Mining Social Networks for Significant Friend Groups

Carson Kai-Sang Leung* and Syed K. Tanbeer

Department of Computer Science, University of Manitoba, Canada
kleung@cs.umanitoba.ca

Abstract. The emergence of Web-based communities and hosted services such as social networking sites has facilitated collaboration and knowledge sharing between users. Hence, it has become important to mine this vast pool of data in social networks, which are generally made of users linked by some specific interdependency such as friendship. For any user, some groups of his friends are more significant than others. In this paper, we propose a tree-based algorithm to mine social networks to help these users to distinguish their significant friend groups from all the friends in their social networks.

Keywords: Advanced database applications, social networks, social media, social computing, knowledge discovery, data mining, social network analysis and mining.

1 Introduction

Rapid growth and exponential use of social digital media has led to an increase in popularity of social networks and the emergence of social network mining, which combines data mining with social computing [6,10,17,20]. As *social networks* [3,15] are generally made of social entities (e.g., individuals, corporations, collective social units, or organizations) that are linked by some specific types of interdependency such as friendship, a social entity can be connected to another entity as his friend. Similarly, a social entity can be linked to another entity as his next-of-kin, friend, collaborator, co-author, classmate, co-worker, team member, and/or business partner via other interdependency such as kinship, friendship, common interest, beliefs, and financial exchange.

Social network mining [2,7,8,12] aims to discover implicit, previously unknown, and potentially useful knowledge from a vast pool of data residing in the social networking sites such as Facebook [4,14], Twitter [13,16,19], and LinkedIn. In LinkedIn, a user can create a professional profile, add connections to other users (as friends, colleagues, and/or classmates), and exchange messages. In addition, he can also join common-interest groups and participate in discussions. Although the number of friends/connections may vary from one user to another, it is not uncommon for a user p to have hundreds of friends/connections. Among

* Corresponding author.

H. Yu et al. (Eds.): DASFAA Workshops 2012, LNCS 7240, pp. 180–192, 2012.

them, some groups of friends are more important or significant to p than others. Significance of a friend group G depends on various measures including the connectivity between G and p (e.g., friends in G who frequently view p's profile, send messages to p, and/or make postings to p's discussion are considered to be more significant to p than others) as well as the "rank" of G (e.g., friends in G who have certain experience, skills and expertise, education, and/or role are considered to be more significant to p than others). To elaborate, a group of friends who frequently participate in p's discussion by adding many posts are considered to be significant to p. Moreover, their postings are considered to be more significant if they are "ranked" as experts.

Note that, although we use LinkedIn in the above example, similar observations on the desire of mining significant groups of friends/connections/followers can be made on other social networking sites (e.g., Facebook, Twitter). So, a natural question to ask is how to find these significant friend groups, especially when a user has hundreds of friends/connections in his social network and there are many users in the network? Manually go through the entire friend list seems to be impractical. A more algorithmic approach is needed. In this paper, we propose an algorithm to help users to mine social networks for their significant friend groups. ***Key contributions*** of this paper are our proposal of a tree structure called **S**ignificant **F**riend-**tree** (SF-tree) to capture important information about friends/connections in social networks and our design of an efficient algorithm to mine significant friend groups from the SF-tree.

This paper is organized as follows. Section 2 states the problem definition. Section 3 describes our algorithm for constructing an SF-tree and mining those significant friend groups from the SF-tree. Experimental results in Section 4 show the effectiveness of our algorithm. Section 5 presents the conclusions.

2 Problem Definition

Before we define the problem of mining social networks for significant friend groups, let us consider Table 1 (which shows an illustrative friend database F_{DB} capturing social information about 10 users in a social network) and Table 2 (which shows the pre-computed confidence values of these 10 users). $F_{DB} = \{L_1, ..., L_7\}$ in Table 1 consists of seven *friend lists*, each lists friends of a user. For example, L_2 lists four friends/connections of Gail—namely, Amy, Don, Ed, and Jeff. Each friend f_i in the friend list $L_{id(p)}$ of a user p is associated with a **weight** $\boldsymbol{wt(f_i, L_{id(p)})}$ that indicates the strength/association/measure value of f_i from p's point of view. For example, "Don(20)" on L_2 indicates that Don added 20 posts to Gail's discussion. Note that the weight is asymmetric (e.g., the weight of Don on Gail's list is different from the weight of Gail on Don's list: wt(Don,L_2)=20 $\neq$ wt(Gail,L_5)=10) and can vary from one list to another (e.g., the weight of Don on Gail's and Carl's lists are different: wt(Don,L_2)=20 $\neq$ wt(Don,L_3)=30). Then, let $G = \{f_1, f_2, \ldots, f_k\}$ be a group of k common friends (i.e., *friend group*). For example, {Amy, Don} is a (common) friend group of Gail, Carl, and Helen. Let F_{DB}^G denote the set of lists in F_{DB} that contain group G. For example, if G={Amy, Don}, then F_{DB}^G=$\{L_2, L_3, L_4\}$.

