathcal{A}(v) \subseteq \mathcal{A}$).

Attribute-driven Backbone. Given a graph $G = (V, E, F_A)$, an attribute-driven backbone T is a tree (V_T, E_T, F_T), where

- $E_T \subseteq E$, and $V_T \subseteq V$ are the nodes that are the end nodes of the edges in E_T, *i.e.,* T is an edge-induced subtree of G;
- Each edge e=(v, v') $\in E_T$ has a set of *affinitive* attributes $F_T(e) \subseteq \mathcal{A}(v) \cap \mathcal{A}(v')$.

Intuitively, affinity attributes $F_T(e)$ for an edge e indicates possible common attributes of its two end nodes that can determine the strength of e. A common example is that friendship is more likely to be formed between two social network users who share common interests (*e.g.,*'hobby', 'activity', 'fan').

In the following sections, we shall refer to attribute-driven backbones as "backbones" for simplicity.

Example 2: Two backbones T_1 and T_2 are illustrated in Fig. 1. For edge (v_5, v_6) in T_1, $F_{T_1}((v_5, v_6))$ = {location}, stating that v_5 and v_6 are geographically close to each other in T_1. Other possible affinitive attributes for $((v_5, v_6))$ can be {tour_type} or {tour_type, location}. Similarly, the affinitive attributes for edge (v_1, v_8) in T_2 contains location and attr_type, suggesting that they are close in terms of location and both host tourists with common interests. □

Cost model. Given a set of *interested nodes* $V_I \subseteq V$, one wants to to find backbones that can cover V_I with small edge cost. We identify a cost model for backbones.

Node interestingness. Given a node $v \in V_I$, we consider a function I that quantifies the interestingness $I(v) \in (0, 1]$ of v. In practice, I can be specified by relevant entity search in graphs [22]. For example, a user may specify a term "art meseum" and identify nodes of interests as the set $V_I = \{v_3, v_8, v_9, v_{10}, v_{12}\}$ with relevance scores as their interestingness (Fig. 1).

Edge cost. We consider a cost function $C: E \times \mathcal{A} \to \mathbb{R}^+$ that computes a non-negative connection cost for an edge $e = (v, v')$ in the context of its affinitive attributes $F_T(e)$. Given a backbone edge $e \in E_T$ and its affinitive attributes $F_T(e)$, we define $\mathbb{C}(e, F_T(e))$ in the form of

$$C(e, F_T(e)) = 1 - \sum_{A \in F_T(e)} \frac{\text{cl}(F_A(v), F_A(v'))}{|F_T(e)|}$$

where $\text{cl}(F_{A_i}(v), F_{A_i}(v'))$ quantifies the closeness between the values of $v.A$ and $v'.A$ given attributes A and A'. The closeness can be specified by established similarity measures [1], semantic closeness, or geographical distance. The function C can capture tuple similarity by aggregating attribute similarity independently.

Cost of Attribute-driven Backbone. We now introduce a cost measure for backbones. Given a graph $G = (V, E, F_A)$, a set of interested nodes $V_I \subseteq V$, measures I and edge cost model C, the *cost* of a backbone T=(V_T, E_T, F_T) in G, denoted by cost(T), is defined as

$$\text{cost}(T) = \sum_{e \in E_T} C(e, F_T(e)) + \sum_{v \in V_I \setminus V_T} I(v)$$

The cost model penalizes the total edge cost in T (first term), as well as the "lost" interestingness from the nodes in V_I that are not covered by V_T (second term). The smaller cost(T) is, the better T should be. An optimal case for a backbone T is a tree that contains only V_I and assigns a set of affinitive attributes to each edge that minimizes the total edge cost.

We will simply denote cost(T) as $C(E_T) + I(\overline{V_T})$, the sum of total edge cost $C(E_T)$ and penalty $I(\overline{V_T})$.

Example 3: Given interested nodes $V_I = \{v_1, v_4, v_5, v_7, v_{11}\}$ in Fig. 1, where the interestingness $I(v_7) = 0.5$, and $I(v_{11}) = 0.3$. The cost of T_1 can be computed as the sum of the following: (1) $C((v_1, v_2), \text{location})$ refers to the normalized Euclidean distance between coordinates $v_1.\text{location}$ and $v_2.\text{location}$, similarly for $C((v_i, v_{i+1}), \text{location})$ $(i \in [4, 6])$; (2) $C((v_i, v_{i+1}), \text{gender}) = 0$ $(i = 2, 3)$, and (3) the cost $I(\overline{V_{T_1}}) = I(\{v_{11}\}) = 0.3$. The cost of T_2 can be computed similarly. □

2.2 Backbone Discovery Problem

Problem statement. Given a graph G with node set V, node interestingness measure I, edge cost model C, and a set of interested nodes $V_I \subseteq V$ with a root node r, the problem of attribute-driven backbone detection problem (ABD) is to compute a valid attribute-driven backbone T with root r and has a minimum cost cost(T). More specifically, it solves:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} C(e, F_T(e)) x_e + \sum_{U \subseteq V} I(U) z_U \\
\text{subject to} \quad & \sum_{e \in \delta(S)} x_e + \sum_{U \supseteq S} z_U \geq 1 \qquad \forall S \subseteq V - \{r\}, \\
& x_e, z_U \geq 0 \qquad \forall e \in E, U \subseteq V.
\end{aligned}$$

where the Boolean variable x_e denotes whether edge $e \in E$ is in the backbone T; $I(U) = \sum_{v \subseteq U} I(v)$, z_U denotes whether a node set $U \subset V$ not containing root r is spanned by the backbone T, and $\delta(S)$ denotes the edges having exactly one endpoint in $S \subseteq V \setminus \{r\}$.

