Profiling Users in a 3G Network Using Hourglass Co-Clustering

Ram Keralapura
Narus Inc.
570 Maude Ct, Sunnyvale, CA, USA
rkeralapura@narus.com

Antonio Nucci
Narus Inc.
570 Maude Ct, Sunnyvale, CA, USA
anucci@narus.com

Zhi-Li Zhang
University of Minnesota
Minneapolis, MN 55455, USA
zhzhang@cs.umn.edu

Lixin Gao
University of Massachusetts
Amherst, MA 01003, USA
lgao@ecs.umass.edu

ABSTRACT

With widespread popularity of smart phones, more and more users are accessing the Internet on the go. Understanding *mobile* user browsing behavior is of great significance for several reasons. For example, it can help cellular (data) service providers (CSPs) to improve service performance, thus increasing user satisfaction. It can also provide valuable insights about how to enhance mobile user experience by providing dynamic content personalization and recommendation, or location-aware services.

In this paper, we try to understand mobile user browsing behavior by investigating whether there exists distinct "behavior patterns" among mobile users. Our study is based on real mobile network data collected from a large 3G CSP in North America. We formulate this user behavior profiling problem as a *co-clustering* problem, i.e., we group both users (who share similar browsing behavior), and browsing profiles (of like-minded users) *simultaneously*. We propose and develop a scalable co-clustering methodology, *Phantom*, using a novel *hourglass* model. The proposed *hourglass* model first reduces the dimensions of the input data and performs *divisive hierarchical* co-clustering on the lower dimensional data; it then carries out an expansion step that restores the original dimensions. Applying *Phantom* to the mobile network data, we find that there exists a number of *prevalent* and *distinct* behavior patterns that persist over time, suggesting that user browsing behavior in 3G cellular networks can be captured using a *small* number of co-clusters. For instance, behavior of most users can be classified as either *homogeneous* (users with very limited set of browsing interests) or *heterogeneous* (users with very diverse browsing interests), and such behavior profiles do not change significantly at either short (30-min) or long (6 hour) time scales.

1. INTRODUCTION

Understanding user profiles (i.e., the web browsing behavior of users) is critical to cellular service providers (CSP) for implementing dynamic content recommendation systems and web personalization. These content recommendation systems help CSPs to predict user behavior, and thus recommend links that users are most likely to be interested in. The accuracy of such recommendation systems helps in providing a personalized web browsing experience for users that not only increases the satisfaction of existing users but will also attract new users. To thrive in today's market, a CSP is expected to understand its users' needs better than its competitors to offer appropriate location-based services.

In this paper, we try to understand the browsing behavior of mobile users by investigating whether there exists distinct "behavior patterns". We are interested in understanding how similar or different their behavior is, and studying this behavior by grouping users and browsing profiles into meaningful clusters. We conduct this study using real mobile network data collected from a large 3G CSP in America.

The user profiling problem has been addressed in the past as a *clustering* problem where the objective was to group users who exhibit similar behavior [3]. Grouping entities of similar kind (like "users" in user profiling) based on a chosen similarity metric is termed as *one-way* clustering. However, in this work, we argue that the user profiling problem should be treated as a *two-way* clustering problem with an objective to cluster both *users* and *browsing profiles* simultaneously. This will capture the grouping among users induced by the browsing profiles, and the grouping among browsing profiles induced by users simultaneously. Such a grouping is essential since both users and their browsing profiles are influenced by each other and are not independent. This two-way clustering is called *co-clustering* or *bi-clustering*.

Another way to think about this is in terms of a *matrix*. The input for user profiling can be viewed as a matrix with users on the rows and websites (or URLs) on the columns, and each entry in the matrix representing the number of times the user visited the website. A set of websites constitutes a browsing profile that determines the interests of one or more users. The goal of user profiling is not just to group rows or columns that are similar to each other, but to find sub-matrices that exhibit certain cohesive properties (we explain this in detail later). These sub-matrices group both rows and columns of the matrix in such a way that the users and websites in each group will exhibit a high association with each other while they exhibit low associations with other groups. These sub-matrices are termed as *co-clusters*.

Compared to clustering, co-clustering techniques are relatively new. Few approaches like coupled two-way clustering (CTWC) [13], iterative signature clustering [4], statistical algorithmic method [17], fully automatic cross-associations [5], METIS [1], GRACLUS [7], and spectral co-clustering [8, 9] have been proposed for gene expression data analysis [6, 18, 16] and text mining. In this work, we focus on spectral co-clustering [8]. Spectral co-clustering techniques show that simple linear algebra (singular value decomposition of matrices) can provide a solution to the real relaxation of the discrete optimization (i.e., co-clustering) problem by considering the input matrix as a bipartite graph with the objective of minimizing the cut used to form the co-clusters. Finding a globally optimal solution to such a graph partitioning problem is NP-complete; however the authors in [8] show that the left and right singular values of the normalized original data matrix can be used to find the optimal solution of the real relaxation of the co-clustering problem.

Although theoretically well studied, existing co-clustering algorithms typically have some/all of the following issues: (i) For an input matrix with m rows and n columns, the time complexity in the order of $m \times n$. In real-life applications, the number of rows and columns of the input data matrix is usually very large. For example, in user profiling we can see several hundred thousand users and URLs (columns). Current algorithms make an implicit assumption that the whole data matrix is held in main memory, since the original data matrix needs to be accessed constantly during the execution of the algorithm. (ii) Existing techniques assume that we know the number of co-clusters that the input data matrix contains. This assumption is often unrealistic for the majority of real-time applications as it is hard to understand the hidden relationships that the input data matrix may contain. (iii) Most of the proposed techniques are designed for "hard" co-clustering of the input data matrix, i.e., a row or a column belongs to one and only one cluster. However, in reality, a given row or column can belong to one or more co-clusters.

While some of the techniques like [13] and [8] have all the above issues, other approaches like [1], [7] and [4] require a-priori knowledge of the precise number of co-clusters, and support only hard co-clustering. In [5], the authors address the first two issues (scalability and a-priori knowledge of the number of clusters) and come up with a completely automated approach for co-clustering. However this approach also has other disadvantages when applied to our current problem: (i) It works only with binary matrices, (ii) A fully automated approach does not give the user any control regarding the quality of the output co-clusters, and (iii) Every row/column always belongs to a single co-cluster and cannot be shared.

To address these limitations, we propose a novel co-clustering algorithm called *Phantom*, designed for large data matrices. *Phantom* leverages the spectral graph partitioning formulation in [8] to alleviate the above limitations. We collect 1-day long trace from a nationwide 3G CSP, and use this data to show how *Phantom* can be effectively used for user profiling. Our contributions are:

• **Hourglass Model:** To make *Phantom* scalable to large-size data matrices, we introduce a coarsening and un-coarsening model that first aggregates several rows and/or columns, performs the coclustering, and then subsequently disaggregates the rows and/or columns to restore the original matrix[1]. In the context of user profiling, Phantom first groups the hundreds of thousands of distinct columns into a few tens of "categories" that are content-wise similar to each other. For example, two distinct URLs such as amazon.com and ebay.com are collapsed into one single category named E-Commerce. The new lower-dimensional matrix is then used to co-cluster users and categories. In the next step, we expand the categories in each co-cluster into URLs and subsequently run *Phantom* again to yield co-clusters of users and URLs. The sequence of operations starting from dimensionality reduction of original matrix, co-clustering of the reduced-size matrix, expansion of each co-cluster, and the subsequent co-clustering of the expanded matrices is referred in this paper as *Hourglass model*. We show that this model can deal with very large matrices.

• **Number of Co-Clusters:** *Phantom* employs a recursive hierarchical partitioning of data to automatically determine the number of co-clusters in the input matrix. It starts with the entire matrix, with all users and URLs, as one single co-cluster. In the second step, this single co-cluster is split into two children co-clusters. If both of the children co-clusters meet a specific optimality criteria, then the split is declared legitimate and each child co-cluster is further processed in the next step; otherwise, the split is deleted and the the parent node is declared as the leaf co-cluster and no further partitioning is executed. At the end of this process, the cardinality of the leaf co-clusters represents the total number of co-clusters found by *Phantom*. We show that *Phantom* is not only able to automatically determine the number of co-clusters in the input matrix, but is also able to obtain co-clusters that have much better "quality" (in terms of the metrics defined later) compared to the existing techniques.

• **Hard and Soft Co-Clustering:** *Phantom* can partition a matrix by operating in one of the two modes: *hard co-clustering* or *soft co-clustering*. In hard co-clustering mode, every row and column belongs to only one leaf co-cluster. However, if the property of a user or URL is such that the entity cannot be grouped into only one co-cluster, then hard co-clustering mode yields incorrect results. To address this, *Phantom* can operate in a *soft* co-clustering mode where rows and columns can be borrowed between multiple co-clusters, thus allowing multiple leaf clusters to contain the same row/column. We show that soft co-clustering will result in better grouping than hard co-clustering.

