# Effective Path Summary Visualization on Attributed Graphs

Duncan Yung and Shi-Kuo Chang
Department of Computer Science
University of Pittsburgh, PA, USA
{duncanyung,chang}@cs.pitt.edu

## ABSTRACT

When a new dataset is modeled as an attributed graph or users are not familiar with the data, users may not know what can be retrieved from the attributed graph. Sometimes, users may have some intuition about the query, but how to exactly formulate queries (e.g. what attribute constraints to use) is still unclear to users. In this paper, we propose the idea of attributed path summary. In general, attributed path summary is a grouping of vertices such that vertices in each group contain paths from source to destination and the entropy of attributed values within a group is low and biased toward the intuition (i.e. preferred attribute values) given by users. We propose a novel 3-phrase approach which stitches key vertices together to form candidate paths and inflates those candidate paths into path summary. An extensive case study and experimental evaluation using the real Facebook graph that visualizes the path summary demonstrates the usefulness of our proposed attributed path summary as well as the superiority of our proposed techniques.

## 1. INTRODUCTION

Attributed graph is widely used for modeling a variety of information networks [11, 15], such as the web, sensor networks, biological networks, economic graphs, and social networks. When a new dataset is modeled as an attributed graph or users are not familiar with the data, users may not know what can be retrieved from the attributed graph. Sometimes, users may have some intuition about the query, but how to exactly formulate queries (e.g. what attribute constraints to use) is still unclear to users.

In this paper, we propose the idea of visualizable attributed path summary. In general, an attributed path summary is a grouping of vertices such that vertices in each group contain a path from source to destination and the entropy of attributed values within a group is low and biased toward the intuition (i.e. attribute values) given by users. In addition, we argue that a visualizable attributed path summary can be easily visualized and understood by users.

### 1.1 Application Scenario


10.18293/DMSVLSS2017-019.

**Social Network:** For example, an FBI agent has a social network, but he/she is not familiar with the attribute values and graph structure of the social network. The agent wants to investigate the relationship between Duncan and a terrorist leader using the social network as the FBI believes that social network would contain a lot of useful insight for investigation. The agent just got an intuition that people between Duncan and the terrorist leader may live in the country $C_1$ and believe in religion $R_1$,$R_2$. The attributed path summary query computes a summary of paths from Duncan to the terrorist leader that are close to the offered attribute values (i.e. $C_1$,$R_1$ or $R_2$). The path summary offers insight for the agent to formulate different path queries for investigation.

**Metabolic Network:** In metabolic networks, each vertex is a compound, and an edge between two compounds indicates that one compound can be transformed into another one through a certain chemical reaction. Vertex attributes can be features of the compound; edge attributes can be details of a chemical reaction between two compounds. A reachability query on metabolic networks discovers whether compound A can be transformed to compound B under given path attribute constraints. A biologist wants to study how to transform compound A to compound B. The biologist only knows that cost-to-trigger-reaction has to be around $10. The attributed path summary computes a summary of paths from compound A to compound B that are close to the offered attribute value (e.g. cost-to-trigger-reaction$\approx$ $10). The path summary offer insight for the biologist to formulate path queries for the study.

### 1.2 Challenges

Nowadays, a big graph with a few million vertices is common, and that results in an exponential number of paths between any two vertices. A large number of possible paths between 2 vertices makes computing path summary a challenging task.

Among a huge number of possible paths between 2 vertices, which type of path the user prefers is unknown since even the user is not familiar with the graph, and he/she may not know what he/she can get from the graph. Therefore, our task is to compute a path summary for the user.

Computing an effective summary for a user is non-trivial as no user would prefer to read a lot of text to understand the summary. Hence, an effective path summary is a summary that can be easily visualized by users. Visualizing a large portion of the graph is not feasible as that would overwhelm the user. On the contrary, if the summary is too concise, the user may not get the information he/she wants.

## 1.3 Our Contributions

Our first contribution is to introduce and define the attribute path summary query on attributed graph problem. We define attributed path summary to be groups of vertices that contain users' intuition as well as satisfy some path properties. The users' intuition is expressed as hints for computing the path summary. Users can offer whatever attribute values that they consider as the hint. These summaries offer insight to users about the attribute values and connection between the given source and destination vertices.

Our second contribution is to propose an efficient and effective approach for finding attributed path summary. Our proposed approach consist of three phrases. The first phrase efficiently finds all key vertices that have attribute values belonging to the hint offered by the user. Including key vertices ensures the summary would represent paths with attribute values that are close to the intuition of users. Then, based on those key vertices, a novel stitching algorithm is proposed to connect the source, the destination, and key vertices together to form a relatively small key vertex graph. The stitching algorithm finds paths with a small variation in attribute values between key vertices so that users can easily understand the attribute distribution between key vertices. After that, high-quality candidate paths between the source and the destination are found on that small key vertex graph efficiently. Finally, candidate paths are inflated to vertex groups by greedily including adjacent vertices. Including adjacent vertices would offer more attribute values choices for users to formulate their queries.