Theorem 1: *The decision version of* ABD *problem is* NP*-complete. It is* APX*-hard as an optimization problem.* □

Proof sketch: We show the following. (1) To see that ABD is in NP, we provide an NP problem that guesses a backbone with associated affinitive attributes, and verify its cost in polynomial time. The hardness can be verified by a reduction from minimum Steiner tree, a well-known NP-hard problem. The hardness of approximation follows from an approximation ratio preserving reduction from minimum Steiner tree which is also APX-hard [27]. □

The APX-hardness of ABD suggests that no polynomial-time algorithm can approximate the optimal backbone for arbitrary approximation ratio. Despite the hardness, we show that it is within reach in practice. We introduce an approximation algorithm, and a faster heuristic when C defined as a probabilistic model, in Sections 3 and 4, respectively.

3 APPROXIMATE OPTIMAL BACKBONES

Theorem 1 tells us that ABD is inherently hard to approximate; yet not all is lost. We study its *fixed-parameter approximability* [23]. Our first result considers a practical assumption that real-world networks pertain to small $|\mathcal{A}|$.

Fixed-parameter approximability. An instance of a parameterized optimization problem $\mathcal{I}$ is a triple (x, k, C), with an input of size $|x|$ structured by a parameter k, and an objective function C (to be minimized). We say the problem $\mathcal{I}$ is *fixed-parameter α-approximable* ($\alpha \in [0, 1]$), if the following condition holds:
(1) there exists a function f and an algorithm such that for any instance (x, k) of $\mathcal{I}$, it takes $O(f(k) \cdot |x|^c)$ time to find a solution y, where c is a constant; and
(2) $C(y) \leq (1 + \alpha) * C(y^*)$, where y^* is the optimal solution.

In practice, such an algorithm is feasible if k is small. In the case that $\mathcal{A}$ is small for G, we parameterize ABD with $l = \max_{v \in V} |\mathcal{A}(v)|$ $(l \geq 1)$. We have the following result.

Theorem 2: ABD *is fixed-parameter* 2*-approximable with fixed l; more specifically, there exists an algorithm that (1) guarantees an Lagrangean-preserving 2-approximation,* i.e., *it computes a backbone T such that*

$$C(E_T) + 2 \cdot I(\bar{V_T}) \leq 2 \cdot \text{cost}(T^*)$$

where T^* is the optimal attribute-driven backbone; and (2) runs in $O((2^l - 1)|E| \log |V|)$ time. □

As a constructive proof for Theorem 2, we next introduce the algorithm with the optimally guarantees.

3.1 Approximation Algorithm

Our first algorithm, denoted as approxABD, computes attributed backbones with Lagrangean preserving 2-approximation in polynomial time when $\mathcal{A}$ is small.

Overview. In a nutshell, the algorithm approxABD dynamically maintains a set of "active" clusters $\mathbb{P}$ of nodes (initialized as singletons from V_I), where each cluster induces a spanning tree that has potential to be a subtree of an optimal backbone. It iteratively grows each active cluster $P \in \mathbb{P}$ with a class of *affinitive edges* $E_A(P)$, where each affinitive edge $e \in E_A(P)$ (a) contains only one end node in P, and (b) carries a distinct set of affinitive attributes $F(e)$. In this process, approxABD verifies two predicates that can grow the backbone:

- **IsGrow**: Can any cluster be expanded with an affinitive edge (remain 'active')?
- **IsMerge**: Can any two clusters be merged?

The conditions are determined by a set of auxiliary values defined on C and $E_A(C)$. The growth and merge process terminates when there is no active cluster in $\mathbb{P}$, *i.e.,* a largest cluster is constructed and can no longer be expanded. The attributed backbone is then induced from the largest cluster.

We start with an outline of approxABD, followed by the detailed procedures and performance analysis.

Algorithm. The algorithm approxABD (illustrated in Fig. 2) dynamically maintains clusters $\mathbb{P}$, and for each cluster $P \in \mathbb{P}$, it bookkeeps (1) a status flag indicating whether it's "active" or "inactive"; (2) three types of values to control the updates of C, including a *surplus value* $s(P) = \sum_{v \in P \cap V_I} I(v)$, a *moat value* $y(P)$ (initialized as

```
Algorithm approxABD
Input: graph $G$=($V$, $E$, $F_A$), cost model $C$, interested nodes $V_I \subseteq V$.
Output: An attributed backbone $T$ = ($V_T$, $E_T$, $F_T$).
1.  Initializes $\mathbb{P}$, $\mathbb{Q}_E$, $\mathbb{Q}_P$; UpdateAE($\mathbb{P}$); backbone $T$:=$\emptyset$;
2.  while there is an active cluster in $\mathbb{P}$ do
3.      ($e_{\min}$, $P_v$, $P'_v$) := topEdge($\mathbb{P}$, $\mathbb{Q}_E$);
4.      if isGrow ($\mathbb{P}$) = true then /* grow the clusters in $\mathbb{P}$ */
5.          update $s(e_{\min})$;
6.          if isMerge($e_{\min}$, $P_v$, $P'_v$) = true then
    /* merge clusters and update affinitive edges for the new cluster */
7.              Merge($P_v$, $P'_v$, $\mathbb{P}$, $T$); UpdateAE($\mathbb{P}$);
    /* grow clusters with their best affinitive edges */
8.          else Grow($\mathbb{P}$, $\mathbb{Q}_E$);
    /* deactivate the clusters in $\mathbb{P}$ */
9.      else deactivate (topCluster($\mathbb{P}$, $\mathbb{Q}_P$));
10. refine ($T$); /* prune edges to ensure tree structure */
11. return $T$;
```