• **User Profiling:** We collect one day long data trace from one of the largest 3G CSPs in North America. To the best of our knowledge, our current work is the first one to explore such a data trace from a 3G cellular network for profiling users. [2] In this mobile environment, we find that there are both *homogeneous* (i.e., users with very limited set of browsing interests) and *heterogeneous* (users with very diverse browsing interests) co-clusters. The number of users who belong to homogeneous co-clusters are far greater than the number of users in heterogeneous co-clusters. Also, the number of co-clusters required to capture 500K users is around 10 showing that user behavior in a cellular networks can be captured using a few browsing profiles. We comprehensively analyze the user behavior over time and find that user profiles do not change significantly at both a micro timescale (30 min intervals) and a macro timescale (6 hour intervals).

2. USER PROFILING PROBLEM

In this section, we present the user profiling problem formulation, and challenges with the existing solutions.

2.1 Problem Formulation

The input to the user profiling problem is a set of users and websites they visit along with the access frequency. This input can be viewed in two equivalent perspectives: (i) As a **matrix** with users on the rows and websites on the columns, and every entry in the

[1]A coarsening/uncoarsening approach has been used in the multilevel algorithms like [1] and [7] where, unlike our semantic approach, the coarsening/uncoarsening of the graph is based on graph properties.

[2]Although, the authors in [19] also experiment with 3G CSP data, they study whether applications accessed by users are location-dependent and focus on user mobility-based hot-spots.

matrix represents the access frequency. (ii) As a *bipartite graph* with users and websites representing the two sets of vertices, and the access frequencies are the weight of the links from the user nodes to website nodes. We use both these notations equivalently to derive a solution to the problem. The solution in one perspective can be translated into the other perspective using linear algebra and graph theory.

A graph $G = (V, E)$ has a set of vertices, $V = \{1, 2, ..., |V|\}$, and a set of edges, $\{i, j\}$, with edge weight E_{ij}. The adjacency matrix, M, is defined as:

$$M_{ij} = \begin{cases} E_{ij}, & \text{if edge \{i,j\} exists} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Given a partitioning of the vertex set, V, into k subsets, $V_1, ..., V_k$, we define cut as the sum of weights of all links that were cut to form the partitioning. Mathematically, $cut(V_1, V_2, ..., V_k) = \sum_{i<j} cut(V_i, V_j)$ where $cut(V_1, V_2) = \sum_{i \in V_1, j \in V_2} M_{ij}$.

We represent the bipartite graph for user profiling as a triple $G = (U, W, E)$, where $U = \{u_1, u_2, ..., u_m\}$ is the set of user vertices, $W = \{w_1, w_2, ..., w_n\}$ is the set of website (URL) vertices, and E is the set of edges $\{\{u_i, w_j\} : u_i \in U, w_j \in W\}$. Note that there are no edges between users or between websites. An edge signifies an association between a user and a website and the weight on the edge captures the strength of this association. In this paper, we use normalized access frequencies as the edge weights: $E_{ij} = \frac{s_{ij}}{\sum_{j=1,...,m} s_{ij}}$, where s_{ij} is the number of times user u_i accesses the website w_j. Note that $\sum_{j \in 1,...,m} E_{ij} = 1$.

Consider the user-website matrix, D, with m users and n websites. Every element in this matrix, D_{ij}, represents edge weight E_{ij}. The adjacency matrix, M, of the bipartite graph:

$$M = \begin{pmatrix} 0 & D \\ D^T & 0 \end{pmatrix} \tag{2}$$

Note that the vertices in M are now ordered such that the first m vertices represent the users while the last n vertices represent websites.

A basic premise in this work is that there exists a duality between users and websites clustering. In other words, user clustering induces website clustering while website clustering induces user clustering. Now, given a set of disjoint website clusters $c_1^{(w)}, ..., c_k^{(w)}$, the corresponding user clusters $c_1^{(u)}, ..., c_k^{(u)}$ can be determined as follows. A given user u_i belongs to the user cluster c_l^u if its association with the website cluster $c_l^{(w)}$ is greater than its association with any other website cluster.

$$c_l^{(u)} = \{u_i : \sum_{j \in c_l^{(w)}} D_{ij} \geq \sum_{j \in c_h^{(w)}} D_{ij}, \forall h = 1, ..., k\} \tag{3}$$

Similarly for a website w_j,

$$c_l^{(w)} = \{w_j : \sum_{i \in c_l^{(u)}} D_{ij} \geq \sum_{i \in c_h^{(u)}} D_{ij}, \forall h = 1, ..., k\} \tag{4}$$

Observe that the above formulation is recursive in nature since webpage clusters determine user clusters, which in turn determine webpage clusters. In every iteration we find better user and website clusters. Clearly, the "best" user and webpage clustering corresponds to a graph partitioning such that the sum of weights of edges between partitions have the minimum value. This is naturally accomplished when the cut for the partitions is minimized. That is,

$$cut(c_1^{(u)} \bigcup c_1^{(w)}, ..., c_k^{(u)} \bigcup c_k^{(w)}) = \min_{V_1,...,V_k} cut(V_1, ..., V_k) \tag{5}$$

where $V_1, ..., V_k$ is any k-partitioning of the bipartite graph.

2.2 Spectral Graph k-Partitioning

Although there are several heuristics (like KL [15] and FM [11] algorithms) to solve the above k-partitioning problem, the spectral graph heuristic that was introduced in [14, 10, 12] typically results in a better global solution. The spectral graph heuristic proves that the eigenvectors corresponding to the eigenvalues other than the largest eigenvalue of the generalized eigenvalue problem, $Lz = \lambda Wz$, where L is the Laplacian matrix ($L = A - M$, where A is the diagonal degree matrix such that $A_{ii} = \sum_k E_{ik}$) and W is diagonal matrix of vertex weights (explained below), will represent the partition vectors that can be used to determine the membership of the nodes into different clusters by minimizing the following objective function:

$$Q(V_1, V_2, ..., V_k) = \sum_i \frac{cut(V_1, ..., V_k)}{weight(V_i)} \tag{6}$$

The eigenvectors are the real relaxation of the optimal generalized partitioning vectors. W can be defined in several different ways, but the most commonly used definition is $W_{ii} = weight(V_i) = cut(V_1, V_2, .., V_k) + within(V_i)$, where $within(V_i)$ is the sum of the weights of edges with both ends in V_i. Using this in Eqn. 6 leads to the *normalized-cut* objective function.

It is important to note that the normalized-cut objective function accomplishes a very similar objective as our original objective function in Eqn 5. In [8], the authors show that the above spectral k-partitioning algorithm can also be posed in terms of the singular value decomposition of the (normalized) original matrix, D. We refer the reader to [8] for more details, but the main idea is the following:

- Given D, compute $D_n = Y_1^{-1/2} D Y_2^{-1/2}$; Y_1, Y_2 are diagonal matrices; $Y_1(i, i) = \sum_j D_{ij}$ and $Y_2(j, j) = \sum_i D_{ij}$.
- Compute $h = \lceil log_2 k \rceil$ singular vectors of D_n, $u_2, ..., u_{h+1}$ and $v_2, ..., v_{h+1}$. Form, $X^T = [D_1^{-1/2} U D_2^{-1/2} V]$ where $U = [u_2, ..., u_{h+1}]$ and $V = [v_2, ..., v_{h+1}]$.
- Run k-means on the h-dimensional data X to obtain the desired k-partitioning.

In the rest of this paper, we refer to the k-partitioning algorithm as *SpectralGraph-k-Part*.

2.3 Challenges

Although *SpectralGraph-k-Part* algorithm works reasonably well, it has several limitations. In this section, we highlight these limitations. Although we focus on *SpectralGraph-k-Part*, these limitations hold for other co-clustering algorithms as well.

The first limitation comes from the fact that input matrices in real-life are usually very large. For example, the input matrix for user profiling problem in large CSPs can easily have up to several thousand users (rows) and webpages (columns) even when aggregated every hour. If we consider a larger timescale, the size of the input matrix significantly increases. Such high dimensions limit the applicability of the *SpectralGraph-k-Part* algorithm. Furthermore, the *SpectralGraph-k-Part* algorithm implicitly makes the assumption that the entire data matrix is loaded into memory, since the original data matrix needs to be accessed constantly during the execution of the algorithm.

