Mining Popular Patterns: A Novel Mining Problem and Its Application to Static Transactional Databases and Dynamic Data Streams

Alfredo Cuzzocrea[1], Fan Jiang[2], Carson K. Leung[2](✉), Dacheng Liu[2,3], Aaron Peddle[2], and Syed K. Tanbeer[2]

[1] ICAR-CNR and University of Calabria, Rende, CS, Italy
cuzzocrea@si.dimes.unical.it
[2] University of Manitoba, Winnipeg, MB, Canada
kleung@cs.umanitoba.ca
[3] Wuhan University, Wuhan, Hubei, China

Abstract. Since the introduction of the frequent pattern mining problem, researchers have extended frequent patterns to different useful patterns such as cyclic, emerging, periodic and regular patterns. In this paper, we (i) introduce *popular patterns*, which capture the popularity of individuals, items, or events among their peers or groups. Moreover, we also propose (ii) the *Pop-tree* structure to capture the essential information from transactional databases and (iii) the *Pop-growth* algorithm for mining popular patterns from the Pop-tree. Moreover, we illustrate how our algorithm (iv) mines popular friends from social networks. As we are not confined to mining popular patterns from static transactional databases, we extend our work to mining popular patterns from dynamic data streams. Specifically, we propose (v) the *Pop-stream* structure to capture the popular patterns in batches of data streams and (vi) the *Pop-streaming* algorithm for mining popular patterns from the Pop-stream structure. Experimental results showed that (i) our proposed tree structure is compact and space efficient and (ii) our proposed algorithm is time efficient in mining popular patterns from static transactional databases and dynamic data streams.

Keywords: Data mining · Knowledge discovery · Interesting patterns · Popular patterns · Useful patterns · Tree-based mining · Data streams

1 Introduction

Since the introduction of the research problem of frequent pattern mining, numerous works have been proposed. These works can mostly be classified into two broad "categories". Works in the first "category" mainly focused on algorithmic efficiency, while works in the second "category" mainly focused on extending the notion of frequent patterns to other interesting or useful patterns. However, the mining of these patterns are mostly based on the support/frequency measure. While support/frequency is a useful metric, support-based frequent pattern mining may *not* be sufficient to discover many interesting knowledge (e.g., popularity) among patterns in a transactional database (TDB). However, in many real-life situations, users want to find popular patterns. For example, a social analyst may want to find persons with large “groups” of friends in social networks as these persons can be the most influential one in their groups or the social networks [9,26]. Similarly, a new member may want to know individuals with high connectivity so that he can get to know more members quickly. A recommender may want to know researchers with large numbers of collaborators. As the fourth example, an event promoter may want to find events with large numbers of participants. With the increase in usage of social network media, it has become more important to be able to find popular individuals (or items, objects, events).

While data in many real-life situations are *static* (e.g., mining popular merchandise items from shopper market basket transactions in a transactional database), the automation of measurements and data collection in some other real-life situations is producing *dynamic* streams of data. For instance, the development and increasing use of a large number of sensors (e.g., acoustic, chemical, electromagnetic, mechanical, optical radiation and thermal sensors) for various real-life applications have led to data streams [11,23].

Hence, in this paper, we aim to mine popular patterns from both static transactional databases and dynamic data streams. Specifically, our **key contributions** of this paper include the following:

1. our introduction of the notion of *popular patterns*;
2. our proposal of the *Pop-tree*, which is a tree structure to capture essential information about the popularity of individuals, items, objects, or events;
3. our design and development of the *Pop-growth algorithm*, which mines popular patterns from the Pop-tree capturing static data;
4. our proposal of the *Pop-stream*, which is a structure to capture popular individuals, items, objects, or events mined from batches of dynamic data streams; and
5. our design and development of the *Pop-streaming algorithm*, which finds popular patterns from the Pop-stream structure capturing dynamic data.

As the current paper is an extension and enhancement of our DaWaK 2012 paper [25], additional contributions beyond the basic mining of popular patterns from static transactional databases include the following: (i) extending the mining of popular patterns from static transactional databases to the mining of popular patterns from dynamic data streams (Sect. 6), (ii) demonstration of our algorithm for a useful application of mining popular friends from social networks (Sect. 5), and (iii) running additional experiments (especially for popular pattern mining from streams).