Figure 2: Algorithm approxABD

0), and for each affinitive edge $e \in E_A(P)$, a surplus value $s(e) = C(e, F(e))$. In addition, it maintains two priority queues of $\mathbb{P}$: $\mathbb{Q}_P$ that indexes $\mathbb{P}$ by an ascending order of $s(P)$, and $\mathbb{Q}_E$ that prioritizes $\mathbb{P}$ by ascending order of $\min_{e \in E_A(P)} s(e)$, *i.e.,* the smallest surplus value in the affinitive edge set $E_A(P)$.

Algorithm approxABD has the following three phases.

Initialization. approxABD initializes the active clusters $\mathbb{P}$ as follows: for each node of interests $v \in V_I$, there is a singleton $P_v \in \mathbb{P}$, where $s(P_v) = I(v)$ and $y(P) = 0$. It then invokes a procedure UpdateAE (line 1) to initialize the affinitive edges for each cluster $P_v \in \mathbb{P}$. Specifically, for each edge e=(v, v') adjacent to v, $E_A(P_v)$ contains a set of affinitive edges e, each carries a distinct set of affinitive attributes $F(e) \subseteq \mathcal{A}(v) \cap \mathcal{A}(v')$. Finally, it updates $s(e) = C(e, F(e))$. The queues $\mathbb{Q}_E$ and $\mathbb{Q}_P$ are initialized accordingly.

Clustering. It then iteratively performs the following (lines 2-9). It first fetches a triple $(e_{\min}, P_v, P'_v)$, where $e_{\min}$ refers to an affinitive edge (v, v') of P_v with the smallest surplus value $s(e)$, P_v and P'_v are two clusters connected by $e_{\min}$. It then verifies the condition isGrow holds, which indicates that the active clusters can be expanded to cover more interested nodes. Specifically, approxABD tracks, for each cluster P, a set of all the "childern" clusters $\{P_1, \ldots, P_m\}$ that were merged into P. The condition isGrow then tests whether for every $P \in \mathbb{P}$, $\Sigma_{i=1}^{n} y(P_i) < s(P)$, and performs the following:

(a) If all the clusters $\mathbb{P}$ can be expanded and contribute to the growth of backbone T, approxABD verifies two cases below that may lead to larger clusters:

- Clusters P_v and P'_v can be merged, verified by the condition isMerge (lines 6-7). More specifically, isMerge tests whether $s(e_{\min}) = 0$. If so, it invokes procedure Merge to construct a new cluster $P' = P_v \cup P'_v$, set $y(P') = 0$, deactivate P_v and P'_v, and add the affinitive edge $e_{\min}$ to T (not shown). It then constructs new affinitive edges for P' and update $\mathbb{Q}_E$ and $\mathbb{Q}_P$ (line 7) for the next round of clustering.
- Otherwise, it invokes a procedure Grow to expand the clusters (line 8). For each cluster $P \in \mathbb{P}$, it enlarges P with its affinitive edge having the smallest $s(e)$. It also enlarges $y(P)$ for all the clusters P at a same rate until isGrow is violated.

(b) If there exists a cluster that can not be enlarged and contribute to T, it deactivates the cluster. This is simply performed by selecting the top cluster in $\mathbb{Q}_P$ (line 9), which has an affinitive edge e with the smallest $s(e)$ among all affinitive edges of all the active clusters.

Refinement. Once there is no active cluster in $\mathbb{P}$, approxABD induces subgraph T that consists of affinitive edges added in the merge phase (lines 6-7). It then invokes procedure refine to prune T.

Example 4: Fig. 3 illustrates the moment that two clusters C_1 that contains v_1 and C_2 that contains v_2 (Fig. 1) are merged to a new cluster C_4 via an affinitive edge e, with affinitive attribute location. The condition isGrow first verifies that the current clusters C_1 and C_2 can still grow. It then identifies all the affinitive edges of C_4, including three affinitive edges e_1-e_3. As C_4 and C_3 (that contais v_3) "meets", isMerge is tested to verify whether $s(e_3)$ satifies the merge condition. If so, a new cluster that merges C_4 and C_3 is constructed, inducing T that contains edges e and e_3 carrying gender. □

3.2 Performance Analysis

The algorithm approxABD guarantees to terminate: there are at most $|V|$ clusters, and each round either deactivates a cluster that are no longer activated, or reduces a cluster by Merge. We next provide an analysis for the approximability and time cost of the algorithm approxABD.

Approximability. We next introduce the two predicates isGrow and isMerge, and clarify the approximation guarantee of approxABD. To clarify these, we consider a multigraph $\mathbb{G}$ obtained by running procedure UpdateAE over every node $v \in V$, which creates $2^{|\mathcal{A}(v) \cap \mathcal{A}(v')|}$ affinitive edges between two nodes v and v', each carries a subset of $\mathcal{A}(v) \cap \mathcal{A}(v')$. We observe that ABD is equivalent to computing a *prize collecting Steiner tree* problem over $\mathbb{G}$, which has the following dual of the Primal ABD over $\mathbb{G}$ (Section 2):

$$\begin{aligned} \text{maximize} \quad & \sum_{S \subseteq \mathbb{V}-\{r\}} y_S \\ \text{subject to} \quad & \sum_{S: e \in \delta(S)} y_S \leq c_e \quad \forall e \in \mathbb{E}, \\ & \sum_{S \subseteq U} y_S \leq \pi(U) \quad \forall U \subseteq \mathbb{V}, \\ & y_S \geq 0 \quad \forall S \subsetneq \mathbb{V}, \\ & G = (\mathbb{V}, \mathbb{E}) \end{aligned}$$

We take a closer look at the growth stage (lines 2-9). Observe that the moat value $y(P)$ can be viewed as the dual variables. A cluster P is "active" if and only if it has a positive surplus. In each round of the growth stage, approxABD uniformly raises moat values for each cluster P, while decreasing its corresponding surpluse $s(P)$ to pay for the increases, until either edge constraint becomes tight or the cluster constraint becomes tight as constraints of Dual.