## 1.4 Paper Organization

Section 2 talks about related works. Section 3 presents definitions and problem statement. Section 4 gives details of our approach for finding attributed path summary. Section 5 presents an extensive experimental evaluation for the proposed approach. We conduct case studies on the Facebook graph that visualize our path summary results for illustrating the effectiveness of our proposed path summary. We also conduct experiments to study the change in entropy and execution time under different parameter settings. Finally, Section 6 concludes this paper.

# 2. RELATED WORK

In this section, we present a summary of related works.

## 2.1 Attributed Graph Summarization

Graph summarization has been extensively studied [10, 13, 13, 17, 14, 16, 4, 7, 12, 3, 2], and various ways of summarizing graphs have been proposed. Grouping-based summarization methods [10, 13, 13, 17, 14] takes into account both graph structure and attribute distributions for aggregating vertices into supernode and superedges; compression-based summarization methods [16, 4, 7] exploit the MDL principle to guide the grouping of vertices or the discovery of frequent sub-graphs to form a graph summary; influence-based summarization methods [12] leverage both graph structure and vertex attribute value similarities in the problem formualtion so as to summarize the influence process in a network; pattern-mining-based summarization methods [3, 2] identify frequent graph structural patterns for aggreagate into supernodes so as to reduce the size of the input graph and as a result, improving query efficiency. These techniques focus on computing summary for the whole graph. On the other hand, our techniques focus on computing visualizable path summary between two vertices that users are interested in.

## 2.2 Attributed Graph Clustering

Zhou et al [18] proposed $SACluster$, which is an attributed graph clustering algorithm based on both graph structural and attribute similarities through a unified distance measure. Zhou et al [18] proposed first to partition a large graph associated with attributes into k clusters so that each cluster contains a densely connected subgraph with homogeneous attribute values. Then, an effective method is used to automatically learn the degree of contributions of structural similarity and attribute similarity. Zhou et al [19] further improve the efficiency and scalability of $SACluster$ [18] by proposing an efficient algorithm $IncCluster$ to incrementally update the random walk distances given the edge weight increments.

One fundamental difference between summarization and clustering is that former finds coherent sets of vertices with similar connectivity patterns to the rest of the graphs, while clustering aims at discovering coherent densely-connected groups of vertices [8]. Similar to graph summary, graph clustering only computes a summary of the whole graph while our techniques focus on a summary of paths between two vertices.

## 2.3 Graph Visualization

The size of the graph to view is a key issue in graph visualization [6]. To deal with this, researchers proposed a lot of techniques in graph drawing [6], such as H-tree layout, radial view, balloon view, tree-map, spanning tree, cone tree, hyperbolic view, as well as methods for reducing visual complexity [9], such as clustering, sampling, filtering, partitioning. We argue that simply applying those graph drawing technique cannot handle big attributed graphs with million of vertices and edges as these methods are too general. For existing visual complexity reduction methods, how to effectively applying them to our problem needs further investigation.

# 3. PRELIMINARIES

## 3.1 Problem Statement

DEFINITION 1. **[Attributed Graph]** *An attributed graph [15] $G$, is an undirected graph denoted as $G = (V, E, A_v)$, where $V$ is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $A_v = \{A(v)\}$ is a set of $d_v$ vertex-specific attributes, i.e. $\forall v \in V$, there is a multidimensional tuple $A(v)$ denoted as $A(v) = (A_1(v), A_2(v), ..., A_{d_v}(v))$.*

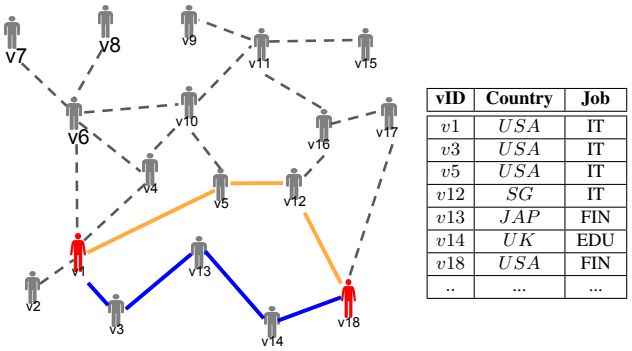
DEFINITION 2. **[Attribute Hint $H$]** *is a set of distinct attribute values.*

$$H = \{H_1, H_2, .., H_{d_v}\}$$

DEFINITION 3. **[Contain Function $\phi(P_i, H)$]**

$$\phi(P_i, H) = \sum_{j=1}^{|P_i|} contain(v_j, H)$$

$$contain(v_j, H) = \begin{cases} 1, & \forall k = 1..d_v \ \ if\ \exists A_k(v_j) \in H_k \\ 0, & otherwise \end{cases}$$




| vID | Country | Job |
|---|---|---|
| $v1$ | $USA$ | IT |
| $v3$ | $USA$ | IT |
| $v5$ | $USA$ | IT |
| $v12$ | $SG$ | IT |
| $v13$ | $JAP$ | FIN |
| $v14$ | $UK$ | EDU |
| $v18$ | $USA$ | FIN |
| .. | ... | ... |

