Identifying Converging Pairs of Nodes on a Budget

Konstantina Lazaridou
Department of Informatics
Aristotle University, Thessaloniki, Greece
konlaznik@csd.auth.gr

Konstantinos Semertzidis
Computer Science and Engineering Department
University of Ioannina, Greece
ksemer@cs.uoi.gr

Evaggelia Pitoura
Computer Science and Engineering Department
University of Ioannina, Greece
pitoura@cs.uoi.gr

Panayiotis Tsaparas
Computer Science and Engineering Department
University of Ioannina, Greece
tsap@cs.uoi.gr

ABSTRACT

In this paper, we consider large graphs that evolve over time, such as graphs that model social networks. Given two instances of the graph at two points in time, we ask to identify the top pairs of nodes whose shortest path distance has decreased the most. We call these pairs *converging*. The straightforward way to address this problem is by computing the shortest path distances of all pairs at both instances and keeping the ones with the largest differences. Since for large networks this is computationally infeasible, we consider a budgeted version of the problem, where given a fixed budget of single-source shortest path computations, we seek to identify nodes that participate in as many converging pairs as possible. We evaluate a number of different approaches for our problem, that employ centrality-based, dispersion-based, and landmark-based distance estimation metrics. We also consider a classification-based approach that builds a classifier that combines the above features for predicting whether a node participates in one of the top converging pairs. We present experimental results using real-world datasets that show that we are able to identify the large majority of the top converging pairs on a very small budget.

Categories and Subject Descriptors

H.2 [**Database Management**]: Applications; E.1 [**Data Structures**]: Graphs and networks

General Terms

Algorithms, Experimentation, Performance

Keywords

graphs, top-k, shortest paths

1. INTRODUCTION

A variety of natural or man-made complex systems can be modeled as networks. Prominent examples include the Internet, the Web, transportation networks, online social networks such as Facebook, Twitter and LinkedIn, biological systems such as protein interactions or metabolic pathways, the global economy, and many more. All these systems consist of individual entities that are interconnected to form a complex system. Understanding them is not possible without understanding the underlying network. The network carries a significant amount of information about the functionality of the system as a whole as well as for its individual entities [9].

A central piece of information revealed through the network is that of *proximity*. Using the network structure, we can determine how close two individuals are in a social network, or how easy it is to navigate between different web pages. There are several ways of measuring proximity, or similarity, within the network (see, for example [6] for a recent survey). A commonly used proximity measure is the length of the shortest path distance between two nodes in the graph. Although simple, the shortest path distance is still the first notion of proximity we want to compute when examining the relationship of two individual entities. In many cases, it also provides an actionable piece of information: It is the path by which we want to route information between two individuals; the path we want to follow when moving from one place to another in a traffic network; the quickest way to create a connection within a professional network.

A succinct characteristic of real-life networks is their continuous evolution. New nodes and edges are added, often at an exponential pace. Even when observing a fixed set of nodes, their relationship can change dramatically by the addition of new edges among nodes in the set, or to other nodes outside the set. As a result, new shortest paths are created and the relationships between individuals are constantly updated.

Identifying the pairs of nodes that came the closest to each other is important in our understanding of the network, and it may be crucial for some applications. For example, in social networking sites such as *Facebook* or *LinkedIn*, if two distant users come closer over time, this could imply the appearance of similar interests or activities between them.

10.5441/002/edbt.2015.28

Hence, this further knowledge can help in making more suitable friendship recommendations. In economic networks, the decrease of the shortest path between two major players could signal a change of strategy, or have future implications for their growth. In a criminal or terrorist network, it is critical to know which suspects have come closer to each other; such moves may be indications of future actions or coalitions.

There could be also an application of this problem in protein-protein interaction networks, where the nodes are proteins within a cell and they are connected by edges if there is a possible interaction between them [3]. As new experiments reveal new connections, for two given proteins, the knowledge that they came closer together in the graph makes them candidates for an upcoming interaction. Furthermore, if a certain protein comes closer to multiple others, they may be part of the same community [12], with all underlying proteins having the same specific function within the cell.

In this paper, we address the following problem. Given two snapshots of an evolving network, we ask for the k pairs of nodes whose shortest path distance has decreased the most between the two snapshots. We call such pairs of nodes *converging*. There is a simple polynomial-time solution to our problem: Compute the all-pair shortest path distances in the two graph instances and find the pairs with the largest distance change. However, even if we used fast algorithms that approximate shortest path computation, e.g., [20], it would still require time quadratic in the size of the graph for just producing the pairs. Given that real graphs have size in the order of thousands or millions, we need solutions that scale linearly with the size of the graph.

Despite its significance, the problem of identifying the top-k converging pairs has received limited attention. The only related work we are aware of is the work in [14] which provides an approach based on the endpoints of the new edges in the second snapshot. In this paper, we address the problem from a different perspective, by predicting the nodes of the converging pairs. We formally define good candidate nodes as those nodes that belong to a *cover* of the set of the top-k converging pairs. We also introduce a novel budget formulation of the problem, where to achieve the required result, we are given a fixed budget of $m \ll n$ single-source shortest-path computations. We propose a suite of algorithms for identifying good candidate nodes based purely on the structural properties of the two graph instances. These algorithms are based on node centrality, node dispersion and landmark-based distance estimations. Then, we build a classifier that combines the proposed algorithms to effectively identify the approach that is the most appropriate for each setting. By further extending the classifier to include features of the graph, such as the graph density, we were able to build a "global" classifier that works on any graph. The classifier takes advantage of our novel method of characterizing good candidate nodes.

Our experiments with four real datasets show that we can identify most of the converging pairs with a budget equal to a very small percentage of the graph nodes. For example, for the *Internet links* dataset, with a budget of just 0.5% of the nodes, we are able to locate over 90% of the top-k converging pairs for various values of k. Furthermore, our classifiers are successful in identifying converging pairs and match the performance of the best algorithm for each dataset.

In summary, in this paper we make the following contributions:

- We formalize the problem of finding the top-k converging pairs in a graph under budget constraints as a problem of finding a vertex cover of an appropriately defined graph over the set of these pairs.
- We propose a suite of methods for identifying candidate nodes that best cover the set given a fixed budget of shortest path computations.
- We show how to combine our techniques into a single algorithm that makes use of all the proposed features.
- We perform extensive experiments that show that our techniques work well in practice, being able to find the top-k converging pairs with only a small number of shortest path computations. Compared to the approach in [14] our algorithms are more effective in identifying converging pairs, while having strong budget guarantees.

The rest of this paper is structured as follows. In Section 2, we present related work. In Section 3, we introduce the problem of identifying the top-k converging pairs and formulate it as a vertex cover problem. In Section 4, we present a suite of algorithms for identifying candidate nodes for the converging pairs. Experimental results are presented in Section 5. Finally, in Section 6, we provide conclusions and directions for future work.

2. RELATED WORK

Dynamic social network analysis has received a lot of attention, including research on community evolution, e.g., [2], and on graph generation models, e.g., [15]. In this paper, we focus on a different aspect. Given two graph instances, we ask what are the pairs of nodes that came closer to each other. This problem is different from the problem of incrementally maintaining shortest path distances in dynamic graphs, e.g., [7, 23]. Here we just want to identify the k pairs of nodes whose shortest path distance has changed the most, without re-estimating the distances for all pairs. Recent work also addresses the problem of monitoring the proximity of nodes in bipartite time-evolving social graphs [21]. The authors propose pre-computing and storing all pair distances for a small number of nodes so as to incrementally update distances and maintain the top-k most closely connected pairs, or the most central nodes. In contrast here, we consider general graphs (non bipartite) and search for the top-k pairs with the largest decrease in their shortest path distance.

Another line of research addresses graph augmentation problems that ask to find a set of edges to add to a graph so as the graph satisfies specific properties including ones involving shortest path distances. In the context of social networks, recent work considers augmentation problems within the context of improving content propagation. The authors of [22] ask which edges to add or delete in a graph so as to improve the information dissemination process and in particular to increase the leading eigenvalue of the adjacency matrix of the graph. Other recent work addresses the problem of recommending edges to add so as to maximize content spread but in addition ensure that the recommendations are also relevant [4]. The authors of [18] study the problem of selecting a k-size subset of the non-existing or ghost edges of a graph such that if they are added to the graph will minimize the average all pairs shortest path distances. The authors of [19] study the same problem as in [18] but the added edges are selected from a given set of candidate edges (e.g., the edges added between two snapshots). Different to this work, we do not assume that we can select edges to decrease the shortest paths, but rather that edges have already been added as part of the evolution of the network, and we want to find the pairs that were most affected.