These correspond to the two conditions below:

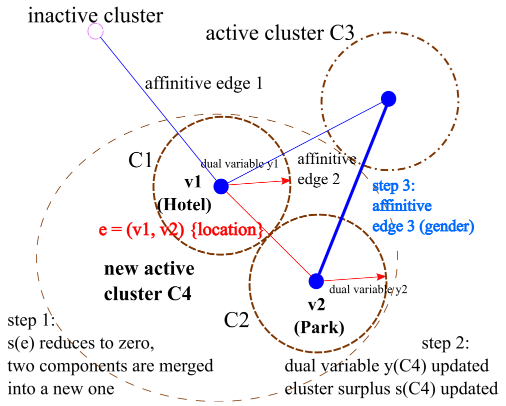


Figure 3: Dynamic clustering with affinitive edges

- the condition isGrow tests whether all the active components can keep growing or not. If $\sum_{S\subseteq U} y_S < \pi(U)$ ($\forall U \subseteq V$), isGrow(P) is true, and false otherwise;
- the condition isMerge tests whether any edge e ($\forall e \in E$) satisfies: $C(e, F(e)) = \sum_{S:e\in\delta(S)} y_S$.

Algorithm approxABD then simulates a GW scheme to compute a prize-collecting Steiner tree over $\mathbb{G}$. The approximation follows from an approximation preserving reduction from the instance of ABD problem to the prize-collecting Steinter tree problem over $\mathbb{G}$. It has been proven to have an Lagrangean-preserving 2-approximation [7, 13] on the Prize-collecting Steiner tree problem.

Time cost. It takes in total $O(|V|+|E|)$ time to initialize the computation. In the growth phase (lines 2-9), the total number of processed edges and connected clusters is at most $(2^l-1)|E|$; and each processing takes $O(\log |V|)$ time by accessing priority queues $\mathbb{Q}_E$ and $\mathbb{Q}_P$. Merge and refine take $O(|V|)$ time. The total time cost is thus $O((2^l-1)|E|\log |V|)$. Putting these together, Theorem 2 follows.

4 EM-BASED HEURISTIC

The fixed-parameter approximation is feasible when the parameter $l = \max |\mathcal{A}(v) \cap \mathcal{A}(v')|$ ($v, v' \in V$) is small, but can be expensive when $|\mathcal{A}|$ is large. Moreover, the closeness function may not be available for all the affinitive attributes. For example, users may only specify interested affinitive attributes but are not aware of a proper closeness measure for the attributes. We next introduce a faster heuristic algorithm that can quickly infer the backbones without requiring any closeness functions, and can quickly response to new queries with changed interested nodes.

Overview. As no closeness functions are available for $\mathcal{A}$, *i.e.,* no semantic closeness is known among attribute values, the main idea is to dynamically infer the edge cost C via an *edge generation model* $\mathbb{M}$, based on a practical assumption that the likelihood of the existence of an edge e= (v, v') is determined by the node attributes $\mathcal{A}(v)$ and $\mathcal{A}(v')$ and their values. Our second algorithm, denoted as fastABD, performs a once-for-all learning process to compute an edge generation model $\mathbb{M}$, characterized by an underlying stochastic network generative model [19]. Upon the specification of interested nodes V_I, fastABD dynamically infers a backbone T with affinitive attributes and corresponding cost $C(\cdot)$ from $\mathbb{M}$, such that the likelihood of existence of T is maximized.

```
Algorithm fastABD
Input: graph $G$, interested nodes $V_I$.
Output: an attribute-driven backbone $T$.
1.  if cost model is not specified then
2.    $\mathbb{M}$ := learnAE (G, $\mathcal{A}$);
3.  $T$ := inferAE (G, $\mathbb{M}$, $\mathcal{A}$, $V_I$);
4.  return $T$;
```

Figure 4: Algorithm fastABD

The algorithm fastABD has two major components: edge generation model learning, and attribute inference.

Learning edge generation model. We revise the multiplicative attribute graph model [19] to an edge generation model with affinitive attributes, which quantifies C as:

$$C(e, F_T(e), \mathbb{M}) = \prod_{i=1}^{|F_T(e)|} M_{A_i}[F_{A_i}(v)][F_{A_i}(v')]$$

Here $M_i \in \mathbb{M}$ is an affinity matrix associated to an attribute $A_i \in \mathcal{A}$. For each attribute $A \in \mathcal{A}$ and matrix M_A, the entry $M_A[j][k] \in [0, 1]$ refers to the probability that an edge (v, v') exists when $F_A(v)$ (resp. $F_A(v')$) takes the j-th (resp. k-th) value in adom(A).