**Figure 1: Path Summary ($P_1$-blue, $P_2$-orange)**

For example in Figure 1, given that $H = \{\{USA, SG, JAP\}, \emptyset\}$,$P_1 = \{v1, v3, v13, v14, v18\}$, and $P_2 = \{v1, v5, v12, v18\}$, $\phi(P_1, H) = 1+1+1+0+1 = 4$ and $\phi(P_2, H) = 1+1+1+1 = 4$.

DEFINITION 4. **[Attributed Path Summary $PSum(G, s, t, l, H)$]** *For an attributed graph $G$, $PSum(G, s, t, l, H)$ is a set of vertices $\{P_1, P_2, ..., P_k\}$ such that:*

1. *$\forall\, v \in P_i$ are connected,*
2. *$\exists\, v, v' \in P_i$, $v$ is adjacent to $s$ and $v'$ is adjacent to $t$,*
3. *$\forall P_i, P_j \in G, i \neq j, P_i \cap P_j = \emptyset$,*
4. *$\forall P_i, \phi(P_i, H) \geq l$,*
5. *$\forall v \in P_i, dist(s, v) + dist(v, t) \leq l$, and*
6. *$\nexists P \in G \wedge P \notin PSum(G, s, t, l, H)$ satisfing condition 1 to 5.*

*where $\phi(P_i, H)$ is the quality of the summary and $dist(v, v')$ is the shortest distance from $v$ to $v'$.*

Continuing the above example, in Figure 1, given that $l = 4$, there are two paths $P_1, P_2$ in the attributed path summary. They are $P_1 = \{v1, v3, v13, v14, v18\}$ and $P_2 = \{v1, v5, v12, v18\}$. For example, all vertices in $P_1$ are connected, $\exists\, v_3$ and $v_14$ adjacent to $s$ and $t$, $P_1 \bigcap P_2 = \emptyset$, $\phi(P_1, H) = \phi(P_2, H) = 4 = l$, and for all $v_i \in P_1, P_2$, and $dist(s, v_i) + dist(v_i, t) \leq 4$.

**Problem Statement**

**[Attributed Path Summary Query $q_p$]** Given an attributed graph $G = (V, E, A_v)$, source $s$, destination $t$, attribute hint $H$, and lower bound of number of vertices in every $P_i$ that contains at least one attribute value in attribute hint $l$, $q_p$ return an attributed path summary $PSum(G, s, t, l, H)$.

## 3.2 Quality of Path Summary

The quality of path summary is defined as the entropy in [18] and is reformulated in definition 5.

DEFINITION 5. **[Path Summary Quality]**

$$entropy(P_j) = \sum_{i=1}^{d_v} entropy(a_i, P_j)$$

*where*

$$entropy(a_i, V_j) = -\sum_{n=1}^{n_i} p_{ijn} log_2 p_{ijn}$$

*and $k$ is the number of $P_i$ in $PSum$, $p_{ijn}$ is the percentage of vertices in $P_j$ which have value $a_n$ on attribute $a_i$.*

For example, $entropy(Country, P_1) = -(\frac{3}{5}log_2\frac{3}{5} + \frac{1}{5}log_2\frac{1}{5} + \frac{1}{5}log_2\frac{1}{5})$.

# 4. COMPUTING PATH SUMMARY

In this section, we introduce our path stitching approach for computing attributed path summary effectively based on attribute hint.

## 4.1 Algorithm Design

Our heuristic approach has the following steps and design principles.

1. Firstly, we want to find all vertices - key vertices, that are related to the given attribute hint. The search of key vertices ensures that all vertices that match any attribute value in the hint and fulfill the distance requirement (Condition 5, Definition 4) are used for computing a path summary.

2. Given those key vertices, we perform a concurrency graph traversal that systematically stitches key vertices, the source, and the destination. Using stitched key vertices, we find candidate paths that go from the source to the destination via key vertices based on the entropy of attribute values on the path. Key vertices are vertices that users care and want to see in the visualized path summary. The stitching algorithm can effectively connect key vertices so that attribute values on the path between key vertices are consistent. That offers a clear view for users to understand the attribute distribution between key vertices.