Table 1. A sample friend database F_{DB}

Friend list L_{id} with weight $wt(f_i, L_{id})$	$limp(L_{id})$ with sig=0.2	
$L_1 \equiv$ Jeff:{Don(30), Gail(40), Ivy(60)}	61	49
$L_2 \equiv$ Gail:{Amy(20), Don(20), Ed(10), Jeff(50)}	56	56
$L_3 \equiv$ Carl:{Amy(10), Bob(10), Don(30), Ed(10), Jeff(20)}	52	44
$L_4 \equiv$ Helen:{Amy(50), Don(30), Ed(30)}	68	68
$L_5 \equiv$ Don:{Amy(20), Fred(10), Gail(10), Jeff(20)}	33	25
$L_6 \equiv$ Ed:{Amy(10), Fred(10), Helen(30)}	30	4
$L_7 \equiv$ Amy:{Carl(20), Don(20), Gail(30)}	45	35

Table 2. Confidence table

Friend f_i	Confidence $conf(f_i)$	Friend f_i	Confidence $conf(f_i)$
Amy	0.40	Fred	0.80
Bob	0.80	Gail	0.70
Carl	0.50	Helen	0.60
Don	0.70	Ivy	0.20
Ed	0.90	Jeff	0.50

While the weight $wt(f_i, L_{id(p)})$ indicates the strength/association/measure value of f_i with respect to a user p, the **confidence value** $\boldsymbol{conf(f_i)}$ shown in Table 2 indicates the "rank" of user f_i (based on his experience, skills and expertise, education, role, importance, reputation, prominence) in social networks. For example, opinions posted by an expert Don (with *conf*(Don)=0.70) are considered to be more important than those posted by a non-expert Ivy (with *conf*(Ivy)=0.20).

Then, the problem of *mining significant friend groups* is to discover from friend database F_{DB} all groups of friends with high significance value (e.g., higher than or equal to a user-specified minimum significance threshold *minSig*). In the following, we define the significance of friend groups in a step-by-step fashion.

Definition 1. *The* ***importance of a friend*** $\boldsymbol{f_i}$ *in a friend list* L_j*—denoted as* $\boldsymbol{fimp(f_i, L_j)}$*—measures the significance of* f_i *in* L_j *and is calculated by* $fimp(f_i, L_j) = wt(f_i, L_j) \times con(f_i)$.

Example 1. The importance of Don in L_2 shown in Table 1 is $fimp(Don, L_2) = 20 \times 0.70 = 14$, which reflects the number of posts Don made on Gail's discussion (20) and Don's individual "rank" (0.70). □

To a further extent, the list importance of a *group* G of friends in a friend list L_j can be computed by summing the $fimp(f_i, L_j)$ values for every $f_i \in G$. For example, the list importance value of G={Amy, Don} in L_2 is (20×0.40) + (20×0.70) = 8+14 = 22, which indicates the total importance of the group (consisting of both Amy and Don) in Gail's friend list L_2.

Definition 2. *The* ***importance of a friend group*** **G** ***in friend database*** $\boldsymbol{F_{DB}}$ *is defined as* $\boldsymbol{dgimp(G)} = \sum_{L_j \in F_{DB}^G} \sum_{f_i \in G} fimp(f_i, L_j)$.

Example 2. Recall that, if G={Amy, Don}, then F_{DB}^G=$\{L_2, L_3, L_4\}$ indicating that Amy and Don appear as (common) friends of Gail (L_2), Carl (L_3), and

Helen (L_4). And, $dgimp(G) = 22 + [(10\times0.40) + (30\times0.70)] + [(50\times0.40) + (30\times0.70)] = 22 + 25 + 41 = 88$. □

Definition 3. *The* ***importance of a list*** $\boldsymbol{L_j}$*—denoted as* $\boldsymbol{limp(L_j)}$*—is the total importance of all friends in a friend list* L_j *and is defined as* $limp(L_j) = \sum_{f_i \in L_j} fimp(f_i, L_j)$.

Example 3. The importance of list L_2 in Table 1 is $limp(L_2) = 8+14+(10\times0.90) +(50\times0.50) = 56$ as indicated in the second column of Table 1. This value indicates that the overall combined importance of all friends of Gail (L_2). □

Definition 4. *The* ***importance of a friend database*** $\boldsymbol{F_{DB}}$ *is the total importance of all friend lists in* F_{DB}*, i.e.,* $\boldsymbol{dimp(F_{DB})} = \sum_{L_j \in F_{DB}} limp(L_j)$.