The work most closely related to ours is that of [14] that studies essentially the same problem. However, their work does not impose a budget constraint on the shortest path computations. Algorithmically, their work focuses on the endpoints of new edges and relies on identifying critical edges that lie in the shortest paths of many pairs. Such notions necessitate the use of betweenness measures [9], which in general are expensive to compute. In comparison, our work introduces the idea of allocating a fixed budget of m shortest path computations, as well as the formalization of good candidate endpoints as the ones belonging to the maximum cover of the top-k converging pairs. The latter can also form the basis for building effective classifiers for the problem. We compare with the approach in [14] in our experiments.

There is a rich literature in using landmarks for shortest path estimation in large graphs, e.g., [20, 23, 25]. Various approaches for selecting landmarks have been proposed, including the ones used here. However, the problem addressed in our work is different from previous work in that we want to estimate shortest path changes rather than actual shortest paths. It is interesting to consider additional methods for selecting landmarks, such as the approach proposed in Orion [25] that maps graphs into a multi-dimensional Euclidean coordinate space, but it is beyond the scope of this work.

A different point of view in the point-to-point distance estimation problem, which bears some similarity with our approach, is considered in [5]. They propose the use of machine learning techniques for predicting the distance between two nodes of a graph. Aiming to answer real-time point-to-point distance queries, they propose to use linear functions that combine vertex-based attributes, such as the closeness centrality with landmark-based attributes, while incorporating different approaches on selecting lower bound and upper bound landmarks.

Finally, there is a large body of research on link prediction (e.g., [16]) that asks to predict which pairs of nodes will be connected in future snapshots. In this work, we are given the future snapshot, and we are interested in the efficient computation of converging pairs of nodes, not predicting direct links.

3. PROBLEM DEFINITION

We consider undirected (weighted) graphs that change over time. Let $G_t = (V_t, E_t)$ denote the graph at time instant t. As is the most common case with social networks, we consider only node and edge insertions. Thus, a dynamic graph can be seen as a sequence of slices $S_1, S_2, \dots S_t, \dots,$ of node and edge insertions. $G_t = (V_t, E_t)$ is the graph that results by aggregating all slices up until t.

For two nodes u, v, and time instant t, we use $d_t(u, v)$ to denote their shortest path distance in G_t. We call a pair of nodes $u, v \in V_t$, connected if u, v belong to the same connected component of G_t.

Now assume two graph instances, $G_{t_1} = (V_{t_1}, E_{t_1})$ and $G_{t_2} = (V_{t_2}, E_{t_2})$, with $t_2 > t_1$. Take two connected nodes u, v in G_{t_1} that are at distance $d_{t_1}(u, v)$. When considering the graph G_{t_2} the addition of new nodes and edges can only decrease the distance between u, and v. Let $\Delta_{t_1,t_2}(u, v) = d_{t_1}(u, v) - d_{t_2}(u, v)$ denote the decrease in distance between u and v. We are interested in identifying the pairs of connected nodes for which we have a sharp decrease in distance, that is $\Delta(u, v)$ is large. We focus on connected nodes since the distance between non-connected nodes is infinite, and thus, in this case, the problem comes down to finding the disconnected components that got connected.

More formally, we define our problem as follows.

Problem 1 (top-k converging pairs). *Given two graph instances G_{t_1} and G_{t_2}, at time instants t_1 and t_2, respectively, $t_2 > t_1$, and a value k, find the k connected pairs u, v of nodes in G_{t_1} with the largest $\Delta_{t_1,t_2}(u, v)$ value among all pairs of connected nodes in G_{t_1}. We call these pairs of nodes the top-k converging pairs.*

There is a simple polynomial-time solution for our problem. We compute the shortest paths of all (connected) pairs of nodes in G_{t_1} and G_{t_2} and we find the ones that have the largest decrease. Regardless of how fast we compute the shortest paths – there are fast algorithms for approximate shortest path computation, e.g., [20] – just outputting the paths for all pairs requires time $O(n^2)$, where n is the number of nodes in G_{t_1}. For networks with millions of nodes this is impractical both in terms of storage and time. We need solutions that scale linearly with the number of nodes in the graph.

To address this issue, we want to reduce the number of nodes for which we need to compute the shortest paths. We view a shortest path computation (SP computation) as a unit of computational cost, and we assume that we have a fixed computational budget that we can use for our task. This is the number m of SP computations we can perform, which is dictated by our resources and our application. We want to retrieve as many of the top-k converging pairs as possible, under the budget constraints.

We will first establish what is the minimum possible number of shortest path computations. Let P denote the set of the top-k converging pairs that we want to compute. Assume that we are given a set C such that for every pair $(u, v) \in P$, either $u \in C$, or $v \in C$. If we compute all shortest paths for the nodes in C in G_{t_1} and G_{t_2}, and we keep the top-k ones with the highest Δ_{t_1,t_2} value, then we obtain again the set P. The space and time requirements for this algorithm is $O(n|C|)$. Note that the size of the set C is at most k, so the complexity of the problem would be linear in the size of the graph. The set C is a *cover* for the set of pairs in P.

To formalize the definition of the set C, given G_{t_1}, G_{t_2} and k, we define a new graph $G_k^p = (V_1, P)$, defined over the nodes V_1 of graph G_1, $V_1 \subseteq V_{t_1}$, such that there is an edge between two nodes u and v in G_k^p, if and only if, (u, v) is a top-k converging pair. The set C described above is a *vertex cover* of the graph G_k^p, since for every edge $(u, v) \in P$ of G_k^p at least one of the endpoints of the edge belongs to C. That is, every edge is *covered* by at least one vertex. Given a vertex cover of the graph G_k^p, we can obtain the set P of the top-k converging pairs efficiently.

Obviously, when tackling Problem 1, we do not have access to graph G_k^p, or its cover, i.e., the set C. Actually, even if we knew graph G_k^p, obtaining a vertex cover C of minimum size is an NP-hard problem. However, using G_k^p, we can now reformulate our problem as follows.

PROBLEM 2 (BUDGETED PATH COVER). *Given two graph instances G_{t_1} and G_{t_2}, at time instants t_1 and t_2 respectively, with $t_2 > t_1$, a value k, and a budget m, find a set of nodes $M \subseteq V_{t_1}$ of size m such that the number of edges of G_k^p covered by M is maximized.*

We note again that we do not have access to the graph G_k^p. Problem 2 is a harder version of Problem 1, where our computational budget is limited. However, the definition of Problem 2 dictates our approach for solving Problem 1. Our goal is to identify a set of candidate nodes M that are likely to be in the cover of G_k^p. Towards this end, we have a fixed computational budget which is defined by the number m of nodes for which we can compute the single-source shortest paths distances in G_{t_1} and G_{t_2}.

Note that even if we had access to the graph G_k^p finding the minimum vertex cover, or the set of m nodes that maximize coverage is an NP-hard problem. However, it is known [24] that the greedy algorithm that each time selects the node that covers the largest number of the uncovered edges provides a solution with a logarithmic approximation ratio, that works well in practice. The greedy algorithm also has an approximation ratio for the *max-coverage* problem, where given a budget of m vertices we want to find the ones that maximize the coverage of edges. In our experiments, when we want to compare against a "good" solution, we use the vertex cover produced by the greedy algorithm. We will often refer to the greedy solution as the cover of the G_k^p graph.

4. ALGORITHMIC TECHNIQUES

We now outline a generic algorithm for finding the top-k converging pairs. As an intermediate step, our algorithm addresses Problem 2, finding a set M of nodes that cover the largest number of the top-k converging pairs. We propose a suite of algorithms for solving Problem 2. In the following, we shall use the term candidate nodes and candidate endpoints interchangeably to denote the nodes in M.

4.1 Finding the Top-k Converging Pairs

In Algorithm 1, we describe a generic algorithm for finding the top-k converging pairs. The algorithm relies on the function COMPUTECANDIDATEENDPOINTS that returns a set M of candidate endpoints of size m. We discuss the candidate generation below. The number of candidate endpoints m is small and thus it is feasible to compute all shortest path distances between M and the nodes in V_{t_1} in graphs G_{t_1} and G_{t_2}. Given these distances, we can compute the $\Delta_{t_1,t_2}(u,v)$ values for the pairs in $M \times V_{t_1}$ and select the top-k pairs with the highest values.

4.2 Candidate Endpoint Generation

We now describe the algorithms for generating candidate endpoints, that is, identifying nodes that best cover the top-k converging pairs, i.e., the edges of G_k^p. Our algorithms take as input the two graph instances, G_{t_1} and G_{t_2}, and the value m, and produce as output a set $M \subset V_{t_1}$ of m nodes. We want to select M such that M covers the largest number of the top-k converging pairs. The size m of M is determined by our resource budget: it is the number of nodes for which we can afford to compute the single-source shortest paths in G_{t_1} and G_{t_2}.