3. Finally, given the candidate paths, we perform a candidate path inflation for computing the path summary. Candidate path inflation includes vertices close to vertices in candidate paths into the candidate paths. That allows users to understand attribute distributions around key vertices. When users are considering what attribute constraint to use for their attribute graph queries, they can consider attribute values on candidate paths as well as attribute values close to the candidate path as an alternative.

Algorithm details are presented in below sections with conceptual examples.

## 4.2 Finding Key Vertices

We first introduce the concept of key vertex (Definition 6). Then, we present two steps that exploit existing approach to efficiently find all key vertices.

DEFINITION 6. **[Key Vertex $v^k$]** *is a vertex that has at least one attribute value belonging to an attribute value in the attribute hint $H$.*

$$\forall i = 1..d_v \forall j = 1..d_v \;\; \exists A_i(v^k) \in H_j$$

$$where \;\; H_j \in H$$

The first step is to retrieve all key vertices. Traditional indexes that support range query (e.g. B+ tree) can be used to index each attribute. Given $H$, for each non-empty $S_j \in H$, we query the corresponding index for a set of vertices that have attribute

values in $S_j$. Then, we do a union of all these vertices and get the key vertex set $V_k$. After that, all vertices $v$ that does not satisfy $dist(s,v) + dist(v,t) \leq l$ are filtered out.

For example, in Figure 1, if the hint contains only $Country = USA$, all vertices with attribute value $Country = USA$ (e.g. $v_1, v_3, v_5, v_{18}$) are retrieved from the precomputed index (e.g. B+ tree).

## 4.3 Finding Candidate Path

After all key vertices are found, the second step is to find candidate paths that satisfy constraints in Definition 4.

### 4.3.1 Stitching Algorithm

Since key vertices are essential (so as to satisfy condition 4 in Definition 4) for paths in path summary, we do not want to find candidate paths that do not contain any key vertex. The idea of the stitching algorithm is to connect $s - t$ and key vertices so as to form candidate paths. During the graph traversal, entropy and hop distance values are taken into account.

Algorithm 1 is the pseudo code of the stitching algorithm. The stitching algorithm first puts $s$, $t$, and all key vertices (lines 7-13) into the priority queue. Each node in the priority queue contains the current vertex, a key vertex, parent of current vertex, the distance from a key vertex to the current vertex, and the entropy of path from a key vertex to the current vertex, where the entropy value is used to determine the priority.

Then, the graph is traversed starting from each of the key vertices. When the algorithm reaches a visited node $cur.v$ (line 16), if key vertex of current node's parent ($key1$) is not equal to the key vertex of curent node ($key2$) (line 19), the path from $key1$ to $key2$ is recovered and put into $PathMap$ (line 21), edges between vertice $key1$ and $key2$ are added in to $KeyVertexG$ (lines 22-23), and the algorithm continue (line 22); when the algorithm reaches a non-visited node (line 25), parent and key vertex of current node is saved (line2 26-27) and current node becomes visited (line 28).

After that, adjacent neighbors of $cur.v$ that satisfy the upper bound distance constraint (line 30) are put into the priority queue (line 33), where the entropy of the path from the key vertex to $cur.v$ as well as the distance are taken into account. Finally, the graph traversal continues until the priority queue becomes empty.

A conceptual example will be presented in Section 4.3.3.

### 4.3.2 Candidate Path Search

After executing the stitching algorithm, $KeyVertexG$ and $Path$ are found. The path search algorithm is used to find paths from $s$ to $t$ via key vertices in the key vertex graph - $KeyVertexG$. The actual path are recovered using $path$ after $s - t$ in $KeyVertexG$ are found. Both entropy and distance from $s$ are taken into account in the priority queue. We set priority as $entropy + current\ \ distance/l$ if $current\ \ distance < l$; otherwise, we set priority as $entropy + current\ \ distance$, in order to pennalize path in $KeyVertexG$ that are longer than $l$.

Algorithm 2 is the pseudo code of the path search algorithm.