Example 4. Recall from Example 3 that $limp(L_2)$=56. By computing the *limp* values for the remaining six friend lists in F_{DB}, we get 61+56+52+68+33+30 +45 = 345 (i.e., the sum of the second column in Table 1) as the total importance of all seven friend lists in F_{DB}. □

Definition 5. *The* ***significance*** *of a friend group* G*—denoted by* $\boldsymbol{sig(G)}$*—is defined as the ratio of the importance of* G *in* F_{DB} *to the importance of* F_{DB}*, i.e.,* $sig(G) = \frac{dgimp(G)}{dimp(F_{DB})}$.

Example 5. Recall from Example 2 that $dgimp(\{\text{Amy, Don}\})$=88, and recall from Example 4 that $dimp(F_{DB})$=345. Then, the significance of {Amy, Don} in Table 1 is $sig(\{\text{Amy, Don}\}) = \frac{88}{345} \approx 0.26$. □

A friend group is *significant* in a social network media if its significance value is no less than the user-specified minimum significance threshold (denoted as *minSig*). If *minSig*=0.20 (which represent 20% of database importance $dimp(F_{DB})$), then G={Amy, Don} is significant because its significance value $sig(G)\approx0.26 \geq 0.20$=*minSig*.

Example 6. Recall from Example 5 that {Amy, Don} is a significant friend group because $sig(\{\text{Amy, Don}\})\approx0.26$. Let us consider friend groups {Don} and {Amy}. For {Don}, its significance value $sig(\{\text{Don}\})= \frac{21+14+21+21+14}{345} = \frac{91}{345} \approx 0.26 \geq 0.20$=*minSig*. However, $sig(\{\text{Amy}\}) = \frac{8+4+20+8+4}{345} = \frac{44}{345} \approx 0.13 < 0.20$=*minSig*. In other words, although {Amy, Don} is a significant friend group, its subset {Amy} is *not* a significant friend group. □

As observed from Example 6, the significance of a friend group does not satisfy the *downward closure property* (cf. support or frequency [1], which is downward closed). This leads to a challenging problem when mining significant friend groups from social networks. To address the issue, we define the following.

Definition 6. *The* ***containing list importance*** *of a friend group* G *in* F_{DB}*—denoted as* $\boldsymbol{climp(G)}$*—is the sum of importance of all lists in* F_{DB} *that contain* G*, i.e.,* $climp(G) = dimp(F^G_{DB}) = \sum_{G \subseteq L_j \in F^G_{DB}} limp(L_j)$.

Example 7. Recall that, if G={Amy, Don}, then F_{DB}^{G}={L_2, L_3, L_4} indicating that Amy and Don appear as (common) friends of Gail (L_2), Carl (L_3), and Helen (L_4). Then, recall from Table 1 that the *limp* values for L_2, L_3 and L_4 are 56, 52 and 68, respectively. Thus, $climp(G) = 56+52+68 = 176$.

Similarly, if G'={Amy}, then $F_{DB}^{G'}$={L_2, L_3, L_4, L_5, L_6} indicating that Amy appears as a (common) friend of Gail (L_2), Carl (L_3), Helen (L_4), Don (L_5), and Ed (L_6). Thus, $climp(G') > climp(G)$ for G={Amy, Don}. This can be easily verified that $climp(G')$=176+33+30=239 as $limp(L_5)$=33 and $limp(L_6)$=30 in Table 1. □

Based on above definition, we observed that $climp(G) \leq climp(G')$ where $G' \subseteq G$. Hence, if G is a significant group, then G' must be a significant group. In other words, if G' is *not* a significant group, then G cannot be a significant group. Hence, we can maintain the downward closure property when significant friend groups are mined based on $climp(G)$ instead of $dgimp(G)$. On the one hand, an advantage of computing significance based on $climp(G)$ is that we can take advantage of the downward closure property. On the other hand, a drawback is that we may generate some false positives, which can be removed with an additional post-processing scan of F_{DB}.

3 Construction and Mining of SF-Trees

In this section, we first propose a prefix tree based data structure to efficiently capture the database content, and we then design a corresponding pattern-growth based mining algorithm to discover significant friend groups from social network database.

Our proposed tree structure is called **S**ignificant **F**riend **Tree** (SF-tree). It is compact and easy to build. Each node in an SF-tree consists of (i) an item, and (ii) *climp* (i.e., the sum of *limp* values of all lists that pass through or end at the node). In addition, the last node of a list maintains the user information (i.e., p). Note that, for any list, the information we maintain in all nodes of its path is always the same (except the last node, which keeps the user information as well), which makes the SF-tree compact. The key steps for the SF-tree construction algorithm are presented below.