The second limitation is that the number of clusters in to which the input data matrix has to be partitioned should be known a-priori. This assumption is often unrealistic for the majority of applications as it is hard to understand the hidden relationships that the input data may contain. For example, in the user profiling problem, the number of user profiles that exists in the data is unknown, and thus it is impossible to guess the number of clusters ahead of time.

The final limitation is that these algorithms have been designed for *hard co-clustering*, i.e, a row or column in the input matrix always belongs to *one and only one* of the several (one or more) output clusters. However, in reality, in several applications we like to allow a row or column to belong to *one or more* output clusters. We refer to such a co-clustering approach as *soft co-clustering* of the input data matrix. There are three reasons why soft co-clustering is important: (i) Hard co-clustering completely removes the association between users and websites if they belong to different clusters. For example, consider three users u_1, u_2 and u_3 and three websites w_1, w_2 and w_3. Let the website access patterns be as follows: $(0.5 \times w_1, 0.5 \times w_2)$ for users u_1 and u_3, and $(0.1 \times w_1, 0.2 \times w_2, 0.7 \times w_3)$ for user u_2. Using hard co-clustering, we obtain two co-clusters, $c_1 = ((u_1, u_3), (w_1, w_2))$ and $c_2 = ((u_2), (w_3))$. The information about the user u_2 accessing websites w_1 and w_3 for 30% of the time is lost. Since soft co-clustering allows a website to belong to multiple clusters, we can perform clustering with minimal loss of information. Using soft co-clustering for the above example will also result in two clusters, $c_1 = ((u_1, u_3), (w_1, w_2))$ and $c_2 = ((u_2), (w_1, w_2, w_3))$, that clearly reflects the different access behaviors of users. (ii) Hard co-clustering is very sensitive to noisy data, and thus may produce incorrect partitions. In fact, hard co-clustering always finds clusters even in random data. The standard procedure to deal with this is to use robust noise pre-filtering techniques. However, such a filtering mechanism is completely problem dependent. On the other hand, soft co-clustering is more resilient to noisy data because of its ability to differentiate the cluster strength and thus, avoid detecting "random" clusters. (iii) A recursive or iterative co-clustering hard co-clustering algorithm tends to accumulate errors, if any, at each iteration. The impact of these errors become bigger as the algorithm goes through several iterations and there is no easy way to fix them. For example, consider the user profiling problem. If a hard co-clustering algorithm erroneously assigned a set of website to a set of users, this mistake is propagated to its children clusters in the next iterations, leading to cumulative errors.

3. PHANTOM ALGORITHM

We address the above limitations and design *Phantom*, a co-clustering algorithm that leverages the *SpectralGraph-2-Part* algorithm with several modifications. We first present the core ideas of *Phantom* at a high level, and then present the detailed algorithm.

3.1 The Hourglass Model

The input matrix for several real-world co-clustering applications can be very large. Such large matrices pose a computational hazard even with the today's technological advancement. In this context, we propose a novel dimensionality reduction and expansion model called *Hourglass* model. The main idea here is to first reduce the dimensions of the input matrix, perform the co-clustering on a lower dimensional matrix, and then subsequently restore the original matrix to perform co-clustering again to give us the clusters that we needed. For the user profiling problem in large CSPs, the input data matrix, D, can have several thousands (or even millions) of rows and columns. Using our *Hourglass* model, we first reduce the dimensions of D. Note that we can choose to reduce both row and column dimensions or reduce either one of the dimensions. In this work, we choose to reduce only the column (websites) dimension. To accomplish this we first group semantically similar websites into few *categories*. For instance, websites such as *Facebook*, *Twitter*, *MySpace*, *Orkut*, etc. will be grouped together into one single category called *social-networking*. We present more details about the rules we use to build such mapping between websites and categories in Section 3.4.

Consider we have l categories to group our websites. As a consequence, D will be transformed into a lower dimensional matrix, D_r, of size $m \times l$. Next, we perform divisive hierarchical clustering (see Section 3.2) on D_r by iteratively dividing clusters into smaller clusters (while retaining certain cluster characteristics) to yield the final co-clusters of users and categories. This will generate k co-clusters, $C^{(u,l)} = \{c_1^{(u,l)}, c_2^{(u,l)}, ..., c_k^{(u,l)}\}$, where each co-cluster $c_i^{(u,l)}$ groups users (with similar category browsing pattern) and categories (with similar users visiting them). Also, $c_i^{(u,l)} = c_i^{(u)} \bigcup c_i^{(l)}$, $i \in 1, ..., k$, where $c_i^{(u)}$ and $c_i^{(l)}$ represent the set of users and categories that belong to the co-cluster $c_i^{(u,l)}$.

After obtaining k user-category co-clusters, each cluster $c_i^{(u,l)}$ is expanded so that the categories in $c_i^{(l)}$ are mapped back to the original websites. This generates the expanded matrix, $D_e(i)$ of size $|c_i^{(u)}| \times |c_i^{(\widehat{w})}|$, where $(\widehat{w})$ represents the new space obtained when considering the websites associated with the categories in $|c_i^{(l)}|$. Note that the cardinality of users belonging to each of these clusters is usually much smaller than the original number of users in D, i.e., $|c_i^{(u)}| \ll |U|$. Each expanded matrix, $D_e(i)$, $i \in 1, ..., k$, is then used as the input to *Phantom* to generate the final user-website co-clusters that we are interested in.

The main advantage of the *Hourglass* model is that we always process low dimensional matrices, thus making it computationally feasible to process large input matrices in terms of memory requirements and processing ability. Also, the *Hourglass* model can be recursively applied to a given input. We refer this recursive model as *Onion-Peel Hourglass* model. For example, for a input matrix we can apply, say, two data reduction steps, perform the first co-clustering step, then carry out the first expansion step, subsequently perform the second co-clustering step, then the second expansion step, and finally perform the last co-clustering step. Such a model will help when: (i) Domains that lend themselves to multiple levels of grouping. In our user profiling problem, we can use multiple levels of categories (say macro-categories that contain several micro-categories which in turn contains several websites). (ii) We require data reduction in both the row and column dimensions.

A disadvantage of this Hourglass model is that it requires domain specific knowledge to carry out dimension reduction. The Hourglass model cannot be applied if a problem does not lend itself to such dimension reduction step.

3.2 Divisive Hierarchical Co-Clustering

Determining the number of clusters in input data before using any clustering algorithm to process it is an extremely difficult problem. In *Phantom*, we adopt a *divisive hierarchical* approach to automatically determine the ideal number of clusters that exists in the data. This iterative algorithm builds a binary tree with the final clusters left behind as the leaves of this tree. It works as follows.

• **Step 1:** Starts with the entire input matrix as a single cluster. At every iteration, partition each of the clusters obtained in the previous iteration into two child clusters by applying the *SpectralGraph-2-Part* co-clustering algorithm [8].

• **Step 2:** After every co-clustering step, both the child clusters are examined to determine their *goodness*. For this we define a *cluster*

Table 1: Classifying URLs into Categories

Category	Sample Keywords	Category	Sample Keywords	Category	Sample Keywords	Category	Sample Keywords
Dating	dating, harmony, personals single, chemistry, match	**Mail**	mail	**Gaming**	poker, blackjack, game, casino	**Download**	download
Music[3]	song, mp3, audio, music, track, pandora	**Maps**	virtualearth maps	**MMS**	mms	**CDN**	cdn,akamai, cachefly,internap
News	magazine, tribune, news, journal, times	**Photo**	gallery, picture, photo, flickr	**Ringtones**	tones	**Porn**	porn,adult xxx
E-Commerce	amazon, ebay, buy, market, craigslist	**Search**	google, yahoo, msn	**Weather**	weather	**Books**	book
Social netw.	facebook, blog myspace, twitter	**Travel**	vacation, hotel expedia, travel	**Video**	video		

cohesiveness metric that captures how self-contained a the child cluster is. If cluster cohesiveness of both children is greater than a pre-specified threshold (say T), then the partitioning is deemed to be legitimate and the child clusters are accepted. Both the children clusters are considered as parent clusters in the next iteration and the process is repeated.

• **Step 3:** If the *cluster cohesiveness* metric of either of the two children is less than T, then the partitioning is considered to be of low quality and the children are rejected. The parent cluster is declared as a leaf and is not subject to further partitioning.

• **Step 4:** The divisive process above is repeated until no cluster can be partitioned further. At the end, the final clusters will form the *leaves* of the binary tree, and the number of leaves represents the number of clusters c in the input data.

The above approach avoids having a-priori knowledge about the number of clusters, which has been a major limitations of the several algorithms including [8]. Compared to the prior algorithms, the input to our algorithm has changed from the number of clusters, k, to the threshold, T, for cluster cohesiveness. We will show later, why T as a parameter works better in practice than k. We will present the details about the *cluster cohesiveness* metric later in this section.