Algorithm fastABD invokes a procedure learnAE to learn the affinity matrix $\mathbb{M}$. Specifically, the probability that E exists given node attribute values can be expressed as

$$\Pr(E|\mathcal{A}, \mathbb{M}) = \prod_{(v,v')\in E} C(e, \mathcal{A}, \mathbb{M}) \prod_{(v,v')\notin E} (1 - C(e, \mathcal{A}, \mathbb{M}))$$

The procedure learnAE aims to identify $\mathbb{M}$ by solving $\arg\max_{\mathbb{M}} \Pr(E|\mathcal{A}, \mathbb{M})$. To this end, learnAE applies a maximum likelihood estimation (EM) process to compute $\mathbb{M}$. To reduce the learning cost, we take a similar strategy as [19] which considers that each attribute follows Bernoulli distribution.

Dynamic inference. The procedure learnAE is a *once-for-all* process. Upon receiving nodes of interests V_I, fastABD invokes a second procedure inferAE, which computes backbones by incorporating the inference of affinitive attributes to the process of clustering.

Procedure inferAE follows a similar clustering process as in approxABD. It started by initializing $\mathbb{P}$, and iteratively groups clusters upon the verification of isGrow and isMerge. The major difference is that it applies a heuristic variant of the procedure UpdateAE, without enumerating affinitive edges for each cluster. More specifically, given a cluster P, it first identifies all the edges with one end node in P. For each edge e=(v, v'), it infers a set of affinitive attributes $F(e)$ directly, by solving

$$\underset{F(e)\subseteq\mathcal{A}(v)\cap\mathcal{A}(v')}{\arg\max} \; C(e, F(e), \mathbb{M})$$

Given the independent likelihood of edges, a heuristic is to sort and select top-k attributes $A_i \in F(e) \subseteq \mathcal{A}(v) \cap \mathcal{A}(v')$ with the largest values $M_{A_i}[F_{A_i}(v)][F_{A_i}(v')]$. This can be inferred at run time, whenever two clusters are merged.

The algorithm fastABD is illustrated in Fig. 4.

Performance analysis. It takes the procedure learnAE $O(|A|^2|E|)$ time to learn the affinity matrix from scratch. The time that infers the top-k best affinitive attributes for each cluster is in $O(|A| \log |k|)$ time, and in total $O(|A| \log |k||E|)$ for all the edges. As the affinitive attributes and the edge costs are dynamically inferred, it takes $O(|E| \log |V| + |V|)$ time for inferAE to simulate GW scheme that computes a prize-collecting Steiner tree. It thus takes in total $O(|A|^2|E| + |A| \log |k||E| + |E| \log |V| + |V|)$ time to compute backbones from scratch, and $O(|A| \log |k||E| + |E| \log |V| + |V|)$ time given the matrix $\mathbb{M}$ which is learned once for all.

5 EXPERIMENTAL RESULTS

Using real-world and synthetic dataset, we experimentally verify the efficiency and effectiveness of our backbone-based techniques.

Datasets. We used four real-world graphs, including (1) ***Yelp*** [1], a restaurant review graph that contains 244,012 edges and 25,881 nodes. Each node includes 10 attributes that are scores of the top-10 high frequency review keywords and the edge reflects that two restaurants are mutually to be one of the 10 nearest neighbors; (2) ***Weibo*** [2], a social network graph that contains 732,143 edges and 161,237 nodes. Each node represents a user that includes 6 attributes and the edge reflects "following" relationship; (3) ***Citation*** [3], an academic network that contains 2,044,459 edges and 1,544,605 nodes. Each node represents a paper that includes 6 attributes and the edge reflects citation relationship; and (4) ***YouTube*** [4], a video recommendation network graph that contains 2,509,862 edges and 129,907 nodes. Each node represents a video that includes 8 attributes and the edge connects one video and the other related videos recommended by YouTube.

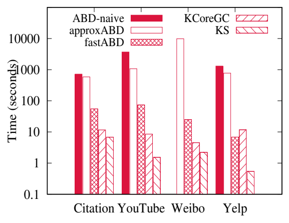
We also developed a synthetic graph generator based on GTgraph[5] controlled by node distribution models, graph size as edge number $|E|$, density d and attribute numbers $|\mathcal{A}|$. We applied Magfit [6] to generate attribute values.

For all datasets, we sample V_I by issuing entity search queries with terms of interests, such as conference topics for Citation, and the range of hitting volume for Youtube.

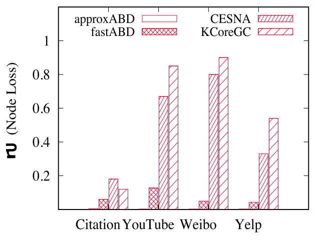
Algorithms. We implemented the following algorithms. (1) Algorithms approxABD, fastABD and fastABD + learning. Here the algorithm fastABD + learning performs a *once-for-all* EM-based learning process and learns the generation model; and fastABD refers to the algorithm that starts with a learned affinity matrix, and grows backbones by iteratively selecting attributes that can minimize an expected edge cost for the backbone. (2) ABD-naive, a variant of approxABD that transforms G to a multigraph $\mathbb{G}$ directly, and performs clustering scheme to compute prize-collecting Steiner tree. (3) CESNA [30], a state-of-the-art algorithm that returns communities as node sets with associated attributes. (4) KCoreGC [18], which detects connected subgraphs. (5) KS, a keyword search algorithm that implements state-of-the-art algorithm [15] to induce minimum Steiner trees over V_I as backbones.