3.1 SF-Tree Construction

The SF-tree construction algorithm starts by scanning both the social network database F_{DB} and the confidence table once to capture the basic information regarding users and their friend lists. With this scan, the algorithm calculates $dimp(F_{DB})$, *climp* and *sig* values of each group with a single friend. Then, it removes friends with low *climp* values and sorts all the remaining friends according to their *climp* values.

To demonstrate the SF-tree construction, let us consider F_{DB} shown in Table 1 and the confidence table shown in Table 2 when mining with *minSig*=0.20. After

Table 3. Significance and *climp* values of friends after the first DB scan

f_i	$sig(f_i)$	$climp(f_i)$	Remove?	f_i	$sig(f_i)$	$climp(f_i)$	Remove?
Amy	0.13	239	No	Fred	0.05	63	Yes
Bob	0.02	52	Yes	Gail	0.16	139	No
Carl	0.03	45	Yes	Helen	0.05	30	Yes
Don	0.26	282	No	Ivy	0.03	61	Yes
Ed	0.13	176	No	Jeff	0.13	141	No

the first scan of F_{DB}, the *sig* and *climp* values of all single friends are calculated (as shown in Table 3). Among the 10 friends, Bob, Carl, Fred, Helen and Ivy are removed from the candidate set due to their low *climp* values. Afterwards, the SF-tree is constructed in descending order of *climp* values. The H-table is then created by arranging the remaining friends in descending order of their *climp* values. Note that even though $sig(\{\text{Amy}\})$, $sig(\{\text{Ed}\})$, $sig(\{\text{Gail}\})$ and $sig(\{\text{Jeff}\})$ are all less than *minSig*, we avoid removing them from further consideration. The reason is that their corresponding *climp* values are high, which indicates that they may be significant to other friends.

With the second scan of F_{DB}, an SF-tree is constructed in a similar fashion as the FP-tree [5] by inserting each list of F_{DB}. Before inserting a list into the SF-tree, we remove all *insignificant* friends from the list and adjust its *limp* value to reflect the removal of insignificant friends. Note that friends with low *climp* values would have no influence on the computation of significant friend groups. Hence, removing these friends at this early stage helps reduce the number of false positives in the long run. Let us continue with our example, when *minSig*=0.20, we remove Ivy from L_1 and adjust $limp(L_1)$ from 61 to 49 (i.e., $limp(L_1) - fimp(\text{Ivy}, L_1) = 61 - (60 \times 0.20) = 49$). The adjusted *limp* value for each list is shown in the last column in Table 1.

For each list, the algorithm stores the new *limp* value in the tree. Fig. 1(a) shows contents of the SF-tree after inserting L_1 (for Jeff). Note that the last node in the tree (i.e., "Gail:49") maintains the user information (i.e., p=Jeff) of the list. L_2 in F_{DB} is then inserted with $limp(L_2) = 56$ (ref. Fig. 1(b)). Since L_1 and L_2 share a common prefix (i.e., "Don"), the algorithm increases the *climp* value for nodes in the common prefix (i.e., "Don") from $limp(L_1)$=49 to $limp(L_1)$+$limp(L_2)$=105 by adding the value of $limp(L_2)$. Nodes in the remaining part of L_2 carry the value of $limp(L_2)$. Since the second list is for Gail,

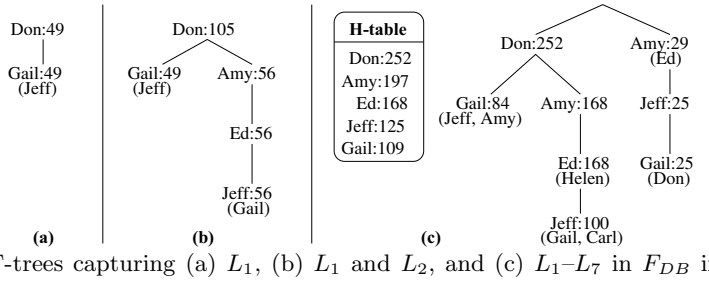

Fig. 1. SF-trees capturing (a) L_1, (b) L_1 and L_2, and (c) L_1–L_7 in F_{DB} in Table 1 when *minSig*=0.20

the last node in the path stores such user information (i.e., "Gail") as shown in Fig. 1(b). Other lists can be inserted in a similar fashion. Fig. 1(c) shows contents of the SF-tree after inserting all seven lists in F_{DB}.