Algorithm 1 Generic top-k Algorithm.

```
Input: Graph snapshots $G_{t_1} = (V_{t_1}, E_{t_1})$, $G_{t_2} = (V_{t_2}, E_{t_2})$,
    $k$, $m$
Output: The set of the top-$k$ converging pairs
$M \leftarrow$ COMPUTECANDIDATEENDPOINTS($G_{t_1}$,$G_{t_2}$, $m$)
$D_1 \leftarrow m \times n$-dimensional array with shortest path distances in $G_{t_1}$ between the nodes in $M$ and $V_{t_1}$.
$D_2 \leftarrow m \times n$-dimensional array with shortest path distances in $G_{t_2}$ between the nodes in $M$ and $V_{t_2}$.
$\Delta_{t_1,t_2} \leftarrow D_1 - D_2$.
return the top-$k$ pairs $(i, j)$ with the highest values in $\Delta_{t_1,t_2}$.
```

We consider the following approaches in selecting the nodes for the set M:

- **Centrality-based:** In this case, we select nodes based on their degree or the change in their degree.
- **Dispersion-based:** In this case, we select nodes that are highly dispersed in the graph G_{t_1}, that is, they are far apart from each other. These nodes are likely to participate in a large number of top-k converging pairs, since they were far apart in the first place.
- **Landmark-based:** In this case, we select a small sample $L \subset V_{t_1}$ of the nodes in G_{t_1} to act as *landmarks*. We select the candidate endpoints based on the changes of their shortest path distances from these landmarks.
- **Hybrid:** In this case, we use again a landmark-based approach, but instead of sampling randomly the set of landmarks L, we use a dispersion-based approach to guide our choice.
- **Classification-based:** In this case, we use features provided from the previous algorithms to build a classifier that predicts whether a node is a good candidate endpoint.
- **The Incidence family of algorithms:** In this case, we consider some of the algorithms in [14] for selecting the candidate endpoints.

We now discuss each of the above approaches in detail.

4.2.1 Centrality-based selection

With this approach, we use the centrality of nodes in the graph G_{t_1} to guide us in the selection of candidate nodes. The motivation is that nodes central in graph G_{t_1} are likely to be part of the shortest paths for many nodes. Thus they seem a reasonable choice as candidate endpoints. Furthermore, we consider also nodes that had a large increase in their centrality, either in absolute terms or relative to the original value.

A simple and easy to compute centrality measure is the node degree. Formally, let $\deg_t(u)$ denote the degree of node u in graph G_t. The algorithm DEGREE ranks nodes in descending order of $\deg_{t_1}(u)$ and selects the top-m nodes. The algorithm DEGDIFF ranks nodes in descending order of $\deg_{t_2}(u) - \deg_{t_1}(u)$ and selects the top-m nodes. Finally, the DEGREL ranks nodes in descending order of $(\deg_{t_2}(u) - \deg_{t_1}(u))/\deg_{t_1}(u)$ and selects the top-m nodes.

4.2.2 Dispersion-based selection

With this approach, we want to select as candidates the m nodes in G_{t_1} that are the furthest apart from each other. That is, we want M to contain the most dispersed set of nodes. We define this in two different ways.

The first approach is to select nodes such that the average distance between all pair of selected nodes is maximized:

$$M = \arg \max_{\substack{S \subseteq V_{t_1} \\ |S|=m}} \frac{1}{\binom{m}{2}} \sum_{v_i, v_j \in S} d_{t_1}(v_i, v_j) \qquad (1)$$

This approach tends to select nodes in the perimeter of the graph.

Alternatively, we can select nodes such that the minimum distance between any two pairs of the selected nodes is maximized:

$$M = \arg \max_{\substack{S \subseteq V_{t_1} \\ |S|=m}} \min_{v_i, v_j \in S} d_{t_1}(v_i, v_j) \qquad (2)$$

This approach tends to select nodes that "cover" the graph.

Even if we had the pairwise distances between all nodes, finding the optimal set of nodes for both cases is an NP-hard problem (for example, see [13] for a recent discussion on this topic). We thus use a greedy algorithm that at each step selects the node that maximizes the dispersion with respect to the nodes selected so far. We refer to the algorithm that maximizes the average distance as MAXAVG and to the algorithm that maximizes the minimum distance as MAXMIN.

4.2.3 Landmark-based selection

With this approach, we make use of a set $L \subset V_{t_1}$ of ℓ nodes that we call *landmarks* and compute the distances of all nodes in G_{t_1} and G_{t_2} from the nodes in L. We select our candidates based on how close they came to the landmarks in graph G_{t_2}.

Specifically, let $L = (w_1, \ldots, w_\ell)$ be an ordered set of landmark nodes of graph G_{t_1}, with $\ell \ll |V_{t_1}|$. We associate with each node $u \in V_{t_1}$, two ℓ-dimensional vectors, $DL_1(u)$ and $DL_2(u)$, where $DL_1(u)[i] = d_{t_1}(u, w_i)$ and $DL_2(u)[i] = d_{t_2}(u, w_i)$. The vector $\Delta L_{t_1,t_2}(u) = DL_1(u) - DL_2(u)$ captures the change in the shortest path distance from u to the landmarks in L.

We now select as candidate endpoints the nodes that came "closest" to the landmarks L in the graph G_{t_2}. To measure the degree of distance change, we use the L_1 and L_∞ norms of the vector $\Delta L_{t_1,t_2}(u)$.

$$\|\Delta L_{t_1,t_2}(u)\|_1 = \sum_{i=1}^{\ell} \Delta L_{t_1,t_2}(u)[i] \qquad (3)$$

$$\|\Delta L_{t_1,t_2}(u)\|_\infty = \max_{i=1}^{\ell} \Delta L_{t_1,t_2}(u)[i] \qquad (4)$$

Table 1: Shortest-path computations for different approaches

Approach	Candidate Generation	top-k Pairs
Degree-based	0	$2m$
Dispersion-based	m	m
Landmark-based	2ℓ	$2m - 2\ell$
Hybrid	2ℓ	$2m - 2\ell$
Classification-based	$3 \cdot 2\ell$	$2m - 3 \cdot 2\ell$

In the SUMDIFF algorithm, we select the m nodes with the largest L_1-norm, while in the MAXDIFF algorithm, we select the m nodes with the largest L_∞-norm. The intuition is that these are nodes that became more central in the graph G_{t_2} and thus are likely to participate in many changed shortest paths. The SUMDIFF algorithm is somewhat related in motivation to the greedy algorithm for finding the minimum vertex cover. The node with the largest L_1-norm value is the node that covers the most of the distance changes in the $(\ell \times n)$-matrix $\Delta L_{t_1,t_2}$.

4.2.4 Hybrid selection

With the hybrid approach, we attempt to get the best of both worlds by combining the dispersion-based approach with the landmark based approach. Specifically, we use the dispersion-based techniques to select the ℓ landmarks, and then apply the landmark based approach.

This hybrid approach is motivated by two factors. First, in order to obtain the landmark distances in the graphs G_{t_1} and G_{t_2}, we need to pay a cost of ℓ shortest path computations in each graph. When selecting a set of random landmarks this cost comes with no payoff since it is unlikely that the randomly selected nodes will cover any of the converging pairs. Using the nodes selected from the dispersion based algorithm guarantees that we will obtain some benefit from the landmark selection. Second, intuitively, it seems that dispersed nodes should work better as landmarks since they cover different parts of the graph, and thus we can better capture the fact that a node came closer to some part of the graph.

Depending on the algorithm that we use for the landmark selection policy, and the norm we use for measuring the change in the distance to the landmarks, we get four different algorithms, namely the MAXAVG-SUMDIFF (MASD), MAXAVG-MAXDIFF (MAMD), MAXMIN-SUMDIFF (MMSD) and MAXMIN-MAXDIFF (MMMD) algorithms.

4.2.5 Classification-based selection

Our goal in the candidate endpoint generation is to identify nodes that are likely to cover the largest number of converging pairs. We can think of most of the previous algorithms as methods for identifying features that characterize good candidates. A natural extension of the above approach is to combine all these features into a single algorithm using a classifier that will try to predict whether a node is a "good" endpoint or not. The benefit of such an approach is that it combines multiple features instead of one, and it automatically finds the appropriate features to use for each dataset without manual inspection. However, we need to allocate resources for training the classifier.