**Algorithm 1** Stitching Algorithm

```
1:  procedure STITCHING(G, V_k, s, t, l)
2:      Array<bool> visited
3:      Array<int> parents
4:      Array<Array<int>> KeyVertexG
5:      Array<int> keys
6:      priority_queue<node> qu                    ▷ lower entropy first
7:      node src(s, s, s, 0, 0)                    ▷
            (vert., keyVert, parent, dist, entropy)
8:      qu.push(src)
9:      node dest(t, t, t, 0, 0)
10:     qu.push(dest)
11:     for all v^k ∈ V_k do
12:         node n(v^k, v^k, v^k, 0, 0)
13:         qu.push(n)
14:     while !qu.empty() do
15:         cur ← q.pop()
16:         if visited[cur.v] == true then
17:             key1 ← keys[parents[cur.v]]
18:             key2 ← cur.keyVert
19:             if key1 == key2 then   ▷ it is just a cycle but not
                    meeting of 2 traversals from diff KeyVertex
20:                 continue
21:             PathMap ← ComputePathBetween(key1, key2)
22:             KeyVertexG[key1].push(key2)
23:             KeyVertexG[key2].push(key1)
24:             continue
25:         else visited[cur.v] == false
26:             parents[cur.v] ← cur.parent
27:             keys[cur.v] ← cur.keyVert
28:             visited[cur.v] ← true
29:         for all v ∈ G[cur.v].adj do int v = topology[cur.v][i];
30:             if dist_s[v] + dist_t[v] > l then
31:                 continue
32:             en ← CompEntropy(cur.keyVert, cur.v) +
                    (cur.dist + 1)/l
33:             node n(v, cur.keyVert, cur.v, cur.dist + 1, en)
34:             qu.push(n)
35:     return (KeyVertexG, Path)
```

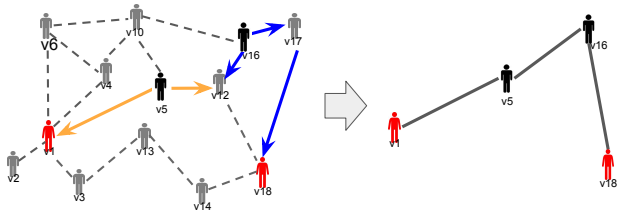



**Figure 2: Stitching Algorithm (left) and Candidate Path (right)**

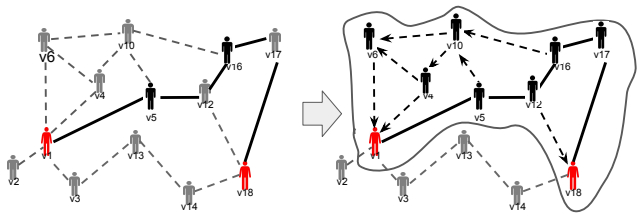



**Figure 3: a Candidate Path (left) and a Vertex Group $P_i$ (right)**

Algorithm 2 finds shortest path from $s$ to $t$ on $KeyVertexG$ based on entropy value (line 5). After a path $p$ is found, $p$ is removed from $KeyVertexG$ (line 7). The algorithm continues until no more path can be found.

### 4.3.3 Conceptual Example

Figure 2 illustrates the concept of stitching algorithm and candidate path search. Suppose $v_5$ and $v_{15}$ are key vertices retrieved from the index and $v_1$ and $v_{18}$ are $s$ and $t$ respectively. $v_5$ expands to $v_1$, and edges from $v_1$ to $v_5$ and $v_5$ to $v_1$ are put into $KeyVertexG$. $v_5$ also expands to $v_{12}$. $v_{15}$ expands to $v_{17}$ and $v_{12}$. When $v_{15}$ expands to $v_{12}$, $v_{12}$ was occupied by $v_5$ already. Hence, we can put edges $v_{15}$ to $v_5$ and $v_5$ to $v_{15}$ into $KeyVertexG$. After that, $v_{15}$ expands to $v_{18}$ , and edges from $v_{15}$ to $v_{18}$ and $v_{18}$ to $v_{15}$ are put into $KeyVertexG$. Given the $KeyVertexG$, candidate paths from $s$ to $t$ are found based on entropy values, and those candidate paths will be used for path inflation in the next phrase.

**Algorithm 2** Path Search Algorithm

```
1:  procedure PATHSEARCH(KeyVertexG, s, t, l, α)
2:      boolean PathFound ← true
3:      Array < Path > CandPath
4:      while PathFound == true do
5:          p ← FindShortestPath(KeyVertexG, s, t, l, α)
6:          if p! = ∅ then
7:              RemovePath(p, KeyVertexG)
8:              CandPath.push_back(p)
9:          else
10:             PathFound ← false
11:     return CandPath
```

## 4.4 Candidate Path Inflation

After all candidate paths, $CandPath$ are found, the candidate paths are used to form path summary. We developed the path inflation algorithm which greedily includes vertices into path vertex groups.

Algorithm 3 is the pseudo code of the path inflation algorithm. Firstly, all vertices in the $CandPath$ are put into a priority queue which uses entropy as the priority (lines 4-8). Then, the vertex $cur$ in the candidate path with the lowest entropy are popped from the priority queue (line 10). If $cur$ was not visited before, $cur$ is included in the path group $PathSummary[cur.PathID]$ (line 14). After that, all adjacent vertices of $cur$ that satisfy $dist_s[v] + dist_t[v] > l$ *or* $(cur.dist + 1) * 3 > dist_s[t]$ (line 19) are pushed into the priority queue with $entropy(cur.P \cup v)$ as cost. $(cur.dist + 1) * 3 > dist_s[t]$ is included so as to prevent vertices that are too far away from vertices in $CandPath$ are included in the path summary. The algorithm terminates when the priority queue becomes empty.