3.3 Soft Co-Clustering Paradigm

As described in Section 2.3, hard co-clustering of input data may result in poor quality clusters. We deal with this issue by designing a *soft* co-clustering approach that allows borrowing of row/s and/or column/s between two child clusters while partitioning the parent cluster. This approach works with a small modification to Step 3 presented in Section 3.2:

• If cluster cohesiveness of both the children is less than T, then the partitioning is rejected and the parent cluster is declared as a leaf cluster.

• If one the two child clusters has a cluster cohesiveness value that is less than T, while the other child cluster has a value greater than T, then we trigger greedy *borrowing*. The child with a lower cluster cohesiveness value starts borrowing columns, one at a time, from the other child starting from the column that will increase its own cohesiveness value the most. Note that when a column is borrowed from one child by another child, the column will exist in both the children. This column borrowing continues until the cohesiveness of both the children is greater than T. After this borrowing process some of the columns may belong to both child clusters. We then declare the partition as legitimate and accept the child clusters.

3.4 Algorithm

3.4.1 Notations and Definitions

First, we remind the reader about the notations that we use in this section. The input matrix, D (of size $m \times n$) has m users ($U = \{u_1, ..., u_m\}$) and n websites ($W = \{w_1, ..., w_n\}$). Each element, $D_{ij} = s_{ij}/\sum_{j=1,...,m} s_{ij}$ where s_{ij} represents the access frequency of user u_i to website w_j. Furthermore, let $L = \{l_1, ..., l_l\}$ represent the l categories that covers the entire website space such that each website belongs to one and only one category[4]. The intermediate and final outputs of *Phantom* are user-category and user-website co-clusters. We use $C^{(u,l)} = \{c_1^{(u,l)}, ..., c_k^{(u,l)}\}$ to represent the user-category co-clusters, while $C^{(u,w)} = \{c_1^{(w,l)}, ..., c_k^{(w,l)}\}$ for user-website co-clusters. Every user-category co-cluster $c_j^{(u,l)}$ consists of users, $c_j^{(u)} \in U$ and categories, $c_j^{(l)} \in L$. Similarly, every co-cluster $c_j^{(u,w)}$ consists of users, $c_j^{(u)} \in U$ and websites, $c_j^{(w)} \in W$.

```
FUNCTION: PHANTOM($U, W, D, L, T$)
 1: /* Initialization */
 2: $\Delta = \{\}, \theta = 0$
 3: /* Dimensionality Reduction: $D_r$ is of size $m \times l$ with $l \ll n$ */
 4: $D_r = ReduceDim(D)$;
 5: /* Co-Cluster Executed on Low-Dimensionality Matrix $D_r$ */
 6: ($C^{(u,l)}, k$)=KernelPhantom($U, L, D_r, T$);
 7: for $j$=1 to $k$ do
 8:     /* Dimensionality Expansion: $c_j^{(u,l)} \in C^{(u,l)}$ are expanded to consider webpages */
 9:     $c_j^{(u,\widehat{w})} = ExpandDim(c_j^{(u,l)})$
10:     $D_e(j) \leftarrow c_j^{(u,\widehat{w})}$;
11:     /* Co-Cluster Executed on each $D_e(j)$ independently */
12:     ($C^{(u,w)}(j), \theta(j)$)=KernelPhantom($c_j^{(u)}, c_j^{(\widehat{w})}, D_e(j), T$);
13: /* Output Preparation: Construction of set $\Delta$ of cardinality $\theta$ */
14: $\Delta = \bigcup_{j=1,...,k} C_j^{(u,w)}$;
15: $\theta = \sum_{j=1}^{k} \theta(j)$;
16: return ($\Delta, \theta$);
```

Figure 1: Hourglass Model

Using the above notation, we define the *cluster cohesiveness* metric used to evaluate the goodness of a co-cluster. Let $c^{(a,b)}$ be a parent co-cluster, while $\{c_i^{(a,b)}\}$, $i \in 1, ..., q$ be the child co-clusters. Here we assume that we partition the parent co-cluster into q child co-clusters. The *cluster cohesiveness* metric, $\gamma_i^{(a,b)}$, of a child co-cluster $c_i^{(a,b)}$, is the ratio of the sum of the weights of associations between the sets $c_i^{(a)}$ and $c_i^{(b)}$ to the sum of the weights of all the associations of $c_i^{(a)}$. Mathematically,

[4]Note that a website can belongs to multiple categories and Phantom can deal with such a situation. However for ease of exposition, we assume that a website belongs to one and only one category

$$\gamma_i^{(a,b)} = \frac{\sum_{h \in c_i^{(a)}} \sum_{k \in c_i^{(b)}} D_{hk}}{\sum_{h \in c_i^{(a)}} \sum_{k \in C^{(b)}} D_{hk}} \tag{7}$$

where $C^{(b)} = \{c_i^{(b)}\}$, $i \in 1, ..., q$ is the entire space of b for all the sibling co-clusters of $c_i^{(a,b)}$ (including itself). Cluster cohesiveness represents how well the child co-clusters (from the same parent) are separated from each other. The higher the value of this metric, the better is the separation. We use the *cohesiveness threshold* parameter, T, to let the user specify the degree cohesiveness of the co-clusters during the co-clustering process. A high value of T ensures that the leaf co-clusters are very well separated from each other.

3.4.2 Hourglass Algorithm

Figure 1 shows the algorithm for the *Hourglass* model in *Phantom*. First, the algorithm (line 4) reduces the dimensions of the input matrix, D ($m \times n$), to generate D_r ($m \times l$). It maps the large number of websites into a few categories ($l \ll n$). This is accomplished by semantically grouping similar websites into the same category using a simple keyword mining over the website URL. A complete list of the categories and the sample keywords used for these categories are shown in Table 1. Some keywords like google, yahoo, and msn represent portals from where users can access different services (e-mail, IM, search, etc). Hence, in order to distinguish between e-mail, IM, and search we apply all the non-search rules first and use the keywords for search as the last step.

Next, *Phantom* co-clusters D_r using the *KernelPhantom* method (line 6). The input to this method are the two sets of entities it has to co-cluster (user set U and category set L), their associations (D_r), and the cluster cohesiveness threshold T that drives the co-clustering process. The function then returns k co-clusters of users and categories, $C^{(u,l)} = \{c_1^{(u,l)}, ..., c_k^{(u,l)}\}$. Each co-cluster $c_j^{(u,l)}$ is then processed independently. First, *Phantom* builds the expanded matrix, $D_e(j)$, that maps each category in $c_j^{(l)}$ to the websites. Thus, the user-category co-cluster, $c_j^{(u,l)}$, is transformed in to a user-website co-cluster $c_j^{(u,\widehat{w})}$ (line 9). Note that $D_e(j) = c_j^{(u,\widehat{w})}$ (line 10). This matrix, $D_e(j)$, is then processed by the *KernelPhantom* method (line 12). Although this phase increases the number of columns of D_e, these columns only correspond to websites belonging to one co-cluster, and hence its size is much smaller when compared to the D. Each of the co-clusters, $c_j^{(u,\widehat{w})}$, $j = 1, ..., k$ results in more user-website co-clusters. In lines 14-16, all these final co-clusters are aggregated and results are returned.

We refer to the co-clustering of D_r as the outerloop and co-clustering of D_e as the innerloop of *Phantom*.

3.4.3 Hard and Soft Co-Clustering Algorithm

The main goal of *KernelPhantom* (Figure 2) is to find all cohesive co-clusters either in hard or soft co-clustering modes. This method does not require the user to specify the number of co-clusters to be used. The choice of the parameter T will determine the number of co-clusters formed by this method. The larger is the value of T the lesser is the final number of co-clusters and vice-versa.