An important question in this setting is how to determine the positive class for the classifier. What do we mean by

Table 2: Dataset Characteristics

Dataset	No of nodes		No of edges		diameter		max Δ_{t_1,t_2}	not-connected
	G_{t_1}	G_{t_2}	G_{t_1}	G_{t_2}	G_{t_1}	G_{t_2}		G_{t_1}
Actors	1,851	1,886	45,584	56,981	5	5	3	0
Internet links	21,835	25,526	83,857	104,824	12	11	6	80
Facebook	4,436	4,734	25,197	31,498	12	11	7	27
DBLP	15,391	17,992	38,866	48,618	17	15	9	3,864

"good" endpoint? For this task we make use of the vertex cover of the graph G_k^p, that we obtain with the greedy algorithm. This is a collection of nodes that concisely cover the changed paths, and thus it is reasonable to use them as the positive class for our classification task. As features, we use features employed by our algorithms such as the degree of the nodes in G_{t_1}, and G_{t_2}, the degree difference, and the relative degree difference, the L_1 and L_∞ norm of the distances to random landmarks and landmarks selected accordingly to the MaxMin and MaxAvg algorithms. We train one such classifier for each dataset as described in the experiments. We refer to this classifier, as the *local* classifier of the dataset.

We also consider a classifier that can operate on *any* dataset. For this classifier, we extend the feature set with features characteristic of the dataset. In particular, we consider the density, and the maximum degree of the two graph snapshots. The resulting classifier, termed *global* classifier, once trained, can be used to generate candidate endpoints for the top-k converging pairs for any two snapshots of any graph.

For the classification, we use logistic regression. The benefit of the logistic regression algorithm is that it outputs a probability for a node to belong to a given class, in our case to the vertex cover. We sort the nodes in decreasing order of this probability and we output the top-m nodes.

4.2.6 *Incidence Algorithm*

To the best of our knowledge, the first paper that addresses the problem of identifying the top converging pairs in evolving social networks, is [14]. In this paper, the nodes that receive new edges in the second timestamp are called *active* and are considered as the most probable nodes to participate in the top converging pairs. The *Incidence Algorithm* is introduced, where after constructing the set of the *active* nodes A, a number of $|A|$ shortest path computations is performed on both instances of the graph, in order to obtain the pairs with the maximum distance difference.

There are two variations of the *Incidence Algorithm.* In the first one, called *Selective Expansion*, the neighbors of the endpoints in A are also considered as candidates. Every neighbor is evaluated according to the number of its *important* edges. For this purpose, the notion of edge *importance* in a social network is introduced, which is an estimate of the edge betweenness centrality, computed using a randomly selected set of shortest path trees. In our experiments, we used the actual edge betweenness centrality, giving an advantage to the *Incidence* algorithm. In this variation, the algorithm proceeds iteratively, inserting to A new neighbors, and executing *Incidence*, until there are no more new pairs discovered.

The second variation of the *Incidence Algorithm* proposes a number of rank strategies for the *active* nodes, in order to choose the top few of them that are likely to participate in the converging pairs. The rank policies are divided to degree-based and betweeness-based. Among other strategies concerning weighted social graphs (which we do not consider in our work), the authors of [14] rank the *active* nodes using four different policies: their degree in G_{t_2}, their degree difference between the two graph instances, the sum of the *importance* of the edges that a node received in G_{t_2} and finally the difference of the last measure among G_{t_1} and G_{t_2}.

4.3 Complexity

Recall that the value m is not only the number of candidate endpoints, but also the computational budget we have in terms of time and space, that determines how many single-source shortest-path computations we can afford to perform on graphs G_{t_1} and G_{t_2}. Therefore, all algorithms perform exactly $2m$ shortest path computations. Table 1 demonstrates how these computations are allocated in the different phases of the algorithm, for different selection policies. The first phase concerns the shortest path computations required for selecting the candidate endpoints. The second phase involves the computation of the single-source shortest-paths for the candidate endpoints, in both snapshots. For the generated pairs, we compute the decrease in their shortest paths, and select the k ones whose distance decreased the most.

Note that the dispersion based methods need to compute shortest paths only in the graph G_{t_1}, in order to select the candidate endpoints. We still need to compute the shortest paths on G_{t_2} though, in order to output the top-k pairs. Also the classifier requires $3 \cdot 2\ell$ shortest path computations for computing the landmarks in three different ways, in order to produce the features.

5. EXPERIMENTAL EVALUATION

In this section, we compare the performance of the various algorithms and classifiers in terms of identifying the actual top-k converging pairs for various values of the available budget m.

5.1 Datasets and Setting

To evaluate the performance of our algorithms, we need to be able to compute the true top-k converging pairs. Therefore, we use datasets of manageable size, for which it is feasible to compute all-pairs shortest paths. We consider the following four real datasets:

- The *Actors* dataset, where the nodes are actors and there is a connection between two film actors if they both appeared in the same movie. The dataset was obtained from the IMDB web site[1], and it spans the years from 1998 to 2010.

[1] http://www.imdb.com

Table 3: Characteristics of the G_k^p graphs (number of nodes and edges) and their maximum vertex cover (number of nodes).

Dataset	$i = 0$			$i = 1$			$i = 2$		
Actors	$\delta = 3$			$\delta = 2$			$\delta = 1$		
	endpoints	pairs	maxcover	endpoints	pairs	maxcover	endpoints	pairs	maxcover
	35	27	10	1,350	4,081	446	1,851	202,899	9
Internet links	$\delta = 6$			$\delta = 5$			$\delta = 4$		
	endpoints	pairs	maxcover	endpoints	pairs	maxcover	endpoints	pairs	maxcover
	28	46	8	382	734	41	9,196	17,896	194
Facebook	$\delta = 7$			$\delta = 6$			$\delta = 5$		
	endpoints	pairs	maxcover	endpoints	pairs	maxcover	endpoints	pairs	maxcover
	4	2	2	44	37	16	409	591	60
DBLP	$\delta = 9$			$\delta = 8$			$\delta = 7$		
	endpoints	pairs	maxcover	endpoints	pairs	maxcover	endpoints	pairs	maxcover
	6	4	2	68	68	12	289	462	46

- The *Internet links* dataset is an undirected graph representing the AS-level connectivity of the Internet [17, 10] (an Autonomous System (AS) represents a single administrative domain on the Internet). Each node in the graph is an AS and an edge between two nodes represents a message that was exchanged between them, using the inter-domain routing protocol.

- The *Facebook* dataset, where nodes are users of Facebook, and edges between two nodes denote friendship. The dataset contains 31,498 connections, which were created sequentially in 31,498 different time points.

- The *DBLP* dataset, where nodes are authors and there is an edge between two authors if they wrote a paper together. The dataset was obtained from the *DBLP* site[2] and includes articles of 14 top conferences in data mining, databases, theory and the WWW from 1983 to 2013.

Each dataset was divided into two snapshots, so that the initial snapshot, G_{t_1}, contains 80 percent of the edges, and the second snapshot, G_{t_2}, contains the entire graph. Table 2 provides a summary of the characteristics of each dataset.

In selecting the values of k on which to evaluate our algorithms, we note that for any given k, there are many ties, that is, there are many pairs with the same shortest path change. This means that there are many different sets of top-k converging pairs and G_k^p graphs. We set k to a value that guarantees a single optimal solution. Specifically, for two graph instances G_{t_1} and G_{t_2}, let Δ be the maximum distance decrease over all connected pairs of nodes in G_{t_1}. We test our algorithms by assigning values to k that correspond to the number of pairs whose distance change is at least equal to δ, where δ takes values Δ, $\Delta - 1$, and $\Delta - 2$. Setting k as above makes the problem harder, since there is a single optimal solution that we must retrieve that includes *all* converging pairs with shortest path distance change at least δ. For smaller values of the given k, our algorithms work even better, since in this case, retrieving any of (many) tying pairs suffices.

In Table 3, we report for each dataset the number of pairs whose path distance change is at least $\delta = \Delta - i$, for $i = 0, 1, 2$, the number of distinct endpoints involved in these pairs, and the size of the maximum cover as computed by the greedy algorithm. For example, for the DBLP dataset, for $\delta = 8$ $(\Delta - 1)$, there are 68 pairs whose distance was reduced by at least 8. These pairs involved 68 distinct nodes, and they can be covered with 12 nodes. When running our algorithms we set $k = 68$, that is, we look for the top-68 converging pairs.

The main parameter of our algorithms is the available budget expressed through parameter m, i.e., the number of candidate endpoints for which we can compute shortest paths. The performance of an algorithm is measured in terms of *coverage*: The percentage of top-k converging pairs that are retrieved by the algorithm. Note that these are pairs with at least one endpoint in the candidate set produced by the algorithm. The goal of the experiments is to study the cost-coverage tradeoff. We want to understand how the coverage grows as we increase the budget, and the maximum coverage we can obtain with a small budget ($m = 100$). For simplicity, we fix the number ℓ of landmarks to 10 for all algorithms. A larger number of landmarks did not improve the performance.