Figure 3 illustrates the concept of candidate path inflation. Given that $v_1 \rightarrow v_5 \rightarrow v_{12} \rightarrow v_{16} \rightarrow v_{17} \rightarrow v_{18}$ is the candidate path. The path inflation algorithm first puts all vertices (i.e. $v_5, v_{12}, V_{16}, v_{17}$) in the candidate path into the priority queue with entropy of the candidate path as priority. Firstly, vertices that are adjacent to $v_5, v_{12}, V_{16}, v_{17}$ are included into the candidate path. Then, other vertices that are adjacent to vertices (e.g. $v_4, v_6$) in the candidate path are gradually included in the candidate path until the distance constraint. Finally, we will get a subgraph shown in Figure 3 (right).

**Algorithm 3** Path Inflation Algorithm

```
1:  procedure PATHINFLATION(G, CandPath, s, t, l)
2:      priority_queue < node > qu
3:      Array < bool > visited
4:      for all p ∈ CandPath do
5:          for all v ∈ p do
6:              entropy ← ComputeEntropy(p)
7:              node n(v, i, v, 0, l, entropy)
8:              qu.push(n)
9:      while !qu.empty() do
10:         cur ← qu.pop()
11:         if visited[cur.v] == true then
12:             continue
13:         visited[cur.v] ← true
14:         PathSummary[cur.pathID].push(cur.v) ▷ assign
                v into that path group
15:         for all v ∈ G[cur.v].adj do
16:             if dist_s[v] + dist_t[v] > l or (cur.dist + 1) * 3 >
                    dist_s[t] then
17:                 continue
18:             en = ComputeEntropy(PathSummary[cur.pathID], v)
19:             node n(v, cur.pathID, cur.dist +
                    1, cur.l, cur.keyVertex, en)
20:             qu.push(n)
21:     return PathSummary ▷ return Path Summary
```

# 5. EVALUATION

All experiments were performed under 64-bit Linux Ubuntu 14.04 on a machine with an Intel 4GHz CPU (4-core), 16 gigabytes of memory, and 1 terabyte solid state drive with 512k block size. All our implementations are in C++ without parallelism.

We first introduce the graph dataset and attributes that we used for the experiments. Then, we present the result of our case studies. Finally, we look at the change of change of path summary quality (i.e. change of entropy) along with the change in the expected number of key vertex $l$ and the number of hints $H$.

**Table 1: Dataset and Parameter**

| Real Graph | Num of Vertex | Num of Edge |
|---|---|---|
| *fb-bfs1 [5]* | *1.18m* | *29.78m* |
| **Parameter** | **Default** | **Vary** |
| *Exp. Num of Key Vert.* | *6* | *3,6,9,12* |
| *Num of Hint* | *3* | *1,3,6,12* |

## 5.1 Datasets

We used a real social network dataset $fb\text{-}bfs1$ [5], which has 1.63m vertices and 15.14m edges, for our experiments. To control the number of attributes and attribute domain sizes, we generate attributes (Table 2) based on vertex attributes in facebook graph-API [1].

**Table 2: Attributes**

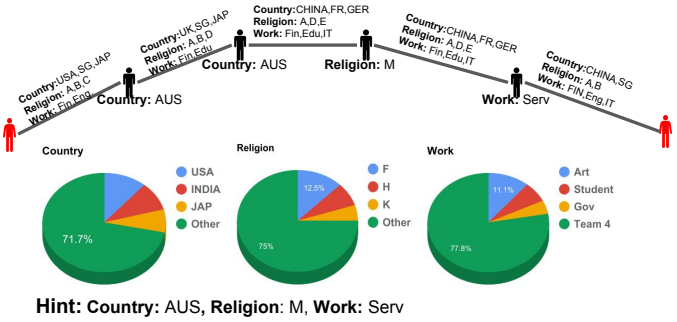
| Vertex Attribute | Domain Size,Distribution ($\mu$,$\sigma$) |
|---|---|
| *AgeGroup* | *10, gau(5,2.5)* |
| *Education* | *5, gau(3,1.25)* |
| *Gender* | *2, uni.* |
| *HomeCountry* | *100, gau(50.25)* |
| *Interested in* | *3, uni.* |
| *Languages* | *50, gau(25,12.5)* |
| *Relationship Status* | *2, uni.* |
| *Religion* | *20, gau(10,5)* |
| *Work* | *50, uni.* |
| *Political* | *10, gau(5,2.5)* |