Figure 2 shows the *KernelPhantom* pseudo-code. It starts with an initialization phase (lines 2-5). The variable S contains the set of parent co-clusters that includes both examined and to-be-examined co-clusters. The index r keeps track of which co-cluster in S is currently being processed. Note that r separates the already examined co-clusters from the to-be-examined co-clusters in S. The set L contains all the leaf co-clusters that output by this algorithm.

```
FUNCTION: KERNELPHANTOM($A, B, F, T$)
1:  /* Initialization */
2:  $S \leftarrow [\{A, B\}]$;
3:  $r \leftarrow 1$;
4:  $L \leftarrow \{\}$;
5:  $stop \leftarrow false$;
6:  while $stop == false$ do
7:      $c^{(a,b)} \leftarrow S[r]$;
8:      $F_r$=ExtractMatrix($F, c^{(a,b)}$);
9:      $\{c_1^{(a,b)}, c_2^{(a,b)}\}$=SpectralGraph-2-Part($F_r, c^{(a,b)}$)
10:     $[\gamma_1^{(a,b)}, \gamma_2^{(a,b)}]$=Cohesion($F_r, c^{(a,b)}, c_1^{(a,b)}, c_2^{(a,b)}$);
11:     if $(\gamma_1^{(a,b)} < T)\&\&(\gamma_2^{(a,b)} < T)$ then
12:         $L \leftarrow L \bigcup \{c^{(a,b)}\}$;
13:     else if $(|c_1^{(b)}| == 0)||(|c_2^{(b)}| == 0)$ then
14:         $L \leftarrow L \bigcup \{c^{(a,b)}\}$;
15:     else if $(|c_1^{(a)}| == 0)||(|c_2^{(a)}| == 0)$ then
16:         $L \leftarrow L \bigcup \{c^{(a,b)}\}$;
17:     else
18:         /* Soft Co-Clustering */
19:         if $SoftCoClustering$ then
20:             for $j$=1 to 2 do
21:                 if $\gamma_j^{(a,b)} < T$ then
22:                     $(c_j^{(a,b)})$=BorrowCol($c_j^{(a,b)}, c^{(a,b)}, T$);
23:             for $j$=1 to 2 do
24:                 if $|c_j^{(a)}| == 1$ or $|c_j^{(b)}| == 1$ then
25:                     $L \leftarrow L \bigcup \{c_j^{(a,b)}\}$;
26:                 else
27:                     $S \leftarrow S.add(c_j^{(a,b)})$;
28:         /* Hard Co-Clustering */
29:         else if $HardCoClustering$ then
30:             if $(\gamma_1^{(a,b)} < T)||(\gamma_2^{(a,b)} < T)$ then
31:                 $L \leftarrow L \bigcup \{c^{(a,b)}\}$;
32:             else
33:                 for $j$=1 to 2 do
34:                     if $|c_j^{(a)}| == 1$ or $|c_j^{(b)}| == 1$ then
35:                         $L \leftarrow L \bigcup \{c_j^{(a,b)}\}$;
36:                     else
37:                         $S \leftarrow S.add(c_j^{(a,b)})$;
38:     $r \leftarrow r + 1$;
39:     if $r > |S|$ then
40:         $stop \leftarrow true$
41: return $(L, |L|)$;
```

Figure 2: Soft and Hard Co-Clustering Algorithm

The main loop of the algorithm (lines 6-34) performs divisive hierarchial co-clustering to construct a binary tree until the variable $stop$ is set to $true$. In every iteration, the variable $c^{(a,b)}$ is initialized to the current parent co-cluster to be processed (line 7). This parent co-cluster is used to extract the corresponding associations matrix, F_r from the original associations matrix, F (line 8). This parent co-cluster, $c^{(a,b)}$, is now partitioned into two child co-clusters, $c_1^{(a,b)}$ and $c_2^{(a,b)}$, using the *SpectralGraph-2-Part* algorithm (line 9). The cluster cohesiveness of the children co-clusters are then computed as per Eqn 7 (line 10). If the cohesiveness metric of both the children is less than T, then the parent node is declared as the leaf (lines 11-12). Similarly, if any of the children have no elements of either one of the two entities (i.e., either no users or no websites/categories), then the parent node is declared as the leaf (lines 13-16). If *KernelPhantom* is set to soft co-clustering mode, then the cohesiveness of each of the two children are examined to see if there is a need to borrow columns. If required, the columns are borrowed from the sibling co-cluster until the cluster cohesiveness is at least equal to T (lines 20-22). In lines 23-27, we examine if any of the children co-clusters contain just one element of an entity (i.e. one user or one website/category). If this is the case, then we will be unable to perform further co-clustering on this co-cluster; thus we declare it as a leaf. Lines 29-37 deal with the hard co-clustering case, which is similar to the soft co-clustering case except that we now declare the parent co-cluster as a leaf if one of the two children have a cluster cohesiveness value less than T. We repeat the above steps until we have processed all possible parent co-clusters in the binary tree (lines 39-40), and return all the leaf co-clusters, L (line (41)).

4. DATA TRACE

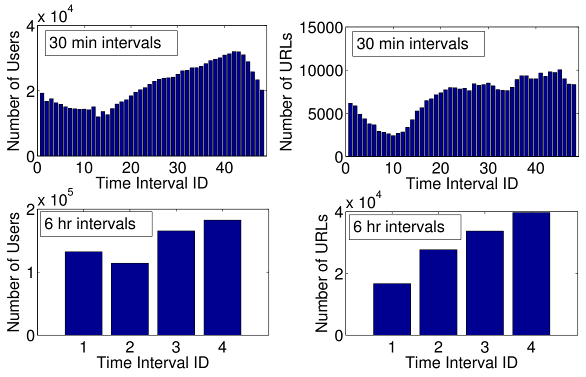


Figure 3: Number of users and webpages in the micro and macro intervals

To address the user profiling problem, we collect an entire day of traces on April 16, 2008 from one of the largest CSP networks in North America. Our data collectors are located in a central location and receives RADIUS [2] authentication and data sessions from the CSP's data network. Note that our collectors do not monitor the CSP's voice network. From the RADIUS authentication sessions we identify the ip-address allocation to a user at the current time. Note that the ip-address allocation for a user can change several times in a day due to mobility, reconnection, etc. We then use this information to identify all the data sessions that belong to a particular user. Finally, after all the processing we obtain all details about a user's browsing activity including the URL of the website and the time when the website was visited. Here we only consider the top-level domain name in the URL.

The data trace collected above contains users in different time zones. The default timestamp in our traces is the time when the collector sees the authentication/data session. However, given that our goal is not only to analyze user behavior at a given point in time, but over the course of a day, we convert all the timestamps to the local time stamps of the users. After this step all the timestamps in our trace are between 0 and 24 hours. To make this point clearer, let us consider two users, one in San Francisco (PST) and the other in New York (EST). We are interested in analyzing the behavior of these two users, say, in the evening between 6pm and 7pm. A session between 6pm and 7pm from New York user will be seen a few hours earlier by our collector than a session from San Francisco user (also between 6pm and 7pm). So we need to consider local timestamps for these two users in their own time zone instead of the collector timestamp.

For our analysis, we split the trace in two ways: 30-min and 6-hour windows. Both of these start at 12 midnight on the day of our collection. We have 24 30-min traces and 4 6-hour traces (Morning (6am-12pm), Afternoon (12pm-6pm), Evening (6pm-12am), and Night (12am-6am)). We refer to our 30-min window analysis as *micro-analysis* and 6-hour window analysis as *macro-analysis*. The reason for the two time windows is to analyze both dynamic (short timescales) and stable (longer timescales) user behavioral trends.

Figure 3 shows the number of users and websites that we see in our 1-day trace for both the time window sizes. We can see that on a average there are 22K users and 7K websites in 30-minute windows. These numbers increase to 150K and 30K for 6-hour windows. Overall, in the entire day we see about 0.5 million users and 150K URLs.

5. EXPERIMENTAL EVALUATION

5.1 Walking through an Example

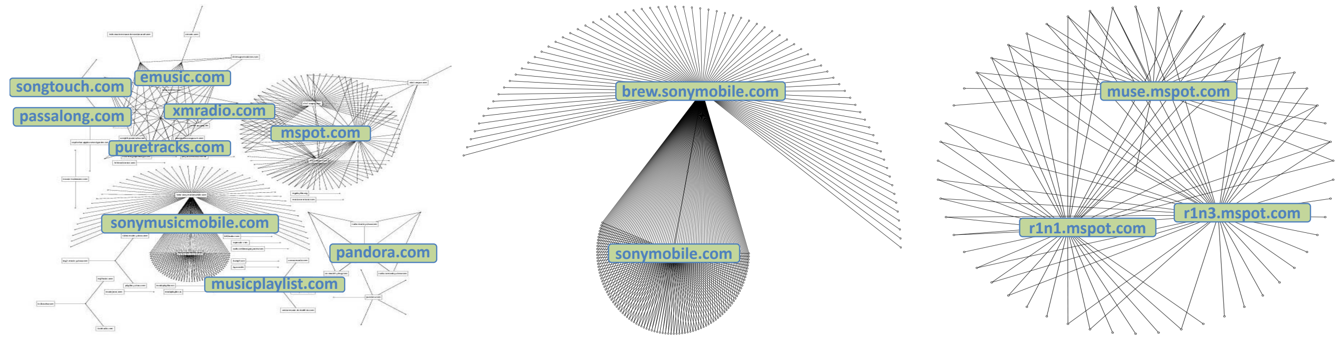
We now consider a 30-minute data set (8:00 am - 8:30 am) described in Section 4. Note that in this 30-minute data, there are around $18,000$ users and $8,000$ urls. Running on a machine with 2.66GHz processor and 4GB of memory size, the Phantom algorithm finished processing (both the outer and inner loops) this data matrix in under 20 seconds.