In Table 4 we give an overview of the algorithms we consider, and a quick index for the algorithm names.

5.2 Single Feature Algorithms

In this experiment, we evaluate the performance of the proposed single-feature algorithms (all algorithms, except for the classification-based algorithms). Our evaluation is in terms of coverage of the top-k converging pairs.

We first evaluate the coverage of our algorithms (percentage of top-k converging pairs found) for a fixed budget m = 100 and various values of k. Table 5 reports the coverage of converging pairs with distance change at least δ for various δ, which corresponds to a number of different k values, ranging from $k = 2$ for Facebook and $\delta = 7$ to $k = 202{,}899$ for Actors and $\delta = 1$. The bold entries correspond to the best performing algorithm for a particular input.

As shown, the *centrality-based algorithms* (based on degree) achieve the lowest coverage almost always for all datasets but for *Actors*. The algorithm that orders the candidate endpoints according to their degree in the original graph has actually almost zero coverage for all datasets, indicating that degree is negatively correlated with participation to changed shortest paths. It appears that nodes with very high degree are already central in the graph, and thus most of their shortest paths are already short. Surprisingly the degree difference is also an ineffective feature for selecting good candidates. This can be explained from the prefer-

[2] http://dblp.uni-trier.de/

Table 4: Overview of Candidate Selection Algorithms.

Degree	Selects the m nodes with the largest $\deg_{t_1}(u)$.
DegDiff	Selects the m nodes with the largest $\deg_{t_2}(u) - \deg_{t_1}(u)$.
DegRel	Selects the m nodes with the largest $(\deg_{t_2}(u) - \deg_{t_1}(u))/\deg_{t_1}(u)$.
MaxMin	Selects greedily m nodes in the first snapshot, such that each new node maximizes the minimum distance to the already selected nodes.
MaxAvg	Selects greedily m nodes in the first snapshot, such that each new node maximizes the average distance to the already selected nodes.
SumDiff	Selects the m nodes with the largest sum of distance decreases from a set of random landmarks L.
MaxDiff	selects the m nodes with the largest maximum distance decrease from a set of random landmarks L.
MMSD	MaxMin-SumDiff: Uses MaxMin for landmark selection, and SumDiff for selecting the m nodes.
MMMD	MaxMin-MaxDiff: Uses MaxMin for landmark selection, and MaxDiff for selecting the m nodes.
MASD	MaxAvg-SumDiff: Uses MaxAvg for landmark selection, and SumDiff for selecting the m nodes.
MAMD	MaxAvg-MaxDiff: Uses MaxAvg for landmark selection, and MaxDiff for selecting the m nodes.
IncDeg	Selects the m of the *active* nodes with the largest $\deg_{t_2}(u) - \deg_{t_1}(u)$ [14].
IncBeet	Selects the m of the *active* nodes with the largest increase in the total betweenness of their incident edges [14].

ential attachment principle [1]: nodes with high degree are more likely to obtain new links. We indeed observed strong correlation between degree and degree change. Relative degree difference mitigates the effect of high degree to some extent, and this is why it performs better than degree and degree difference on all datasets. Still it underperforms compared to other algorithms. The poor performance of the centrality-based algorithms indicates that approaches based on the endpoints of the new edges as the one in [14] are less efficient than selecting candidate nodes based on other features.

The exception to the above observations is the *Actors* dataset. For this dataset, DegRel is among the best algorithms. We note that this is a dense dataset where many of the top changed shortest paths are reduced to single edges. In this setting, the addition of new edges to a node has a stronger effect on the shortest path changes.

The *dispersion-based algorithms* are relatively successful in discovering the converging pairs. MaxAvg outperforms MaxMin in almost all cases. MaxAvg gives preference to nodes in the outskirts of the graph while MaxMin tends to select nodes that cover the different clusters in the graph [8]. Peripheral nodes are more likely to come significantly closer to other nodes in the graph. The satisfactory performance of the dispersion-based algorithms and the fact that they do not require knowledge of the second snapshot indicates that dispersion techniques could also be used as predictors of converging pairs.

From the two *landmark-based algorithms*, SumDiff works consistently better than MaxDiff. SumDiff considers nodes that came closer to many landmarks and thus discovers nodes that become central in the new graph. We think of the SumDiff algorithm as a sampling of the distance changes of the nodes in the graph. Nodes with high SumDiff value are likely to have come close to many nodes in the graph. Thus, in some sense, SumDiff is trying to approximate the methodology of the greedy algorithm for finding a vertex cover for G_k^p.

Finally, among the *hybrid algorithms*, the best coverage is attained in most cases by MaxMin-SumDiff (MMSD). The MaxMin-SumDiff algorithm exploits the ability of the MaxMin algorithm to select representative landmarks that cover the initial graph and the ability of SumDiff to select nodes that come closer to all these representative nodes. Note that although in general MaxAvg is a better dispersion algorithm than MaxMin, MaxMin-based landmark selection tends to outperform MaxAvg-based selection. This is reasonable, since, for landmark selection, it is better to select nodes that cover the graph rather than peripheral nodes.

We now study the coverage achieved for different values of the budget m. Figure 1 shows the coverage achieved by the landmark-based algorithms for various values of m for all datasets. In general, the algorithms based on SumDiff converge faster confirming our intuition that they cover many pairs. The plot also demonstrates that landmark-based algorithms waste part of their computational budget in computing shortest path distances for the $l = 10$ random nodes selected as landmarks that are not likely to be endpoints. Thus, these algorithms obtain no coverage for this computation. On the other hand, the hybrid algorithms benefit from the fact that they choose meaningful candidates as landmarks, and as a result the effort for the landmark shortest path computations is not wasted. Also, notice that MASD and MMSD attain 90% coverage for m smaller than 50.

We also look into the set of candidate endpoints generated by our algorithms to see to what extent this set consists of (1) nodes in G_k^p and (2) nodes in the vertex cover of G_k^p produced by the greedy algorithm. We refer to the second set as greedy-cover. For this experiment we use the Facebook dataset and $\delta = 6$ (k = 37). We study the best-performing algorithms for this dataset: the landmark-based and hybrid algorithms. Figure 2(a) reports the percentage of the candidate nodes that belong to G_k^p and Figure 2(b) the percentage that belongs to greedy-cover for various values of m. Similar behavior is noticed for the other datasets and algorithms. Not surprisingly, we observe that algorithms that cover many paths also have high intersection with both sets. It is interesting to note that the algorithms based on SumDiff have the largest intersection with the greedy-cover, that is, they discover high-quality candidate nodes.

5.3 Classification-based methods

In our experiments, we observed that different algorithms perform well for different datasets. A natural question is whether we can combine the individual algorithms to generate better candidate endpoints and if so, how we should weight the contribution of each individual algorithm. To this end, we view the above algorithms as features and use them

Table 5: Coverage: percentage of converging pairs found for $m = 100$.

	Actors			*Internet links*			*Facebook*			*DBLP*		
	$\delta = 3$ $k = 27$	$\delta = 2$ $k = 4{,}081$	$\delta = 1$ $k = 202{,}899$	$\delta = 6$ $k = 46$	$\delta = 5$ $k = 734$	$\delta = 4$ $k = 17{,}896$	$\delta = 7$ $k = 2$	$\delta = 6$ $k = 37$	$\delta = 5$ $k = 591$	$\delta = 9$ $k = 4$	$\delta = 8$ $k = 68$	$\delta = 7$ $k = 462$
Degree	0	3.99	5.10	0	0	0.19	0	0	0.51	0	0	0
DegDiff	37.04	37.03	22.46	0	0.27	1	0	5.41	6.77	0	7.3	5.2
DegRel	**100**	**66.16**	**34.24**	41.30	44.82	45.17	0	70.27	44.50	**100**	57.35	39.61
MaxMin	59.26	28.13	14.0	67.39	65.67	66.12	**100**	64.87	50.42	75	52.94	42.42
MaxAvg	**100**	43.23	17.03	86.96	81.88	78.65	50	86.49	**93.40**	75	58.82	47.62
SumDiff	74.07	57.59	28.34	96.96	95.05	95.07	55	93.24	93.37	75	84.12	69.83
MaxDiff	83.33	31.74	16.97	92.39	92.00	88.31	45	84.32	87.73	77.5	74,41	63.31
MMSD	96.30	56.26	27.14	**97.83**	95.1	**96.0**	50	**94.56**	90.86	75	79.41	62.34
MMMD	37.04	25.41	15.06	**97.83**	88.69	80.16	50	83.78	80.20	75	72.06	67.53
MASD	**100**	54.64	27.11	**97.83**	**95.1**	95.60	50	89.19	92.22	75	86.77	67.53
MAMD	44.44	26.32	15.06	95.65	94.41	82.09	0	89.19	81.56	75	**88.24**	**76.19**
IncDeg	37.04	37.44	22.77	0	0	0.32	0	2.7	3.9	75	45.6	32.47
IncBeet	29.63	27	15.34	0	0.41	0.73	0	16.22	8.63	0	2.94	1.73

to build a classifier to predict whether a node is a "good" candidate endpoint or not. More precisely we use the following features: the degree of the nodes in the first snapshot; the degree difference; the degree relative difference; the L_1 and L_∞ norm of the distances to random landmarks and landmarks selected accordingly to the MaxMin and the MaxAvg algorithms. All features are normalized in the interval [-1,1]. An important benefit of using a classifier is that it automatically finds the appropriate features for each dataset.