For (1) and (2), the costs $c(.)$ in the experiment are calculated by taking dissimilarity measures, We define the common

[1]http://www.yelp.com/dataset_challenge
[2]https://aminer.org/data-sna
[3]https://aminer.org/citation
[4]http://netsg.cs.sfu.ca/youtubedata/
[5]http://www.cse.psu.edu/~kxm85/software/GTgraph/
[6]http://snap.stanford.edu/snap/index.html



(a) Efficiency



(b) Effectiveness: node loss

Figure 5: Performance on real-world graphs

attribute set between v_i and v_j as $\mathcal{A}_{ij} = \mathcal{A}(v_i) \cap \mathcal{A}(v_j)$. Then, $c(v_i, v_j) = \frac{\sum_{A \in \mathcal{A}_{ij}} d_{ij}(A)}{|\mathcal{A}_{ij}|}$, where $\mathcal{A}_{ij} \neq \emptyset$. $d_{ij}(A)$ stands for the distance between the common attribute values of A of node v_i and v_j. For (3), we retain the settings described in the paper of CESNA and consider all returned vertices as one big community. For (4), KCoreGC returns the coreness of each vertex in G. We sort the value of coreness from the largest to the smallest and only consider nodes with top-k coreness values to form an important community since the larger coreness value represents higher degrees and requires more attention. In the experiment, we set k as 20.

Metrics. For fastABD, we report the once-for-all learning time cost, and the inference time that constructs backbones from the model $\mathbb{M}$. We use two loss metrics applicable to all the algorithms for a fair comparison. Denote the subgraph returned by an algorithm as G_T, (1) the node loss ratio r_u = 1- $\frac{|V_T \cap V_I|}{|V_I|}$, *i.e.,* the fraction of V_I not covered by G_T, and (2) the interestingness loss ratio $r_I = \frac{\sum_{v \in V_I \setminus V_T} I(v)}{\sum_{v \in V_I} I(v)}$, where V_T is the node set of G_T. For all metrics, the smaller, the better the backbones are.

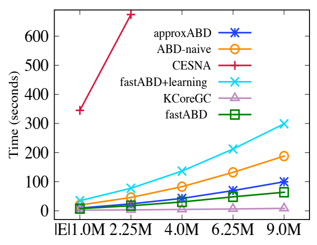
We ran all our experiments on a Linux machine powered by an Intel 2.4 GHz CPU with 128 GB of memory. We ran each test 5 times, and report the averaged results. Our source code and test cases are available online.[7]

We next present the details of our findings.

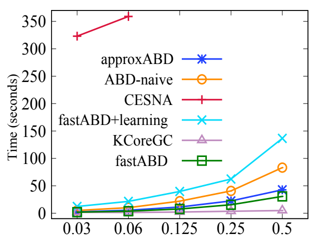
Exp-1: Efficiency. We first report the efficiency of approxABD, fastABD, and ABD-naive, compared with CESNA, KS, and KCoreGC on real-world datasets, as shown in Fig.5(a). We sample interested nodes for each dataset. CESNA does not run to completion after 3 hours for all datesets; same for ABD-naive over Weibo. fastABD is quite feasible over larger networks. For example, (1) fastABD takes up to 75 seconds over all the cases; (2) it improves approxABD and ABD-naive by 14 and 49 times respectively on Youtube; (3) it achieves comparable performance with KCoreGC and KS without attribute enumeration. On average, approxABD outperforms ABD-naive by 2.1 times due to more efficient clustering and affinity attribute selection process. KCoreGC and KS take less time, yet only return structures without attributes.

We next report the impact of several factors to the efficiency of the algorithms, over larger synthetic graphs under network models.

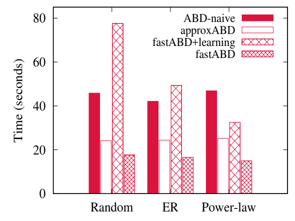
[7]https://github.com/wsu-db/Attribute-Driven-Backbone-



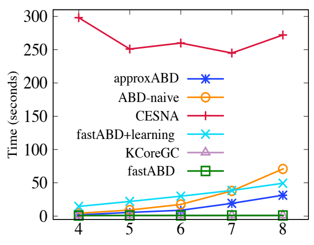
(a) Varying $|E|$



(b) Varying d

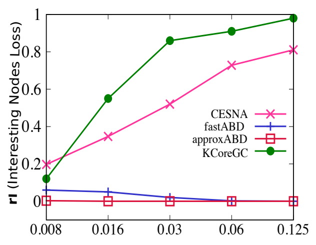


(c) Varying Network Model

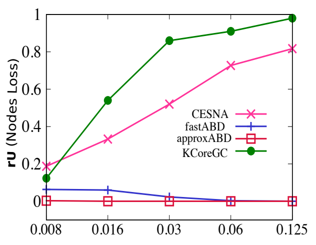


(d) Varying $|\mathcal{A}|$

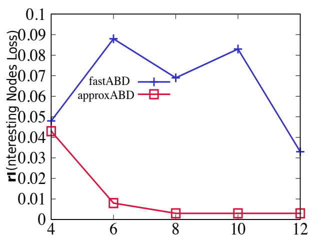
Figure 6: Efficiency: synthetic graphs



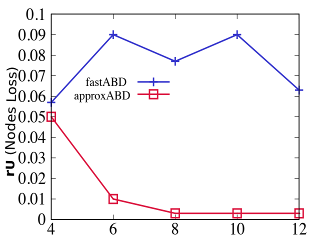
(a) Interestingness loss vs density



(b) Node loss vs. density



(c) Interestingness loss vs. $|\mathcal{A}|$



(d) Node loss vs. $|\mathcal{A}|$

Figure 7: Effectiveness: synthetic graphs

Varying graph size. Fixing attribute size $|\mathcal{A}| = 2$, density $d = 0.5$ ($d = \frac{2|E|}{|V||V-1|}$), interested node set size $|V_I|$=250, we varied $|E|$ from 1 million to 9 million under the Random, Erdos-Renyi and Power-Law distribution models. As shown in Fig. 6(a), algorithms approxABD and fastABD are quite feasible over large networks. Specifically, approxABD and fastABD improves ABD-naive by 1.95 and 2.72 times, respectively. The once-for-all learning cost of fastABD takes up to 300 seconds over a network with 9 million edges .