We now show how the *Phantom* framework can be used to solve the problem. Using the url to category mapping shown before, we map all the urls into 20 categories - 19 categories listed in Table 1 and an "Unknown" category that contains all urls that we could not decisively map to one of the 19 categories. About 9% of the urls get mapped to the unknown category. In the rest of this work, we ignore the urls in the unknown category. Using the url-category mapping we obtain a lower dimensional matrix, D_r, and use this matrix as input to *Phantom*. We set the cohesiveness threshold parameter, T, to be 0.99 to ensure that we get very cohesive clusters while trying to form as many clusters as possible. To keep things simple, we set our algorithm to perform hard co-clustering, i.e., we do not allow borrowing of categories. For this 30-minute trace, *Phantom* resulted in 6 clusters (as shown in Figure 7(b)). Most of the clusters were homogeneous clusters - clusters that just have one single category in them - while one of them was a heterogeneous cluster - a cluster with several different categories grouped together. We will focus on homogeneous clusters in this subsection and dive in to heterogeneous clusters in the next subsection.

Consider the homogeneous cluster with the "Music" category. Fig. 4(a) shows this homogeneous cluster with the category mapped back to its urls. The small circles on the periphery of the figure represents users and the rectangular boxes represents urls. The links between users and urls represent the fact that the user visits the url. We can see that users clearly prefer some websites over others, and hence we have several seemingly disconnected components in this graph. Running the innerloop of *Phantom* co-clusters these users along with the specific websites that are part of their browsing profile. Figures 4(b) and 4(c) show two of the several co-clusters formed by the innerloop of Phantom. Comparing Figures 4(b) and 4(c) with Figure 4(a), we can clearly see that Phantom extracts the dominant components in the graph in Figure 4(a) that represent the group of users and urls that form their browsing profile. Note that this is true even for heterogeneous clusters.

5.2 Refining the Algorithm

If the cohesiveness threshold used for the outerloop of *Phantom* is very high, then the resulting co-clusters are very cohesive, but could be heterogeneous. We refer to co-clusters with more than 5 different categories grouped together as "giant" heterogeneous co-clusters. Cluster-3 in Figure 7(b) (Figure 5) is one such co-cluster with 15 categories. Such co-clusters with a disparate set of cate-



(a) (b) (c)

Figure 4: (a) The homogeneous music cluster expanded into URLs. (b) and (c) Two (of the eight) clusters formed by running the inner-loop of Phantom.

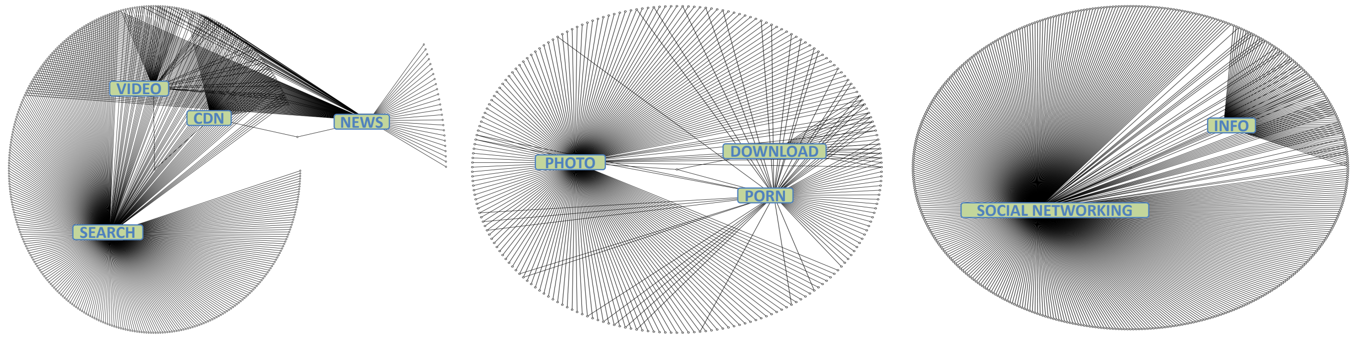


Figure 6: Three sub-clusters of the giant cluster

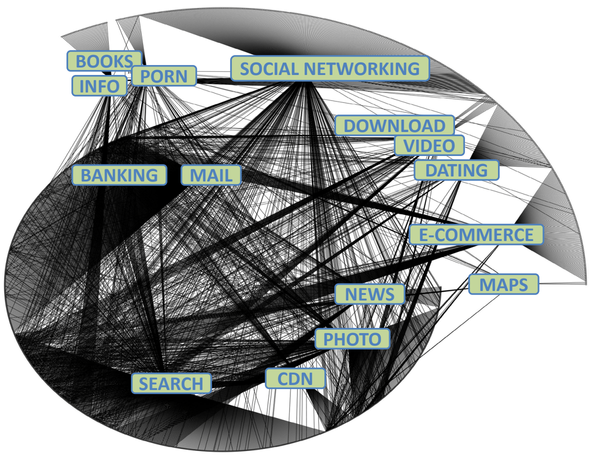


Figure 5: Example of a heterogeneous giant cluster

gories do not accurately represent the browsing profile of individual users that we are interested in. Hence we adopt the following strategy to further split these giant heterogeneous clusters. We first define a *lower limit* and an *increment* for the cohesiveness threshold. If we obtain any giant clusters in the first round of co-clustering in the outerloop of *Phantom*, we decrease the cohesiveness threshold by the increment value and run the outerloop again on the giant cluster. We repeat this process until there are no giant clusters or the cohesiveness threshold value has fallen below the permissible lower limit. In this example, we define the lower limit of the cohesiveness threshold as 0.95, the increment as 0.01, and $T = 0.99$. This ensures that the resulting clusters are still very cohesive, but gives *Phantom* a chance to form more clusters. This iterative step of Phantom resulted in 6 additional co-clusters. Figures 6(a), 6(b) and 6(c) show three of these clusters.

5.3 *Phantom* Vs *SpectralGraph-k-Part*

In this subsection, we compare the results obtained from the *SpectralGraph-k-Part* algorithm with *Phantom*. We use the same 30-minute dataset as in the previous subsection, but only use the user-category matrix since we were unable to run *SpectralGraph-k-Part* on the user-websites matrix.

SpectralGraph-k-Part is a non-hierarchical approach that requires the number of co-clusters to be formed as an input parameter. To determine the ideal number of clusters in the input data, we follow a brute-force approach. We vary the number of clusters to be formed from 1 to 19 (the maximum possible clusters with at least one category in each cluster). For each value of this parameter we compute the average leaf cluster cohesiveness and standard deviation of leaf cluster cohesiveness (see the black curve in Figure 8(b)). We can see, by inspection, that the optimal value of the number of clusters based on the average and standard deviation of the leaf cluster cohesiveness values is 4. If the number of clusters increases beyond this value, then the average leaf cluster cohesiveness considerably decreases (top graph in Figure 8(b)) and its standard deviation considerable increases (bottom graph in Figure 8(b)). However a lower value of the number of clusters does not affect the cohesiveness as much. Hence, we use 4 as the *ideal* number of clusters for *SpectralGraph-k-Part* in the rest of this subsection.