For the positive class of the classifier, that is for the class that corresponds to "good" endpoints, we use the vertex cover computed by the greedy algorithm for the graph G_k^p. As shown by our experiments, participation of a node in the vertex cover is a strong indication that a node is a good candidate. We also experimented with using all endpoints in G_k^p, and the results were very similar.

We use the *LIBLINEAR* implementation of the logistic regression classifier [11]. The logistic regression classifier outputs a probability for every node to belong to the positive class. Using this probability, we order the nodes according to their likelihood of being good endpoints. We sort the nodes in descending order of this probability and we output the top-m nodes.

For the evaluation of the classifier, we split each dataset into four snapshots. To train the classifier, we use snapshots $G_{t_1}^0$ that includes 40 percent of the edges and $G_{t_2}^0$ that includes 60 percent of the edges. To test the classifier, we use snapshots G_{t_1} that includes 80 percent of the edges and G_{t_2} that includes the full graph, that is, the same snapshots used for the evaluation of the single-feature algorithms. This allows for a direct comparison of the results. We used the same δ for both training and testing.

We consider two types of classifiers. A *local* classifier build for each dataset which we denote as L-Classifier, and a *global* classifier that can work for any dataset which we denote as G-Classifier. For the global classifier, we extract additional features from all datasets. In particular, we compute the density and the maximum node degrees of the training graph snapshots that we normalize appropriately within the interval [-1, 1]. We create the global classification model using training data from all four datasets in equal proportions.

Figure 3 compares the coverage achieved by L-Classifier and G-Classifier, with the best algorithm for each dataset. Note that the best algorithm is different for each dataset. The classification algorithm is able to automatically detect the appropriate features and produce a solution that has high coverage for each dataset. Note that the classifiers are handicapped by the set-up cost of the landmark computation (the first 30 shortest-path computations). Still, they are able to catch up with the best algorithm.

The only case where this does not happen is for the *Actors* dataset, when using G-Classifier. The poor performance can be explained by the differentiation of *Actors* dataset compared to the other datasets (Section 5.2). Since 75% of the training set for G-Classifier consists of features extracted from the *Facebook*, *Internet links*, and *DBLP* datasets, and only 25% of *Actors* features, the G-Classifier fails to produce a high coverage.

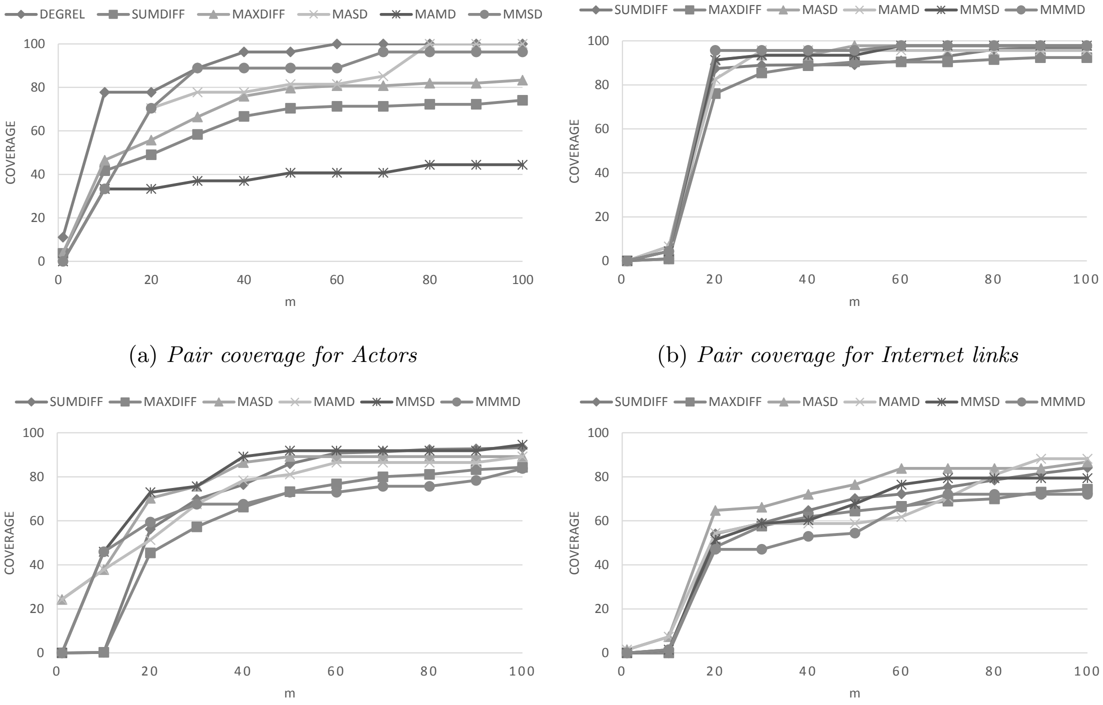
5.4 The Incidence algorithms

Finally, we compare our work to the approach in [14]. For this comparison, we implemented the original version of the Incidence algorithm, which does not use any kind of budget in the shortest path computations. This algorithm achieves very high coverage as shown in Table 6. However the set of the *active* nodes A, which is the set of nodes for which we need to compute the shortest paths, is a large fraction of the original graph. The smallest set A is computed for the *DBLP* dataset, and it is 11.66% of the G_{t_1} size (m = 1,794). In comparison our budget (m = 100) does not exceed 0.65% of the graph size. The largest set A is computed for the *Facebook* dataset, and it is around 66% of G_{t_1}, while our budget of m = 100 candidates is just 2.25% of the graph. Given the large number of candidates used by Incidence, the algorithm achieves almost complete coverage. At the same, time the time complexity is excessively high. For efficiency reasons, we did not test the effectiveness of *Selective Expansion*, as it is a recursive process that is very time consuming. It would lead us to use a very large set of candidate nodes and eventually to solve the problem by performing the baseline algorithm (computing all-pairs shortest paths), which is prohibitively expensive.

We also compare with the approach of [14] under budget constraints. Table 5 shows the best of the degree-based policies, IncDeg, where the *active* nodes are ranked by their

Table 6: The percentage of G_{t_1} that the active nodes form and the coverage of *Incidence* Algorithm.

	Actors			*Internet links*			*Facebook*			*DBLP*		
	$\delta = 3$	$\delta = 2$	$\delta = 1$	$\delta = 6$	$\delta = 5$	$\delta = 4$	$\delta = 7$	$\delta = 6$	$\delta = 5$	$\delta = 9$	$\delta = 8$	$\delta = 7$
Incidence active *nodes*	m = *1,197* (64.66%)			m = *5,071* (23.22%)			m = *2,914* (65.7%)			m = *1,794* (11.66%)		
Incidence *coverage*	100	100	99.35	89.13	95.1	93.91	100	100	99.32	100	97.1	90.9