To facilitate a fast tree traversal, in addition to keeping the *climp* value for each friend, the H-table also maintains node pointers to the first occurrence of each friend in the SF-tree. Similar to that of an FP-tree [5], the SF-tree also maintains horizontal node pointers for all nodes having the same friend's name. For simplicity, we do not show these pointers in the figure.

Based on the above description of SF-tree construction, the resulting SF-tree possesses the following important property: The *climp* value in a node x in an SF-tree maintains the sum of *limp* values of all the lists that pass through or end at x for all the nodes in the path from x to the root.

Lemma 1. *Given a friend database F_{DB} and a user-specified minimum significance threshold minSig, the complete set of all significant friend groups can be mined from an SF-tree built when minSig is applied to F_{DB}.*

Proof. An SF-tree keeps a set of significant friends in a list L_j for every list L_j, and the tree stores the accumulated *climp* value for each node. Hence, SF-tree mining based on this *climp* value ensures that no significant friend group will be missed. Moreover, an SF-tree is constructed by considering only the candidate significant friends (based on their *climp* values) in a list. As such, it can be assured that all potentially significant friend groups can be mined from the SF-tree built for a specific *minSig*. □

Based on Lemma 1, we can find all significant friend groups from the constructed SF-tree using a pattern-growth mining algorithm, which will be discussed in the next section.

3.2 SF-Tree Mining

Recall from Section 3.1 that a complete set of significant friend groups can be found with the first scan of F_{DB}. Hence, the SF-tree can be used for finding potentially significant friend groups having number of friends more than one. We follow the usual tree-based [5] pattern mining approach when mining our SF-tree. The basic operations in SF-tree mining are the construction of the projected databases for a potentially significant friend group and the recursive mining of the further potentially significant friend extensions of that group. It does so by examining all the conditional SF-trees consisting of the set of potentially significant friend groups occurring with a suffix group. Hence, the mining proceeds to recursively mine the SF-tree of decreasing size to generate candidate significant friend groups without additional database scan.

To illustrate how to mine the SF-tree, let us revisit our example. Specifically, given the SF-tree in Fig. 1(c) that captures F_{DB} shown in Table 1 with *minSig*=0.20, the SF-tree mining starts with the construction of a projected database for the last friend (i.e., Gail) in the H-table. Such a projected database for Gail is constructed by taking all the branches with suffix Gail as shown in

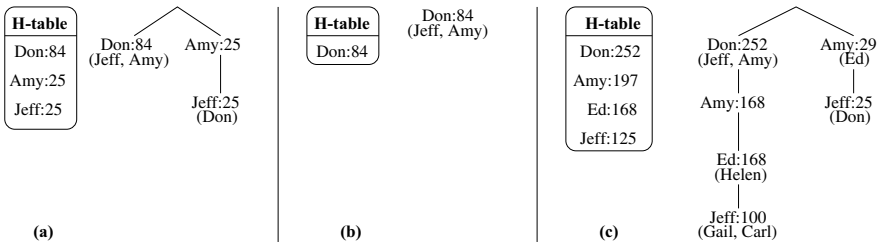

Fig. 2. Applying SF-tree mining to F_{DB} in Table 1 when *minSig*=0.20: (a) Projected DB for {Gail}, (b) Conditional DB for {Gail}, and (c) SF-tree after projecting Gail

Table 4. Significant friend groups

Candidate group:*climp*(*G*)	Friends of ...	*dgimp*(*G*)	*sig*(*G*)	Significant?
{Gail, Don}:84	{Jeff, Amy}	84	0.24	Yes
{Jeff, Ed}:100	{Gail, Carl}	53	0.15	No
{Jeff, Ed, Amy}:100	{Gail, Carl}	65	0.19	No
{Jeff, Ed, Amy, Don}:100	{Gail, Carl}	100	0.29	Yes
{Jeff, Ed, Don}:100	{Gail, Carl}	88	0.26	Yes
{Jeff, Amy}:125	{Gail, Carl, Don}	65	0.19	No
{Jeff, Amy, Don}:100	{Gail, Carl}	82	0.24	Yes
{Jeff, Don}:100	{Gail, Carl}	70	0.20	Yes
{Ed, Amy}: 168	{Gail, Carl, Helen}	77	0.22	Yes
{Ed, Amy, Don}:168	{Gail, Carl, Helen}	133	0.39	Yes
{Ed, Don}:168	{Gail, Carl, Helen}	103	0.30	Yes
{Amy, Don}:168	{Gail, Carl, Helen}	88	0.26	Yes

Fig. 2(a). The table shows the sum of *climp* values of all friends that co-occur with {Gail}. Based on this value for each friend in the SF-tree, we can find the list of friends in the projected database of {Gail} that may generate potentially significant friend group with {Gail}. For example, *climp*(Don) in the projected database of {Gail} is at least *minSig*, while *climp* values for other friends (i.e., Amy and Jeff) are less than *minSig*. Hence, we can safely remove Amy and Jeff from the projected database of {Gail} and construct the conditional database for {Gail}, as shown in Fig. 2(b).