Varying d. Fixing $|V|$, $|\mathcal{A}|$, $|V_I|$ as 4000, 2 and 250, respectively, we varied the density of G from 0.03 to 0.5. As shown in Fig. 6(b), all algorithms take longer time over denser networks as more affinitive edges need to be inspected.

Varying Distribution and $|\mathcal{A}|$. We are also interested in the efficiency over different network models. As shown in Fig. 6(c), the performance of our algorithms over synthetic graphs are consistent with their real-world counterparts. The total cost of fastABD including learning cost is comparable with the cost of approxABD. Its inference cost only takes up to 39% of the cost of ABD-naive. Fig. 6(d) verifies that approxABD and ABD-naive take more time when $|\mathcal{A}|$ is larger; while for all cases, they take up to 70 seconds to identify backbones from networks with 8 attributes (where $|E|$, d and $|V_I|$ are set as 62500, 0.008 and 250, respectively).

Exp-2: Effectiveness. We report the node loss r_u of the algorithms in Fig. 5(b). To ensure that all the algorithms run to completion, we randomly sampled real-world graphs and created connected subgraphs with $|E|$=62,500. As KS always cover all the interested nodes, and ABD-naive has the same result as approxABD, their results are not shown. approxABD has the smallest coverage loss, while CESNA and KCoreGC easily lose more than 60% of the interested nodes over denser graphs. This is because both algorithms tend to return results that are "skewed" to high-degree nodes, leading to a larger missing rate for random interested nodes.

We next evaluate the impact of density d and $|\mathcal{A}|$. Fixing $|V|$=4000, $|\mathcal{A}|$=2, $|V_I|$=250, we varied d from 0.008 to 0.125. Fig. 7(a) and Fig. 7(b) show that it is more likely for approxABD and fastABD to find better backbones and reduce both node and interestingness loss in denser graphs. On the other hand, both CESNA and KCoreGC have higher chance to lose interested nodes, due to the distraction of high degree nodes and dense areas. This is consistent with the result in Fig. 5(b).

Fixing $|E|$= 62500, d= 0.008, $|V_I|$=300, we varied $|\mathcal{A}|$ from 4 to 12. Figs. 7(c) and 7(d) show that more attribute values help approxABD to find better backbones.The result constantly gets improved, ensured by its approximation scheme. fastABD is less sensitive to $\mathcal{A}$. We also found that it infers better backbones if more attributes are inferred (by setting larger k; not shown).

Exp-3: Case analysis. Fig. 8 and Fig. 9 illustrate real-world backbones from approxABD over Weibo and Citation, respectively.

(1) The social backbone on Weibo connects users having common topic on "Langlangconcert" (interested users; marked in orange) via close users determined by various affinitive attributes indicating same genders, ages, locations or common topics. We observed that such backbones can provide intuitive information that suggest how two entities are connected. For example, user 2 and user 3 both tweet "Langlangconcert" with strong correlation to location and age range. The backbone can be used to suggest potential communities that include a group of users along with new social relationship under various similar properties. Community models without affinity attributes and assume fixed topology constraints [31] can not be applied to characterize such structures.

(2) The academic backbone on Citation suggests potential collaborators driven by their potential co-authorship, common research

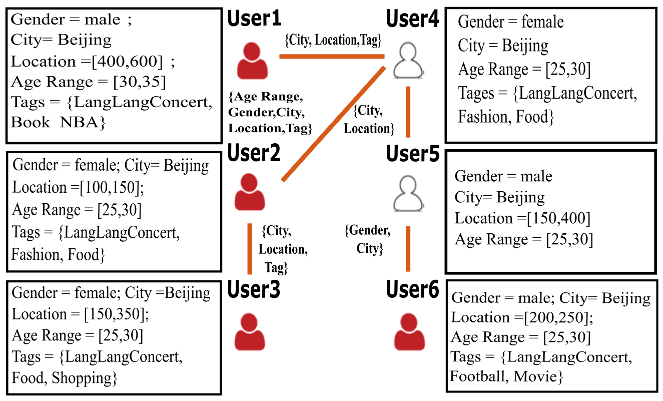


Figure 8: A social backbone of Weibo users

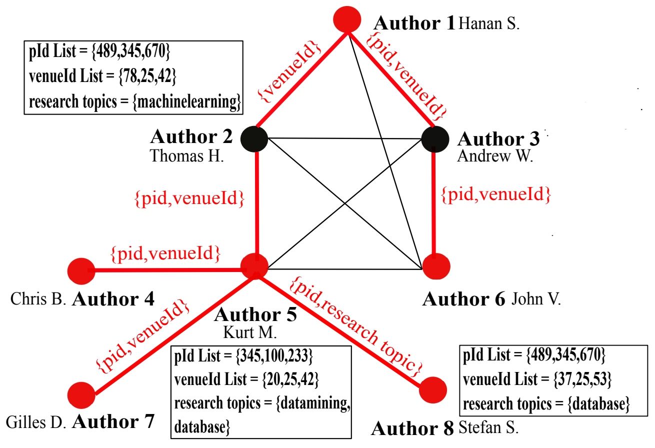


Figure 9: Academic Backbone

topics or venues (marked by red; Fig. 9). We found that CESNA can only discover a small fraction induced by attributed community set with Author nodes {1,2,3},{4,5,8} (missing Authors 6 and 7), and KCoreGC stops at k-cores including only the Author nodes{1,5,6} .