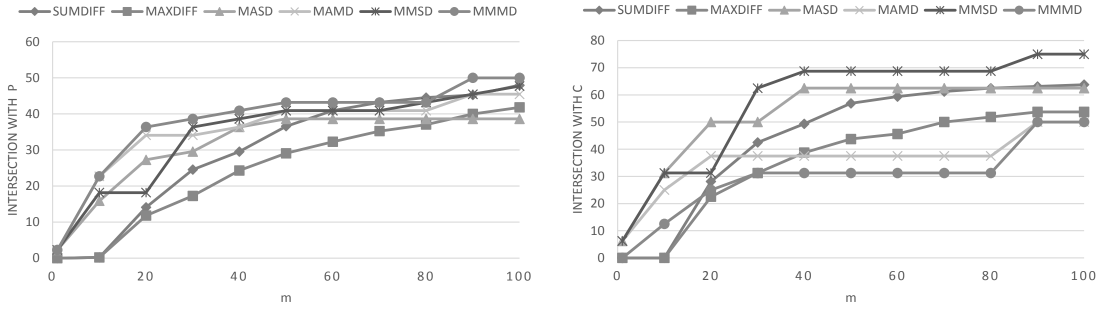
(a) *Pair coverage for Actors*

(b) *Pair coverage for Internet links*

(c) *Pair coverage for Facebook*

(d) *Pair coverage for DBLP*

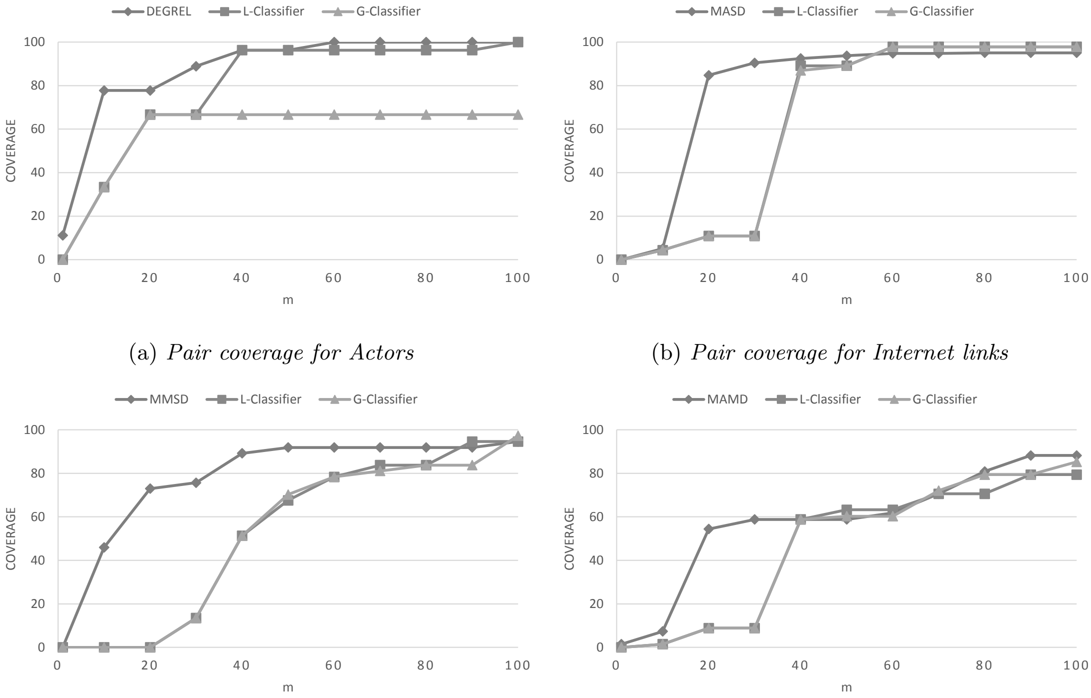
Figure 1: Coverage of the top-k converging pairs, for the (a) *Actors* dataset and $\delta = 3$ ($k = 27$), (b) *Internet links* dataset and $\delta = 6$ ($k = 46$), (c) *Facebook* dataset and $\delta = 6$ ($k = 37$), and (d) *DBLP* dataset and $\delta = 8$ ($k = 68$).



(a) *Intersection with nodes in* G^p_k

(b) *Intersection with greedy-cover*

Figure 2: (a) Intersection of candidate nodes with (a) the nodes of G^p_k and (b) the greedy-cover set, for various values of m for the *Facebook* dataset and $\delta = 6$ ($k = 37$).



(a) *Pair coverage for Actors*

(b) *Pair coverage for Internet links*

(c) *Pair coverage for Facebook*

(d) *Pair coverage for DBLP*

Figure 3: Coverage of the top-k converging pairs, achieved by best algorithm, G-Classifier, L-Classifier trained with the vertex cover computed by the greedy algorithm, for the (a) *Actors* dataset and $\delta = 3$ ($k = 27$), (b) *Internet links* dataset and $\delta = 6$ ($k = 46$), (c) *Facebook* dataset and $\delta = 6$ ($k = 37$), and (d) *DBLP* dataset and $\delta = 8$ ($k = 68$).

degree change and IncBeet, where active nodes are ranked by the increase of the betweeness centrality of the edges incident to the nodes. We compute the actual edge betweenness centrality instead of estimating it, giving an advantage to the IncBeet algorithm. We can observe in Table 5 that IncDeg and IncBet have low performance. In particular, for the *Internet links* dataset, with $\delta = 4$ and $m = 100$ (0.5% of G_{t_1} size), MMSD achieves almost 96% coverage, while the coverage of IncBet does not exceed 1%. In almost all of our experiments, with exception the case of the *DBLP* dataset with $\delta = 9$, both of the two best variations of the Incidence algorithm underperform in term of coverage, as they can not discover more than 50% of the top converging pairs.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we focus on the problem of identifying top-k converging pairs of nodes, that is, pairs of nodes that came closer together between two snapshots of an evolving social graph. We address the problem using purely structural properties of the two graph instances. Since a brute-force method for computing all pair shortest path distances in the two instances is not cost effective, we tackle the problem from a different angle, by predicting the endpoints of such pairs. In doing so, we introduce two novel ideas: (1) allocating a fixed budget of m shortest-path computations and (2) formally defining good candidate endpoints as those belonging to the vertex cover of the top-k converging pairs. We propose a suite of algorithms for selecting candidate nodes and build a classifier that combines them to effectively identify the most appropriate algorithm for each setting. The classifier takes advantage of our novel method of characterizing good candidate endpoints.

For future work, an interesting variation of the problem to consider is the converging pair prediction task, where we are given only the initial graph snapshot and we are asked to "predict" the converging pairs. This problem can be seen as an extension of the link prediction problem which asks whether a single edge will be added between a pair of nodes. Finally, our work can be extended by considering non structural properties, such as additional attributes of the edges or nodes.

7. ACKNOWLEDGMENTS

This work has been supported by the Marie Curie Reintegration Grant project titled JMUGCS which has received research funding from the European Union, as well as the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales which has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds. Investing in knowledge society through the European Social Fund.

8. REFERENCES

[1] Réka Albert and Albert-László Barabási. Emergence of scale in random networks. *Science*, 286:509–512, 1999.

[2] Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. Group formation in

large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[3] Mario Cannataro, Pietro H. Guzzi, and Pierangelo Veltri. Protein-to-protein interactions: Technologies, databases, and algorithms. *ACM Comput. Surv.*, 43(1):1:1–1:36, December 2010.

[4] Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.

[5] M. Christoforaki and T. Suel. Estimating pairwise distances in large graphs. In *BigData*, 2014.

[6] Sara Cohen, Benny Kimelfeld, and Georgia Koutrika. A survey on proximity measures for social networks. In *SeCO Book*, pages 191–206. ACM, 2012.

[7] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. In *STOC*, pages 159–166, 2003.

[8] Marina Drosou and Evaggelia Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.

[9] David A. Easley and Jon M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[10] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[11] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[12] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[13] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[14] Manish Gupta, Charu C. Aggarwal, and Jiawei Han. Finding top-k shortest path distance changes in an evolutionary network. In *SSTD*, pages 130–148, 2011.

[15] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[16] David Liben-Nowell and Jon M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.

[17] Alan Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems*. PhD thesis, Rice University, Department of Computer Science, May 2009.

[18] Manos Papagelis, Francesco Bonchi, and Aristides Gionis. Suggesting ghost edges for a smaller world. In *CIKM*, pages 2305–2308, 2011.

[19] N. Parotisidis, E. Pitoura, and P. Tsaparas. Selecting shortcuts for a smaller world. In *SIAM International Conference on Data Mining (SDM)*, 2015.

[20] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pages 867–876, 2009.

[21] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.

[22] Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.

[23] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *CIKM*, pages 1785–1794, 2011.

[24] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[25] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. Orion: Shortest path estimation for large social graphs. In *WOSN*, 2010.

The CHESS Project: Adaptive Personalized Storytelling Experiences in Museums

Maria Vayanou[1], Manos Karvounis[1], Akrivi Katifori[1], Marialena Kyriakidi[1],
Maria Roussou[1], Yannis Ioannidis[1]

[1] MaDgIK Lab, Department of Informatics and Telecommunications
University of Athens, Panepistimioupolis, Ilissia, 15784 Athens, Greece
{vayanou,manosk,vivi,marilou,mroussou,yannis}@di.uoa.gr

Abstract. In this work, we describe the basic elements of an effort towards achieving personalized storytelling for museum visits in the context of the CHESS project, with a focus on the profiling techniques employed.