The potentially significant friend groups are generated from the corresponding conditional databases. For example, the set of potentially significant friend groups with {Gail} is generated as {Gail, Don}:84 from the conditional database of {Gail} in Fig. 2(b), where 84 indicates the *climp* value of the group. Along with the potentially significant friend group, we also keep the user information for the group (i.e., {Jeff, Amy}) in the mining result.

After creating the projected database, the original SF-tree is adjusted by pushing the user information at the node up to its parent. For example, the user information of the node "Gail:84" (i.e., Jeff, Amy) and the node "Gail:25" (i.e., Don) are pushed to their respective parent nodes, as shown in Fig. 2(c). Such operation enables us to obtain the correct user information for any node in the SF-tree during the whole mining phase.

Further extension of the potentially significant friend group {Don, Gail} is mined by creating a projected database for {Gail, Don} from the conditional

database of {Gail}. In our example, the projected database for {Gail, Don} is empty, which indicates that no further candidate significant friend group will be generated from {Gail}'s conditional database. Hence, mining for {Gail} is terminated. Mining for the remaining friends in the H-table is carried out in a similar fashion. The set of all candidate significant friend groups generated by mining the SF-tree is shown in Table 4. With another database scan, we eliminate all the non-significant friend groups from the set of candidate significant friend groups.

4 Experimental Results

In this section, we evaluated different aspects (e.g., the number of generated candidate groups, runtimes, and scalability) of SF-trees in mining significant friend groups from social media. To the best of our knowledge, SF-tree mining is the first approach to mine significant friend groups from social media databases. However, as the mining of high utility patterns [18] can be considered to be relevant to our mining of significant friend groups, we compared our SF-tree mining with three existing high utility pattern mining algorithms (e.g., Two Phase [11], FUM and DCG+ [9]). All programs were written in Microsoft Visual C++ 6.0 and run with Windows XP operating system on a 2.13 GHz CPU with 2GB main memory. There are some basic differences among our proposed SF-tree mining, Two Phase, FUM, and DCG+. First, the three existing high utility mining algorithms are all Apriori-based [1], i.e., they use the levelwise candidate generation-and-test paradigm. They require N scans of F_{DB} (where N is the maximum size of high utility patterns) and a high computation cost for generating the candidates. In contrast, our SF-tree mining explores the tree-based pattern-growth mining technique, which allows us to mine the complete set of significant friend groups with three scans of F_{DB} without using the level-wise candidate generation-and-test paradigm. Second, unlike our SF-tree, the high utility mining algorithms do not maintain the user information in their respective data structure, which restricts them from providing the knowledge of association between significant friend groups and others.

Since the three high utility pattern mining algorithms were not designed for social network mining, we used the datasets that are mostly used in frequent pattern mining domain for fair comparison. In these datasets that are available at the Frequent Itemset Mining Implementation dataset repository (`http://fimi.cs.helsinki.fi/data`), each transaction consists of a unique transaction ID and a set of items (which is similar to a list of friends described

Table 5. Characteristics of datasets

Dataset	Type	#transactions	#items	Max trans. len.	Avg trans. len.
Mushroom	Dense	8124	119	23	23
Retail	Sparse	88162	16470	76	10.31
Kosarak	Sparse	990002	41270	2498	8.1

in this paper). Table 5 shows the characteristics (e.g., dense vs. sparse, large vs. small dataset, long vs. short transactions) of these datasets. For example, *Mushroom* is a dense dataset with many long frequent patterns. We use this dataset to demonstrate the type of social media data with fewer number of people but long/large groups in each person's list. Both *Retail* and *Kosarak* are sparse and very large datasets (in terms of number of transactions and domain items) with a combination of long and short transactions. Hence, they may correspond to scenarios when a very large number of individuals use a social network and/or when the connectivity between a user and their friends in a social network vary a lot. For all these datasets, we mapped transaction IDs into user information, sets of items into friend groups in every list, and transactions into friend lists. In addition, to represent $wt(f_i, L_j)$, we associated a random number to each item in a transaction. We also generated random numbers as the confidence values of friends in F_{DB}. As ongoing work, we plan to conduct additional experiments using social network data.