## 5.2 Case Study

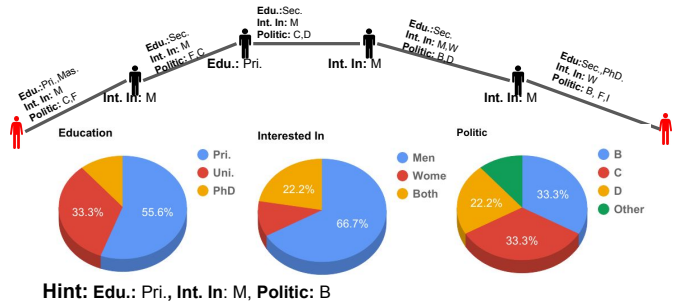
Figure 4 and 5 are four case studies using the $fb-bfs1$ [5] graph. Each of the figures contains a visualization of one of the paths in the path summary. At the top of each figure, we can see the key vertices (man icon) from source to destination. Below each key vertex is the attribute value of the key vertex that matches attribute value in the hint. Attribute value summaries of the path between every two key vertices are shown above the edges between every two key vertices. The pie chats below the path are the summaries of attribute value found by the inflation algorithm. This attribute value summary summarizes the attribute value near to the key vertices.

**Case 1:** For the first case study in Figure 4(a), we set the expected number of key vertex $l = 6$, the number of hint $H = 3$, and the hint contains attribute $Country = AUS$, $Religion = M$, and $Work = Service$. We can see there are 6 key vertices (including source and destination), which match our expected number of key vertices. Furthermore, the summaries of attribute values on the path between every two key vertices are concise. That gives users a clear idea of attribute values between key vertices. From the pie charts, we can see that *other* (green) occupies a large portion of the pie. That tells users that attribute values close to the path are inconsistent and probably having large attribute value domain.

**Case 2:** For the first case study in Figure 4(b), we set the expected number of key vertex $l = 6$, the number of hint $H = 3$, and the hint contains attribute $Education = Primary$, $Interested\ In = Men$, and $Politic = B$. We can see there are 6 key vertices (including source and destination), which matches our expected number of key vertices. Furthermore, the summaries of attribute values on the path between every two key vertices are concise. That gives users a clear idea of attribute values between key vertices. From the pie charts, we can see that the top-2 attribute values (blue and red) in each attribute occupies a large portion of the pie. That tells users that attribute values close to the path are consistent and that helps users to efficiently construct their queries.

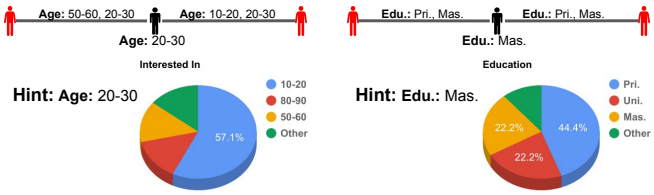



(a) Case 1




(b) Case 2

**Figure 4: Path Summary (Expected Num of Key Vertex=6, Num of Hint=3)**




(a) Case 3 (b) Case 4

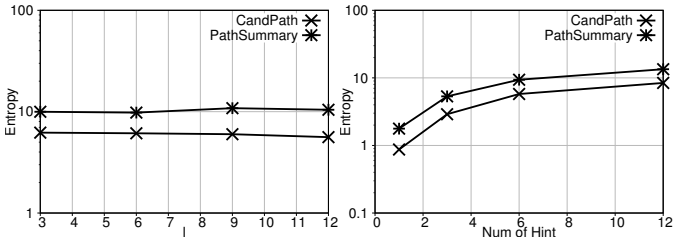
**Figure 5: Path Summary (Expected Num of Key Vertex=3, Num of Hint=1)**

After we study path summary with 6 key vertices and 3 hints, we try to look at cases with less key vertices and hints.

**Case 3:** For the first case study in Figure 5(a), we set the expected number of key vertex $l = 3$, the number of hint $H = 1$, and the hint contains attribute $Age = 20 - 30$. We can see there are 3 key vertices (including source and destination), which matches our expected number of key vertices. Furthermore, the summaries of attribute values on the path between every two key vertices are also concise. From the pie charts, we can see that the top-1 attribute values (blue) in each attribute occupies a large portion of the pie. On the contrary, the "*other*" attribute value (green) only occupies a small portion. That tells users that attribute values close to the path are very consistent.

**Case 4:** For the first case study in Figure 5(b), we set the expected number of key vertex $l = 3$, the number of hint $H = 1$, and the hint contains attribute $Education = Master$. We found similar result as in Figure 5(a).

## 5.3 Query Formulation Using Path Summary




(a)Vary l (b)Vary Num of Hint

**Figure 6: [fb-bfs1] Entropy**

In order to connect source and destination via vertices that satisfy attribute hint, we suggest users take into account major attribute values and alternative attribute values when they are formulating queries.