Figure 7 compares the clusters formed by *SpectralGraph-k-Part* with the *Phantom* hard and soft co-clustering algorithms. We can

Spectral Graph-k-Part	Cluster Cohesiveness	Number of Users	Categories
Cluster-1	0.99	10060	Ringtones, Travel, Music
Cluster-2	0.92	545	Weather, Photo, Dating
Cluster-3	0.96	1356	Maps, Download, E-Commerce, Mail, CDN, Video, Gaming, News, Search, Porn, Books, Social Networking
Cluster-4	0.99	6062	MMS

Average Leaf Cluster Cohesiveness = **0.969**
Std Deviation of Leaf Cluster Cohesiveness = **0.036**

(a)

Phantom (Hard)	Cluster Cohesiveness	Number of Users	Categories
Cluster-1	0.99	6064	MMS
Cluster-2	0.92	238	Music
Cluster-3	0.98	1596	Maps, Download, E-Commerce, Mail, CDN, Dating, Video, Gaming, Photo, News, Search, Porn, Books, Social Networking
Cluster-4	0.97	277	Weather
Cluster-5	0.98	9819	Ringtones
Cluster-6	0.90	29	Travel

Average Leaf Cluster Cohesiveness = **0.963**
Std Deviation of Leaf Cluster Cohesiveness = **0.038**

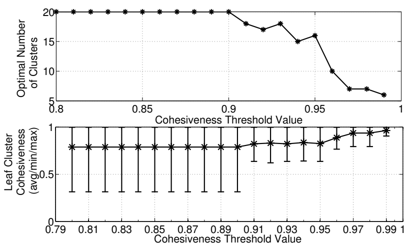
(b)

Phantom (Soft)	Cluster Cohesiveness	Number of Users	Categories
Cluster-1	0.99	6064	MMS
Cluster-2	0.98	1596	Maps, Download, Trading, Mail, CDN, Gaming, Dating, Video, Photo, News, Search, Porn, Books, Social Networking
Cluster-3	0.97	277	Weather
Cluster-4	0.99	238	Ringtones, Music
Cluster-5	0.94	29	Ringtones, Travel
Cluster-6	0.98	9819	Ringtones

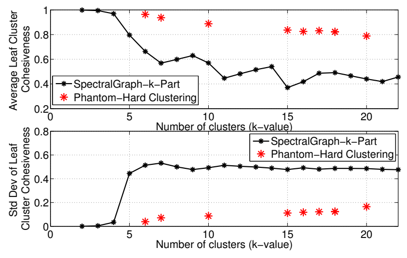
Average Leaf Cluster Cohesiveness = **0.983**
Std Deviation of Leaf Cluster Cohesiveness = **0.018**

(c)

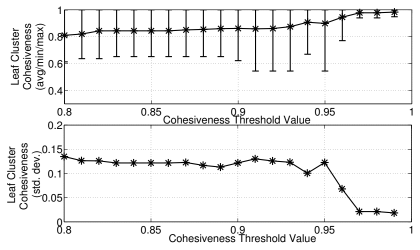
Figure 7: Clustering results from (a) *SpectralGraph-k-Part* Co-Clustering Algorithm (b) Phantom Hard Co-Clustering Algorithm (c) Phantom Soft Co-Clustering Algorithm.



(a)



(b)



(c)

Figure 8: (a)Influence of cohesiveness threshold on Phantom hard co-clustering algorithm in terms of the number of clusters formed and average cohesiveness of the clusters. (b) Phantom compared to the *SpectralGraph-k-Part* algorithm. (c) Influence of cohesiveness threshold on Phantom soft co-clustering algorithm in terms of the average cohesiveness of the clusters and the standard deviation of the cluster cohesiveness.

see that the results from the Phantom-hard algorithm are very similar to that of *SpectralGraph-k-Part* in terms of the average and standard deviation of leaf cluster cohesiveness. However, the Phantom-hard algorithm results in two additional clusters compared to the *SpectralGraph-k-Part* algorithm. One of the key benefits of *Phantom* is that it typically results in *higher* number of co-clusters than the existing techniques without sacrificing the cohesiveness of the resulting co-clusters. Figure 7(c) shows the results for the Phantom-soft algorithm, and we find that it out-performs *SpectralGraph-k-Part* in terms of all the metrics (number of clusters, average and standard deviation of the leaf cluster cohesiveness).

The only parameter that need to be tuned in *Phantom* is the cohesiveness threshold, T. In Figure 8(a) we show the influence of this parameter on the number of clusters that are formed and the leaf cluster cohesiveness. As one can expect, higher values of T results in lower number of *more cohesive* clusters. Depending on the particular application, this parameter can be set to the right value. In this work we use $T = 0.99$. Figure 8(b) compares *SpectralGraph-k-Part* with the Phantom-hard algorithm in terms of leaf cluster cohesiveness for different values of the number of clusters. Note that in *SpectralGraph-k-Part*, the number of clusters is a parameter and hence every value of this parameter results in a value for the leaf cluster cohesiveness. However, for the *Phantom* algorithm, varying the parameter T, results in different values of the number of clusters as shown in the top graph in Figure 8(a). We plot the results from this on the same graph (Figure 8(b)) to compare the two algorithms. We can clearly see that the *Phantom* outperforms *SpectralGraph-k-Part* in terms of the average/standard deviation of leaf cluster cohesiveness. Figure 8(c) shows the same metrics for the Phantom-soft algorithm and we can see that the gain is significantly higher when compared to the Phantom-hard algorithm.

We refer to the set of categories that are grouped together in a single cluster by *Phantom* as a category-level "browsing profile". Intuitively, a category-level browsing profile represents the interests of all users who belong to the same cluster. If we choose a high value of cohesiveness threshold, T, then the browsing profile captures "most" (or a large portion) of the browsing interests of users in the clusters. Choosing a lower threshold implies that *Phantom* will capture a smaller portion of users' interests in each cluster. The number of users in each cluster indicates the "strength" of the browsing profile represented by the categories in that cluster. The higher is the strength of a browsing profile, the more popular is the profile among users. A CSP can use the browsing profiles and their strength to determine the diversity of services that can be offered to customers. For instance, assume that *Phantom* finds a browsing profile with a large number of users that contains both Music and Travel categories. Using this information, the CSP can now offer bundled services that includes both music and travel to all users since users with interest in music are most likely to be interested in travel as well and vice versa..

The set of categories in the categories column in Figure 7 are the category level browsing profiles of users in our 30-min data trace. We split the giant heterogeneous co-cluster using the refinement presented in Section 5.2 to yield 6 more co-clusters. Thus only *11 different browsing profiles are sufficient to capture the browsing behavior of all 18000 users in this 30-min data trace*.

Unlike the *SpectralGraph-k-Part* algorithm, *Phantom* automatically finds the number of clusters in the data based on T. This is a big advantage in several problems (including the current problem of finding browsing profiles of users), since it is much easier to intuitively specify the required cohesion in clusters than the total number of clusters. For example, in the context of our problem, our objective is to find the number of browsing profiles that naturally come out of the dataset rather than force the number of profiles to

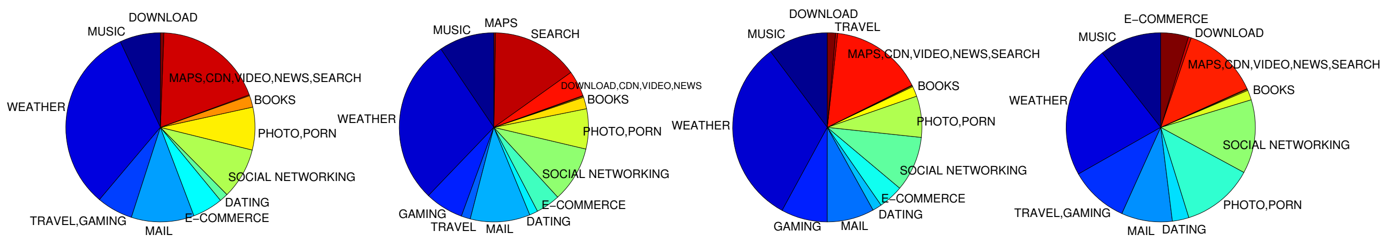
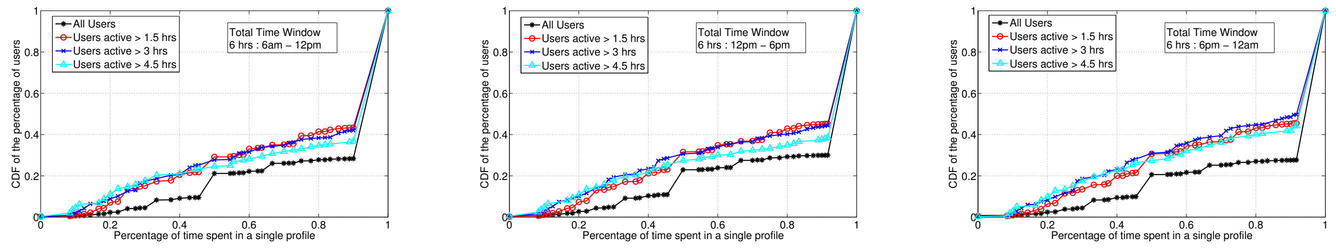


Figure 9: Macro-Profile in 6-hour intervals (a) Morning (b) Afternoon (c) Evening (d) Night)



(a) (b) (c)

Figure 10: Users tend to stick to the same browsing profile at the category level in (a) Morning (b) Afternoon (c) Evening

have a certain value before performing clustering. In summary, T can be used to specify the quality of clusters that are output by the *Phantom* algorithm. If we need clusters that are very cohesive then we can set a high value of T (for example, $T = 0.99$); we can sacrifice the cohesiveness (or quality) of clusters in favor of a larger number of clusters by setting a lower value of T.