The remainder of this paper is organized as follows. The next section reviews some related works. We introduce popular patterns in Sect. 3. In Sect. 4, we propose (i) the Pop-tree structure that captures important contents of the TDB and

(ii) the Pop-growth algorithm that constructs the Pop-tree, from which popular patterns can be mined recursively. In Sect. 6, we propose (i) the Pop-stream structure that captures popular patterns mined from batches of data streams and (ii) the Pop-streaming algorithm that calls Pop-growth to find popular patterns from a batch, stores the mined patterns in the Pop-stream structure, and returns to users popular patterns that can be mined from the dynamic data streams. We demonstrate a useful real-life application of mining popular friends from social networks in Sect. 5. Experimental results are presented in Sect. 7. Finally, conclusions and future work are provided in Sect. 8.

2 Related Work

Recall from the previous section that numerous works have been proposed since the introduction of the research problem of frequent pattern mining. These works can mostly be classified into two broad "categories". Works in the first "category" mainly focused on algorithmic efficiency [22,23]. For example, to avoid the candidate generation-and-test approach of the Apriori algorithm [1], a tree-based algorithm called FP-growth [17] was proposed to build an FP-tree to capture the contents of a TDB so that frequent patterns can be mined recursively from the FP-tree with a restricted test-only approach.

Works in the second "category" mainly focused on extending the notion of frequent patterns to other interesting or useful patterns such as sequences, episodes, maximal and closed sets. Note that the mining of these patterns are mostly based on the support/frequency measure. While support/frequency is a useful metric, support-based frequent pattern mining may *not* be sufficient to discover many interesting knowledge (e.g., correlation, regularity, periodicity, popularity) among patterns in a TDB. This leads to the introduction of some interestingness measures [31] and their corresponding patterns such as emerging patterns [2], constrained patterns [19,24], correlated patterns [20], periodic patterns [29,34], regular patterns [30], hyperclique patterns [32], and high utility patterns [33].

Nowadays, the automation of measurements and data collection is producing tremendously huge volumes of data. For instance, the development and increasing use of a large number of sensors (e.g., acoustic, chemical, electromagnetic, mechanical, optical radiation and thermal sensors) for various real-life applications (e.g., environment surveillance, security, manufacturing systems) have led to data streams [11,23]. To discover useful knowledge from these streaming data, several mining algorithms [10,12,14] have been proposed. In general, mining frequent patterns from dynamic data streams [16,18,28] is more challenging than mining from traditional static transaction databases due to the following characteristics of data streams:

1. Data streams are continuous and unbounded. As such, we no longer have the luxury to scan the streams multiple times. Once the streams flow through, we lose them. We need some techniques to capture important contents of the

streams. For instance, landmark windows capture contents of all batches after the landmark (i.e., sizes of windows keep increasing with the number of batches).
2. Data in the streams are not necessarily uniformly distributed. As such, a currently infrequent pattern may become frequent in the future and vice versa. We have to be careful not to prune infrequent patterns too early; otherwise, we may not be able to get complete information such as frequencies of some patterns (as it is impossible to recall those pruned patterns).

Over the past few years, several stream mining algorithms—including FP-streaming [15], UF-streaming [22], TUF-streaming [21,23], and XTUF-streaming [21]— have been proposed. However, most of them mine *frequent* patterns (instead of popular patterns. In contrast, our Pop-streaming algorithm mines *popular* patterns.

3 Our Proposed Notion of Popular Patterns

Let `Item`$=\{x_1, x_2, \ldots, x_m\}$ be a set of m domain items. A transactional database (TDB) is the set of n transactions: $\{t_1, t_2, \ldots, t_n\}$, where each transaction t_j in the TDB is a subset of `Item`. We use $|t_j|$ to represent the transaction length of t_j. Let $X = \{x_1, x_2, \ldots, x_k\} \subseteq$ `Item` be a pattern consisting of k items (i.e., a k-itemset), where $|X| = k \leq m$. Then, the projected database of X (denoted as DB_X) is a set of TDB transactions (in the TDB) that contain X. We use $maxTL(X)$ and $sumTL(X)$ to respectively represent the maximum length and the total length of all transactions in DB_X.