4.1 Candidate Significant Friend Group Generation

We compared the number of candidate significant friend groups (i.e., false positives) generated by our SF-tree mining with the three existing algorithms over different datasets by varying *minSig* values. As shown in Fig. 3, the number of candidates increased when lowering the value of *minSig* for all datasets and algorithms. However, SF-tree outperformed the other three algorithms as SF-tree generated significantly fewer candidates. The reason was that, for dense datasets (e.g., *Mushroom*), there is a very high probability for each friend to occur in every list, which increases the number of potential groups when applying the candidate generation-and-test paradigm. Hence, the three high utility mining algorithms generated comparatively large number of candidates. Conversely, sparse datasets (e.g., *Retail*) contain too many individual friends in many transactions, which further increases the number of candidate groups for the three high utility mining algorithms. In contrast, as SF-tree uses tree-based pattern-growth mining, it avoids generating candidates in dense or sparse datasets.

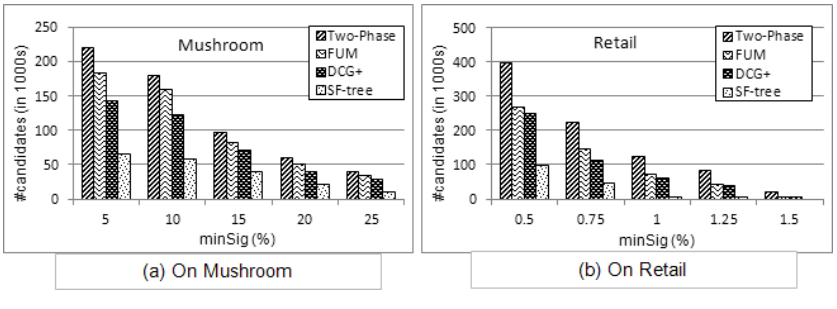

Fig. 3. The number of candidates (i.e, false positives)

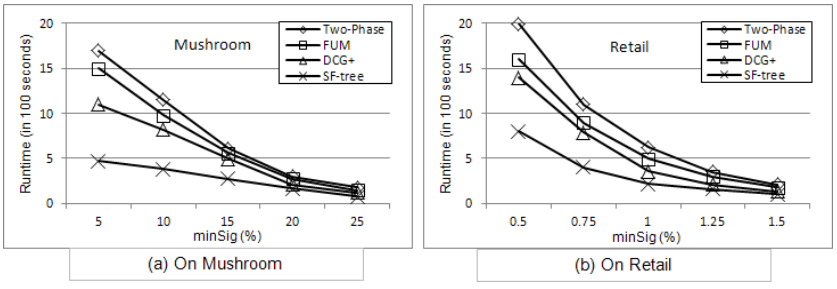

Fig. 4. Runtime

4.2 Runtimes

We measured runtimes of SF-tree and the three high utility mining algorithms. The reported runtimes for SF-tree include times for tree construction (i.e., two scans of F_{DB}), mining, and candidate pruning (i.e., an additional scan of F_{DB}); the reported runtimes for the other three algorithms include times for multiple database scans for candidate generation-and-test.

The number of generated candidates (which increased with the decrease of *minSig*) directly influences the runtimes for all algorithms. The more the generated candidates, the longer were the runtimes. As shown in Fig. 4, SF-tree performed better than the other three algorithms for all datasets. For example, the three algorithms generated very large numbers of candidates for *Mushroom*. To handle a large number of friends and friend lists in *Retail*, the three algorithms required substantially longer runtimes to scan F_{DB} when compared to SF-tree (which required three scans of F_{DB}). Although the performance of all algorithms were similar for higher *minSig* values (due to a very small number of generated candidate friend groups), the gap was widened for lower *minSig* values.

4.3 Scalability Test

To test the scalability of our SF-tree mining, we used very large datasets such as *Kosarak*. Fig. 5 shows results on scalability tests of all four algorithms. Due to substantially long runtime required to handle *Kosarak* for some low thresholds, we did not obtain reportable results for DCG+. Moreover, although both Two Phase and FUM completed the mining process for high *minSig*, they suffered from some problem for low *minSig*. Hence, we did not plot their runtimes for low thresholds. In contrast, our SF-tree mining completed the mining process and showed linear scalability with high and low *minSig* thresholds. SF-tree generated significantly fewer candidate friend groups within very reasonable runtimes due to early pruning of non-significant friends during the tree construction and mining process. This demonstrated that SF-tree mining is scalable.