Keywords: Profiling, Adaptive Storytelling, User Modeling, Personalization

1 Introduction

CHESS (Cultural Heritage Experiences through Socio-personal interactions and Storytelling) is a research prototype that was developed under the CHESS project (http://www.chessexperience.eu/), aiming to enrich museum visits through personalized interactive *storytelling*. It uses a) personalized information about cultural artefacts to create customized stories that guide individuals or groups through a museum and b) aspires to (re-)inject the sense of discovery and wonder in the visitor experience. The CHESS system employs mixed reality and pervasive games techniques, ranging from narrations to augmented reality on smart phones. Two museums participated in the effort, each with a different scope and end user requirements: the Acropolis Museum in Greece, and the Cité de l'Espace in France.

There are two types of CHESS users, namely the visitors, who “consume” CHESS experience through their web or mobile terminals, and the authors, who design the experiences. CHESS is following a hybrid, plot-based approach with pre-defined content, where *story authors* (curators, museum staff, script writers) write stories around pre-selected museum themes.

2 Authoring CHESS Stories and Storytelling Model

Similarly to the making of a movie, the creation of CHESS stories includes four main phases, namely scripting, staging, producing and editing. During scripting, the author chooses the main story concepts, sketches the plot and writes the narrative text, i.e., the script. In staging, the author associates parts of the script with exhibits, paths and other spots in the museum environment. Then, a set of multimedia resources are produced for the staged script, including audiovisual materials, games, quizzes, augmented reality applications, referred to as activities. Finally, the author does the montage, selecting and ordering the activities to realize the script. In correspondence to the authoring phases, stories are represented as graphs in three different levels of abstraction, namely, the scripting, the staging, and the editing graphs, defining the succession of their atomic pieces and enabling conditional branching based on a variety of events or/and visitor characteristics, over all the three levels. The three graphs are interlinked, so the combined graph forms the story's Storytelling Graph. The overall CHESS Storytelling Graph (CSG) starts with a branching point which leads to all the CHESS stories authored so far.

All the CSG entities (i.e. their atomic pieces as well as the graph branches) are annotated with author selected features. Several features have been exploited so far, such as the topic, information type (real or fictional), script tone, connection to exhibits, required user role, duration, multimedia type, etc.

3 Personalized and Adaptive CHESS Visitor Experience

A typical CHESS experience starts as soon as the visitor enters the museum. The visitor goes to a specific web location with his tablet to log-in into the CHESS application and fill out a short quiz, to gather initial evidences regarding his/her preferences. Then the Adaptive Storytelling Engine (ASTE) starts traversing the CSG graph. Whenever a branching point is met, the ASTE: i) evaluates any hard constraints on each branch to identify the valid ones (e.g. a branch may be unavailable for children) and then ii) estimates the visitor's interest to rank them accordingly.

Aiming to balance the mental load created to the visitor by the presentation of numerous questions and the fully automated decision making, branching points are annotated by the authors as *mandatory*, *automatic* or *optional*. When a mandatory branching point is reached, the ASTE generates a menu where the available options are ranked according to the visitor's profile, while highlighting the first one. In automatic ones, the ASTE makes a decision without informing the visitor about the options. Finally, when an optional one is reached, the ASTE decides whether a menu will be displayed or an automatic decision will take place. An automatic decision is taken if (a) there exists one option significantly better than the rest for the current visitor, and choosing it will not omit other parts that the visitor may also like, or, (b) there exists only one option that the visitor will most probably like and the rest will most probably be disliked.

Depending on the visitor's choice or the ASTE's decision, the CSG is traversed accordingly and the appropriate multimedia resources are fetched and presented to the visitor's terminal.

4 Matching CHESS Visitors to CHESS Content

The visitor's profile contains information about past actions and demographic data, as well as his preferences over the objects he has interacted with (i.e. the CSG entities), referred to as *Interaction Objects*. Visitor's actions are interpreted and a preference value in [-1,1] is extracted. To estimate the visitor's interest in a set of *Candidate Objects*, we use the well-known k-nearest neighbor recommendation algorithm [1]. First, we calculate the similarity of each Candidate Object to each Interaction Object in the visitor's profile and for each Candidate Object we keep the k Interaction Objects with the largest similarities. Given the Interaction Objects with the k largest similarities, we calculate the predicted preference value for the corresponding Candidate Object utilizing a weighted average.

In this way, visitor profiling and matching to CSG entities is not closely tied to the actual features' values used by the authors for annotation. Authors are enabled to have an open tag vocabulary, based upon their understanding of the current story; the only requirement is to use the same vocabulary throughout the story. To achieve this we have used PAROS [2], a system that builds and maintains user profiles following a generic, graph-based user modeling framework.

5 Profile Initialization and Story Selection

To support the authoring process and address the personalization cold start problem, CHESS utilizes the notion of *personas*, a design tool from the marketing world. The system leverages persona definition to match visitors to personas, essentially aligning visitor preferences to the author's understanding of the museum visitors [3]. This approach has been applied in an evaluation study that took place in the Cité de l'Espace and it is described in [4]. We have also explored an alternative approach, interpreting visitors' answers to the initial quiz as evidences about their likes in fictional story parts, which are appropriately annotated. Initial story selection is then conducted by matching the visitor's profile to the corresponding stories' annotations. Results from a recent evaluation study with 24 participants in the Acropolis Museum showed that the adopted approach reached approximately 82% of correct decisions.

6 Profile Update under the CHESS Experience

The CHESS Profiler monitors the visitor's behaviour, interprets it as negative or positive feedback, adjusts the visitor's profile and uses the updated profile in the rest of the experience. The exploited visitor actions are the following: skipping (as high negative feedback on the corresponding activity), completion of activity (low positive feedback on the activity), menu selections (high positive feedback on the script branch) and non-selections in menus (low negative feedback on the corresponding script branches). Based on the visitor's actions on all the activities that comprise a script unit, the Profiler estimates the visitor's preference on the current script unit. Moving a step further, the script unit preferences are used together to infer the visitor's preferences on the whole script branch.

However, due to the big amount and diversity of entities included in the visitor's experience, which may last from 15 minutes to 1 hour, implicit feedback on its own may lead to inaccurate conclusions. Accounting for the main issues observed during the evaluation studies, we have also implemented a conditional explicit feedback approach. A feedback dialogue is dynamically injected into the user's experience through menus, when certain conditions are met, aiming to increase profiling accuracy.

For instance, if the visitor skips many activities in a short time then a menu is shown asking if he disliked the story part, if he is getting tired, or if he already knew the upcoming story part. Similarly, if the visitor skips a part that his/her profile indicates he will like, a menu is shown asking if he is getting tired, if he disliked only the parts of the story he skipped but liked the rest, or if he disliked the whole story part. The main strength of this approach is that explicit feedback is requested only when needed, thus minimizing story interrupts and feedback overhead, while feedback requests occur as the system's response to the visitor's actions.

To further increase the accuracy of implicit feedback, we have also leveraged the visitor's location and movements, the time he spends at each story part, and the way he holds the tablet, examining the visitor's viewing angle. Due to the lack of precise location tracking these techniques have been implemented as a proof of concept and they are showcased with a Javascript demonstration running in the tablet.

7 Conclusions

The provision of personalized content for storytelling experiences in museums entails several profiling challenges. Explicit feedback needs to be maintained minimal to avoid fragmenting the story's plot. At the same time, visitor actions are the result of a guided, complicated interaction with the story's content, the terminal, the museum's space and exhibits, thus requiring for sophisticated and precise visitor monitoring techniques to detect visitor divergence and increase the accuracy of implicit profiling.

Acknowledgements. This research has been partially co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

1. Ricci, F., Rokach, L., Shapira, B., Kantor, P.:Recommender systems handbook, Springer 2011, isbn 978-0-387-85819-7
2. Ioannidis, Y.E., Vayanou, M., Georgiou, T., Iatropoulou, K., Karvounis, M., Katifori, V., Kyriakidi, M., Manola, N. Mouzakidis, A., Stamatogiannakis, L., Triantafyllidi, M.L.: Profiling Attitudes for Personalized Information Provision. IEEE Data Eng. Bull. 34(2): 35-40 (2011)
3. Roussou, M., Katifori, A., Pujol, L., Vayanou, M., Egglestone, S.R: A life of their own: museum visitor personas penetrating the design lifecycle of a mobile experience. CHI Extended Abstracts 2013: 547-552
4. Vayanou, M., Karvounis, M., Kyriakidi, M., Katifori, A., Manola, N., Roussou, M., & Ioannidis, Y.:Towards Personalized Storytelling for Museum Visits. 6th Int Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (2012)