Example 1. Consider the TDB shown in Table 1, which consists of n=7 transactions and m=9 domain items $a, b, \ldots, i$. For pattern $X = \{b, c\}$, its projected database $DB_{\{b,c\}}$=$\{t_2, t_3\}$. Hence, $|DB_{\{b,c\}}| = 2$. In other words, the support (or frequency) of $\{b, c\}$ in the TDB is $sup(\{b, c\}$, TDB$) = sup(\{b, c\}, DB_{\{b,c\}}) = 2$. Moreover, $|t_2| = |\{b, c, f, g, h\}| = 5$, $|t_3| = |\{b, c, d, e, f, h\}| = 6$, $maxTL(\{b, c\})$ $= \max\{|t_2|, |t_3|\} = \max\{5, 6\} = 6$, and $sumTL(\{b, c\}) = |t_2|+|t_3| = 5+6 = 11$. □

Table 1. A transaction database

Transaction ID	Transaction
t_1	$\{b, d\}$
t_2	$\{b, c, f, g, h\}$
t_3	$\{b, c, d, e, f, h\}$
t_4	$\{c, e, g, h\}$
t_5	$\{a, d\}$
t_6	$\{a, b, i\}$
t_7	$\{a, d, e\}$

Definition 1. *The* **transaction popularity** $\boldsymbol{Pop(X, t_j)}$ *of a pattern X in transaction t_j measures the membership degree of X in t_j. For simplicity, we compute the membership degree based on the difference between the transaction length $|t_j|$ and the pattern size $|X|$:*

$$Pop(X, t_j) = |t_j| - |X|. \tag{1}$$

Note that, depending on real-life applications, the above equation can be adapted to incorporate some other functional operations on t_j and X.

Definition 2. *The* **long transaction popularity** $\boldsymbol{Pop(X, t^{maxTL(X)})}$ *of a pattern X in transaction $t^{maxTL(X)}$ measures the membership degree of X in $t^{maxTL(X)}$, where $t^{maxTL(X)}$ is the transaction having the maximum length in DB_X:*

$$Pop(X, t^{maxTL(X)}) = \left(\max_{t_j \in DB_X} |t_j|\right) - |X|. \tag{2}$$

Definition 3. *The* **popularity** $\boldsymbol{Pop(X)}$ *of a pattern X in DB_X measures an aggregated membership degree of X in all transactions in DB_X. It is defined as an* average *of all transaction popularities of X:*

$$Pop(X) = \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} Pop(X, t_j). \tag{3}$$

Example 2. Reconsider the TDB shown in Table 1. The transaction popularity of pattern $\{b, c\}$ in t_2 can be computed as $Pop(\{b, c\}, t_2) = |t_2| - |\{b, c\}| = 5 - 2 = 3$. Similarly, $Pop(\{b, c\}, t_3) = |t_3| - |\{b, c\}| = 6 - 2 = 4$. Recall from Example 1 that $DB_{\{b,c\}}=\{t_2, t_3\}$ (i.e., $\{b, c\}$ appears only in t_2 and t_3). As t_3 is the longest transaction in $DB_{\{b,c\}}$ (because $maxTL(\{b, c\})=6$), the long transaction popularity of pattern $\{b, c\}$ in $t^{maxTL(\{b,c\})}$ can be computed as $Pop(\{b, c\}, t^{maxTL(\{b,c\})}) = \max\{|t_2|, |t_3|\} - |\{b, c\}| = 6 - 2 = 4$. Hence, the popularity of pattern $\{b, c\}$ is $\frac{1}{sup(\{b,c\}, DB_{\{b,c\}})}(Pop(\{b, c\}, t_2)+Pop(\{b, c\}, t_3)) = \frac{1}{2}(3+4) = 3.5$ □

Definition 4. *Given a user-specified minimum popularity threshold minpop, a pattern X is considered* **popular** *if its popularity is at least minpop (i.e., $Pop(X) \geq minpop$).*

Example 3. If the user-specified *minpop* is 3.3, then pattern $\{b, c\}$ is popular in the TDB shown in Table 1 because $Pop(\{b, c\})=3.5 \geq 3.3=minpop$. However, pattern $\{b\}$ is *not* popular because $Pop(\{b\}) = \frac{1}{sup(\{b\}, DB_{\{b\}})}(Pop(\{b\}, t_1)+ Pop(\{b\}, t_2)+Pop(\{b\}, t_3)+Pop(\{b\}, t_6)) = \frac{1}{4}(1+4+5+2) = 3 < 3.3 = minpop$.□