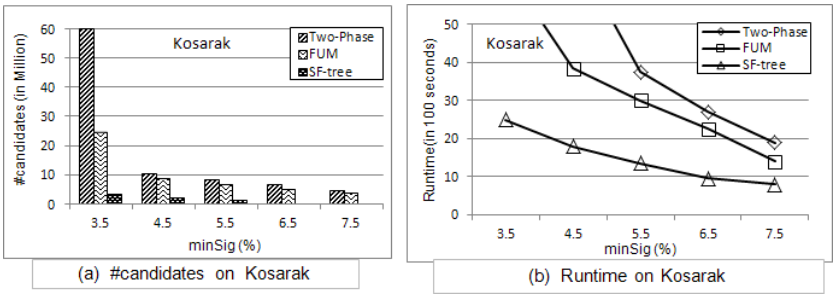

Fig. 5. Scalability test

5 Conclusions

In this paper, we proposed a novel algorithm to mine social networks for significant friend groups. Our social network mining algorithm first constructs a significant friend-tree (SF-tree) to capture important information about linkage between users in the social networks, and it then uses the SF-tree to find significant friend groups among all friends of users in the social networks.

Acknowledgement. This project is partially supported by NSERC (Canada) and University of Manitoba.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 1994, pp. 487–499 (1994)
2. Cameron, J.J., Leung, C.K.-S., Tanbeer, S.K.: Finding strong groups of friends among friends in social networks. In: IEEE DASC (SCA) 2011, pp. 824–831 (2011)
3. Carrington, P.J., Scott, J., Wasserman, S. (eds.): Models and Methods in Social Network Analysis. Cambridge University Press (2005)
4. Fan, W., Yeung, K.H.: Virus propagation modeling in Facebook. In: ASONAM 2010, pp. 331–335 (2010)
5. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1–12 (2000)
6. Lee, W., Lee, J.J.-H., Song, J.J., Eom, C.S.-H.: Maximum reliable tree for social network search. In: IEEE DASC (CSN) 2011, pp. 1243–1249 (2011)
7. Leung, C.K.-S., Carmichael, C.L.: Exploring social networks: a frequent pattern visualization approach. In: IEEE SocialCom 2010, pp. 419–424 (2010)
8. Leung, C.K.-S., Carmichael, C.L., Teh, E.W.: Visual Analytics of Social Networks: Mining and Visualizing Co-authorship Networks. In: Schmorrow, D.D., Fidopiastis, C.M. (eds.) FAC 2011, HCII 2011. LNCS (LNAI), vol. 6780, pp. 335–345. Springer, Heidelberg (2011)
9. Li, Y.-C., Yeh, J.-S., Chang, C.-C.: Isolated items discarding strategy for discovering high utility itemsets. DKE 64(1), 198–217 (2008)
10. Liu, H., Yu, P.S., Agarwal, N., Suel, T.: Guest editors' introduction: social computing in the blogosphere. IEEE Internet Computing 14(2), 12–14 (2010)

11. Liu, Y., Liao, W.-k., Choudhary, A.K.: A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005)
12. Obradovic, D., Pimenta, F., Dengel, A.: Mining shared social media links to support clustering of blog articles. In: CASoN 2011, pp. 181–184 (2011)
13. Pennacchiotti, M., Popescu, A.-M.: Democrats, republicans and starbucks afficionados: user classification in Twitter. In: ACM KDD 2011, pp. 430–438 (2011)
14. Tang, C., Ross, K., Saxena, N., Chen, R.: What's in a Name: A Study of Names, Gender Inference, and Gender Behavior in Facebook. In: Xu, J., Yu, G., Zhou, S., Unland, R. (eds.) DASFAA Workshops 2011. LNCS, vol. 6637, pp. 344–356. Springer, Heidelberg (2011)
15. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press (1994)
16. Weng, J., Lim, E.-P., Jiang, J., He, Q.: TwitterRank: finding topic-sensitive influential twitterers. In: ACM WSDM 2010, pp. 261–270 (2010)
17. Xu, G., Zong, Y., Pan, R., Dolog, P., Jin, P.: On Kernel Information Propagation for Tag Clustering in Social Annotation Systems. In: König, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R.J., Jain, L.C. (eds.) KES 2011, Part II. LNCS (LNAI), vol. 6882, pp. 505–514. Springer, Heidelberg (2011)
18. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: SDM 2004, pp. 482–486 (2004)
19. Ye, S., Wu, S.F.: Measuring Message Propagation and Social Influence on Twitter.com. In: Bolc, L., Makowski, M., Wierzbicki, A. (eds.) SocInfo 2010. LNCS, vol. 6430, pp. 216–231. Springer, Heidelberg (2010)
20. Yumoto, T., Sumiya, K.: Measuring Attention Intensity to Web Pages Based on Specificity of Social Tags. In: Yoshikawa, M., Meng, X., Yumoto, T., Ma, Q., Sun, L., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 6193, pp. 264–273. Springer, Heidelberg (2010)