**Major Attribute Values:** Major attribute values are attribute values that appear on the path between key vertices. For example, in Figure 4(b), "**Edu.**:Sec.,PhD., **Int. In**: W, **Politic**: B,F,I" are major attribute values between destination and the last key vertex. By putting these attribute values into the query, key vertices can be connected. However, based on users preferences, they may not always want to include these major attribute values. Continue with the example in Figure 4(b), users may not want to include "**Edu.:**Sec.,PhD" into the query. If that is the case, users can consider the alternative attribute values.

**Alternative Attribute Values:** Alternative attribute values are attribute values displayed in the pie charts. They are the distribution of attribute values near to paths between key vertices. Continue with the example in Figure 4(b), if users do not prefer to have "**Edu.**:Sec.,PhD" in the query, they may consider to replace it by "**Edu.**: Uni". Based on the "Education" pie chart, there are $33.3\%$ of vertices has attribute value "**Edu.**: Uni" near to the paths between key vertices. Therefore, conceptually, choosing "**Edu.**: Uni" is similar to rerouting the path between the destination and the last key vertex.

## 5.4 Change of Entropy

The default expected number of key vertex and number of hint are 6 and 3 respectively. We randomly generate 200 pairs of source and destination and measure the average entropy and execution time.

Figure 6(a) shows the change of entropy along with $l$. We can see that for both $CandPath$ and $PathSummary$, the entropy does not really increase with $l$. Although it seems that a longer path would contain more vertices and is more likely to contain different attribute values, this intuition is not supported by Figure 6(a). Since large $l$ offers more opportunity for the algorithm to search for $s-t$ paths with similar attribute values, the increase in path length does not directly imply an increase in entropy.

Figure 6(b) shows the change of entropy along with $H$. We can see that for both $CandPath$ and $PathSummary$, the entropy increases with $H$. That is contributed by the fact that more attribute hints mean more attribute are involved, which makes the consistency of attribute values lower.

# 6. CONCLUSION AND FUTURE WORK

In this paper, we study the problem of computing effective path summary for attributed graphs. We first define a meaningful definition for path summary on attribute graphs that takes into account user's intuition on attribute values as well as path structure properties. Then, we propose an effective 3-phrase algorithm that finds key vertices, stitches key vertices, and searches for path summary. Finally, case studies on the Facebook graph that visualize our path summary results illustrated the effectiveness of our proposed path summary. In the future, we plan to further reduce the number of path in the path summary by proposing the approach that can effectively merge similar paths together so as to further reduce the effort that users need for understanding the path summary.

# 7. REFERENCES


[1] https://developers.facebook.com/docs/graph-api/reference/user.

[2] S. Cebiric, F. Goasdoué, and I. Manolescu. Query-oriented summarization of RDF graphs. *PVLDB*, 8(12):2012–2015, 2015.

[3] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. Mining graph patterns efficiently via randomized summaries. *PVLDB*, 2(1):742–753, 2009.

[4] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res. (JAIR)*, 1:231–255, 1994.

[5] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proceedings of IEEE INFOCOM '10*, San Diego, CA, March 2010.

[6] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43, 2000.

[7] K. Khan, W. Nawaz, and Y. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[8] Y. Liu, A. Dighe, T. Safavi, and D. Koutra. A graph summarization: A survey. *CoRR*, abs/1612.04883, 2016.

[9] R. Pienta, J. Abello, M. Kahng, and D. H. Chau. Scalable graph exploration and visualization: Sensemaking challenges and opportunities. In *2015 International Conference on Big Data and Smart Computing, BIGCOMP 2015, Jeju, South Korea, February 9-11, 2015*, pages 271–278, 2015.

[10] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 405–416, 2003.

[11] S. Sakr, S. Elnikety, and Y. He. G-SPARQL: a hybrid engine for querying large attributed graphs. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 335–344, 2012.

[12] L. Shi, H. Tong, J. Tang, and C. Lin. VEGAS: visual influence graph summarization on citation networks. *IEEE Trans. Knowl. Data Eng.*, 27(12):3417–3431, 2015.

[13] M. Shoaran, A. Thomo, and J. H. Weber-Jahnke. Zero-knowledge private graph summarization. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 597–605, 2013.

[14] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, pages 567–580, 2008.

[15] Z. Wang, Q. Fan, H. Wang, K. Tan, D. Agrawal, and A. El Abbadi. Pagrol: Parallel graph olap over large-scale attributed graphs. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 496–507, 2014.

[16] Y. Wu, Z. Zhong, W. Xiong, and N. Jing. Graph summarization for attributed graphs. In *2014 International Conference on Information Science, Electronics and Electrical Engineering*, volume 1, pages 503–507, April 2014.

[17] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *ICDE*, pages 880–891, 2010.

[18] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[19] Y. Zhou, H. Cheng, and J. X. Yu. Clustering large attributed graphs: An efficient incremental approach. In *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 689–698, 2010.