6 CONCLUSION

We have introduced attributed backbones, which explicitly incorporate affinitive attributes to interpret connectivity in attributed networks. We have developed both approximation and fast heuristics, to cope with backbone detection in networks with small number of attributes, and with large attribute size and no explicitly specified edge cost model, respectively. Our experimental study verified that these algorithms can efficiently identify interpretable critical structures that cannot be identified by conventional community detection. A future work is to generalize our methods to identify attribute-driven graphs under constraints such as density.

ACKNOWLEDGMENTS

This work is supported in part by USDA/NIFA 2018-67007-28797, Siemens, and the Department of Energy under DE-IA0000025 for UI-ASSIST project.

REFERENCES

[1] Nikolaus Augsten and Michael H Böhlen. 2013. Similarity joins in relational database systems. *Synthesis Lectures on Data Management* (2013).
[2] Marc Bailly-Bechet, Christian Borgs, Alfredo Braunstein, J Chayes, A Dagkessamanskaia, J-M François, and Riccardo Zecchina. 2011. Finding undetected protein associations in cell signaling by belief propagation. *PNAS* (2011).
[3] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. 2015. Clustering attributed graphs: models, measures and methods. *Network Science* (2015).
[4] An-Jung Cheng, Yan-Ying Chen, Yen-Ta Huang, Winston H Hsu, and Hong-Yuan Mark Liao. 2011. Personalized travel recommendation by mining people attributes from community-contributed photos. In *ACM Multimedia*.
[5] Vincent W Cheng, Jamil Y Khan, and Robert I Chaplin. 1998. Development of an integrated backbone network for a high capacity PCN network. In *Global Communications Conference*.
[6] Mung Chiang, Henry Lam, Zhenming Liu, and Vincent Poor. 2013. Why Steiner-tree type algorithms work for community detection. In *AISTATS*.
[7] Fabián A Chudak, Tim Roughgarden, and David P Williamson. 2004. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming* (2004).
[8] David Combe, Christine Largeron, Mathias Géry, and Előd Egyed-Zsigmond. 2015. I-louvain: An attributed graph clustering method. In *IDA*.
[9] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. 2007. Finding top-k min-cost connected trees in databases. In *ICDE*.
[10] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2018. Community detection in Attributed Network. In *WWW*.
[11] Christos Faloutsos, Kevin S McCurley, and Andrew Tomkins. 2004. Fast discovery of connection subgraphs. In *KDD*.
[12] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *PVLDB* (2016).
[13] Paulo Feofiloff, Cristina G Fernandes, Carlos E Ferreira, and José Coelho De Pina. 2010. A note on Johnson, Minkoff and Phillips' algorithm for the prize-collecting Steiner tree problem. *arXiv* (2010).
[14] Michel X Goemans and David P Williamson. 1995. A general approximation technique for constrained forest problems. *SICOMP* (1995).
[15] Andrey Gubichev and Thomas Neumann. 2012. Fast approximation of steiner trees in large graphs. In *CIKM*.
[16] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *PVLDB* (2017).
[17] Mehdi Kargar and Aijun An. 2011. Keyword search in graphs: Finding r-cliques. *PVLDB* 4, 10 (2011), 681–692.
[18] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. 2015. K-core decomposition of large networks on a single PC. *PVLDB* 9, 1 (2015), 13–23.
[19] Myunghwan Kim and Jure Leskovec. 2011. Modeling social networks with node attributes using the multiplicative attribute graph model. *arXiv* (2011).
[20] K Kumar. 2012. Optimization of minimum cost network flows with heuristic algorithms. *International Journal of Information and Education Technology* (2012).
[21] Rohit Kumar and Toon Calders. 2017. Information propagation in interaction networks. In *EDBT*.
[22] Matteo Lissandrini, Davide Mottin, Yannis Velegrakis, and Themis Palpanas. 2018. X 2 Q: your personal example-based graph explorer. *PVLDB* 11, 12 (2018), 2026–2029.
[23] Dániel Marx. 2008. Parameterized complexity and approximation algorithms. *Comput. J.* 51, 1 (2008), 60–78.
[24] Mark EJ Newman. 2004. Coauthorship networks and patterns of scientific collaboration. *PNAS* 101, suppl 1 (2004), 5200–5205.
[25] Bobo Nick, Conrad Lee, Pádraig Cunningham, and Ulrik Brandes. 2013. Simmelian backbones: Amplifying hidden homophily in facebook networks. In *ASONAM*.
[26] H. Tong and C. Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. In *SIGKDD*.
[27] Vijay V. Vazirani. 2003. *Approximation Algorithms*. Springer.
[28] Haixun Wang and Charu C Aggarwal. 2010. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*.
[29] Yubao Wu, Ruoming Jin, Xiaofeng Zhu, and Xiang Zhang. 2015. Finding dense and connected subgraphs in dual networks. In *ICDE*.
[30] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *ICDM*.
[31] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. When engagement meets similarity: efficient (k, r)-core computation on social networks. *PVLDB* 10, 10 (2017), 998–1009.