5.4 User profiles using Phantom

Our main goal in this section is to gain more insights about the browsing behavior of users with the help of our novel Phantom co-clustering algorithm. We use the Phantom-hard algorithm in this section, but we would like to remind the reader that using the Phantom-soft algorithm will only improve the results that we present here. We use results from hard co-clustering mainly for the ease of exposition of results in this section.

In this work, browsing profile implies profiling all user-initiated browsing activities. Two of the categories that we considered in the previous section, MMS and Ringtones, contain both user-initiated and network-initiated sessions. Network-initiated sessions are those sessions that are pushed by the CSP to users. Examples of activities resulting in such sessions include mobile services, mms about new services, free ringtone availability, etc. Hence, in this section, we ignore these two categories, and consider only those categories where the HTTP sessions are initiated by the user.

Figure 9 shows the macro-profile of users (at 6-hour timescale). The size of the pie represents the number of users that belong to the cluster with the categories noted in the figure. We can see that several clusters in all the four 6-hour intervals have the same categories indicating that irrespective of the time of the day, certain users tend to almost exclusively browse the urls in these categories. For example, Music, Social Networking and Mail are three categories that belong to separate clusters in all intervals, showing that these are strong browsing profiles that remain constant throughout the day. However, some of the categories like news, cdn, and video get grouped into the same cluster in all the intervals. These set of categories also form strong browsing profile. One way to interpret this grouping is that most of the news and video websites that users tend to browse are served by CDN network content caching. Other categories like Search, get clustered differently in different intervals. In the Morning, Evening, and Night, Search gets grouped with other categories like News, Video, Maps, etc., however in the Afternoon this category is grouped by itself. This shows that some users tend use search urls irrespective of the fact that they are browsing into other categories or not. However, in the Afternoon a certain set of users tend to predominantly use Search engines, thus resulting in a separate cluster for itself.

Now that we have seen the various profiles of users in a macro-timescale, the questions that we want to address are the following: (i) Do users exhibit the same profile even at smaller timescales? In other words, does the dynamic behavior of users change considerably from the stable behavior? If yes, then how long do they keep their profiles? (ii) Do users show similar behavior when we consider urls instead of categories (referred to as url-profile)? If yes, then how long do they keep their url-profile?

Figures 10, 11 and 12 try to address the above questions. For each of the macro-profiles, we first consider 30-min intervals that make up the 6-hour interval, and then compute the micro-profile of users in this 30-min intervals. If the micro-profile of a user matches the macro-profile of the user in the 6-hour time interval then we say that the user has the same profile in the 30-minute interval as in the 6-hour interval. We compute such matches for all the twelve 30-min intervals in every 6-hour interval, and show the percentage of users who tend to keep their profile over this time interval in Figure 10. The x-axis in these graphs represents the fraction of time spent in the same profile and the y-axis represents the CDF of the fraction of users. The four different curves in each of these graphs represent users who are active for a different amount of time. For example, the curve for users who are active for more than 3 hours includes all the users who appear in at least six 30-min intervals.

In all of the graphs in Figure 10, and all the curves in these graphs, we can see that there is big spike between values of 0.9 and 1.0 on the x-axis. This spike shows that around 60-70% of the users do not change their profiles (at the category level) in all the time intervals where they are active. In other words, in our 3G traces, music users predominantly stay as music users and social networking users predominantly stay as social networking users for the entire duration that s/he is active. There are less than 20% of users who tend to stay in their profiles for less than 40% of their active time. That is, less than 20% of users exhibit a volatile (i.e., highly changing) browsing behavior. The take away point here is that majority of users stick to their profiles irrespective of the timescale.

Figure 11 is very similar to Figure 10 except that we are now exploring the url-profile of users. Note that the y-axis is shown in log scale. We can see that the pattern of users staying in the same url-profile is even more dominant in Figure 11, than in Figure 10. In other words, a majority of users not only tend to stay in the same category, but they tend to stay with the same set of urls. We further explore if this behavior holds for every co-cluster individually, and find that users still tend to stick to their urls (Figure 12).

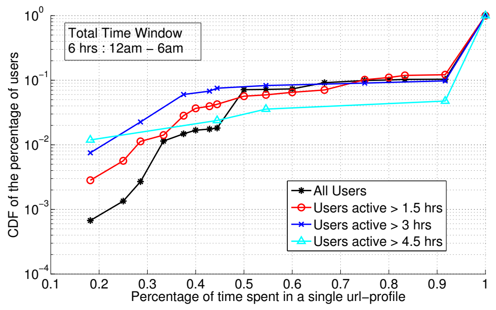


Figure 11: Users tend to stick to the same browsing profile at the URL level. All users in all the clusters.

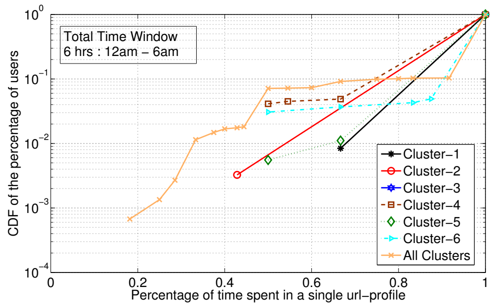


Figure 12: Users tend to stick to the same browsing profile at the URL level. Users in particular clusters.

6. CONCLUSIONS

In this paper we address user profiling in large CSPs. We formulat the problem as a *co-clustering* problem with the goal of grouping both users and websites simultaneously. To overcome the limitations of current techniques, we propose *Phantom*, a novel co-clustering algorithm based on the Hourglass model that reduces the dimensions of the input, carries out the co-clustering before expanding the dimensions again. This model can be applied in *onion-peel* mode where we can have several levels of dimension reduction and expansion. We showed that *Phantom* can deal with high dimensional matrices and produce very cohesive co-clusters.

Phantom is a generic co-clustering framework that can be used to solve any problem that lends itself into the co-clustering formulation. It automatically finds the number of clusters in the data based on a cohesiveness threshold, T, specified by the user. This is a huge advantage, since it is much easier to intuitively specify the required cohesion in clusters than the total number of clusters. In this paper, we have used a constant value of T through out the *Phantom* co-clustering process. However, it may be advantageous to change this value as we build the binary tree. For example, in the hard co-clustering mode errors tend to accumulate as we go down the binary tree. If we increase the value of T with the level in the binary tree, then the error accumulation can be reduced. This is part of our future work.

A disadvantage of the Hourglass model is that we require domain specific knowledge to carry out the dimension reduction. The Hourglass model cannot be applied if a problem does not lend itself to such dimension reduction step.

7. REFERENCES

[1] METIS - Family of Multilevel Partitioning Algorithms. http://glaros.dtc.umn.edu/gkhome/views/metis.

[2] RFC 2865. http://tools.ietf.org/html/rfc2865.

[3] P. Antonellis and C. Makris. XML Filtering Using Dynamic Hierarchical Clustering of User Profiles. *DEXA*, 2008.

[4] S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of largescale gene expression data. *Phys Rev E Stat Nonlinear Soft Matter Physics*, 2003.

[5] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully Automatic Cross-associations. *KDD*, 2004.

[6] Y. Cheng and G. Church. Biclustering of expression data. *Proc. of the Eighth Int. Conf. on Intelligent Systems for Molecular Biology*, 2000.

[7] I. Dhillon, Y. Guan, and B. Kulis. A Fast Kernel-based Multilevel Algorithm for Graph Clustering. *Proceedings of ACM SIGKDD*, 2005.

[8] I. S. Dhillon. Co-clustering documents and words using Bipartite Spectral Graph Partitioning. *KDD*, 2001.

[9] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic coclustering. *KDD*, 2003.

[10] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM R&D Journal*, 1973.

[11] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. *Technical Report 82CRD130, GE Corporate Research*, 1982.

[12] M. Fiedler. Algebraic connectivity of graphs. *Czecheslovak Mathematical Journal*, 1973.

[13] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. *Proceedings of National Academy of Science*, 2000.

[14] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 1970.

[15] B. Kerninghan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system Technical Journal*, 1970.

[16] Y. Kluger, R. Basri, J.T. Chang, and M. Gerstein. Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. *Genome Research, CSH Press*, 2003.

[17] A. Tanay, R. Sharan, M. Kupiec, and R. Shamir. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proceedings of National Academy of Science*, 2004.

[18] A. Tanay, R. Sharan, and R. Shamir. Discovering Statistically Significant BiClusters in Gene Expression Data. *Bioinformatics, Oxford University Press*, 2002.

[19] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Measuring serendipity: Connecting people, locations and interest in a mobile 3G network. *Proc. of ACM Internet Measurement Conference*, 2009.