4 Pop-Growth: Mining Popular Patterns from Static Transactional Databases

When mining frequent patterns, the frequency measure satisfies the downward closure property (i.e., if a pattern is infrequent, its superset is guaranteed to be infrequent). This helps reduce the search/solution space by pruning infrequent patterns, and thus speeds up the mining process. However, when mining popular patterns, observant readers may notice from Example 3 that popularity does *not* satisfy the downward closure property. For example, a pattern (e.g., $\{b\}$) is unpopular, but its superset (e.g., $\{b, c\}$) may be popular. Hence, the mining of popular patterns can be challenging.

To handle the challenge, let us revisit Eq. (3) and redefine the popularity $Pop(X)$ of a pattern X (cf. Definition 3).

Definition 5. *The* **popularity** $\boldsymbol{Pop(X)}$ *of a pattern* X *in* DB_X *measures an aggregated membership degree of* X *in all transactions in* DB_X*. It is defined in terms of* $sumTL(X) = \sum_{t_j \in DB_X} |t_j|$ *as follows:*

$$\begin{aligned} Pop(X) &= \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} Pop(X, t_j) \\ &= \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} (|t_j| - |X|) \\ &= \frac{sumTL(X)}{sup(X, DB_X)} - |X|. \end{aligned} \tag{4}$$

Example 4. Reconsider the TDB shown in Table 1. Recall from Example 1 that $sumTL(\{b, c\})$=11. Then, the popularity of pattern $\{b, c\}$ is $\frac{sumTL(\{b,c\})}{|\{t_2,t_3\}|} - |\{b, c\}| = \frac{11}{2} - 2 = 3.5$. Similarly, the popularity of pattern $\{b\}$ is $\frac{sumTL(\{b\})}{|\{t_1,t_2,t_3,t_6\}|} - |\{b\}| = \frac{|t_1|+|t_2|+|t_3|+|t_6|}{|\{t_1,t_2,t_3,t_6\}|} - |\{b\}| = \frac{16}{4} - 1 = 3$. □

Observant readers may notice from Example 4 that $sumTL(\{b, c\})$=11 $\leq$ 16=$sumTL(\{b\})$. The definition of $sumTL(X)$ further confirms that the total transaction length $sumTL(X)$ of X satisfies the downward closure property. In other words, for $X \subseteq X'$,

$$sumTL(X) \geq sumTL(X'). \tag{5}$$

4.1 Construction of a Pop-Tree

To mine popular patterns, we propose the Pop-growth algorithm, which consists of two key procedures: (i) construction of a Pop-tree and (ii) mining of popular patterns from the Pop-tree.

We first build a tree structure—called **Popular pattern tree (Pop-tree)**—to capture the necessary information from the TDB with only two scans of the TDB. Recall from Sect. 3 that $Pop(X)$ does not satisfy the downward closure property. So, unpopular items need to be kept in the Pop-tree as some of their supersets may be popular. Fortunately, recall from Sect. 3 that $sumTL(X)$ satisfies the downward closure property. So, not all unpopular items need to be kept. Some of them can be pruned. See the following two lemmas.

Lemma 1. *The popularity of a pattern X is always less than or equal to its long transaction popularity, i.e., $Pop(X) \leq Pop(X, t^{maxTL(X)})$.*

Proof. Recall from Equation (3) that $Pop(X) = \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} Pop(X, t_j)$, where $Pop(X, t_j) = |t_j| - |X|$. According to Equation (1), $Pop(X, t_j)$ measures the membership degree of X in t_j. As shown in Equation (2), The long transaction popularity $Pop(X, t^{maxTL(X)})$ measures the membership degree of X in the longest transaction containing X. Hence, $Pop(X)$ is always less than or equal to $Pop(X, t^{maxTL(X)})$:

$$\begin{aligned} Pop(X) &= \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} Pop(X, t_j) \\ &= \frac{1}{sup(X, DB_X)} \sum_{t_j \in DB_X} (|t_j| - |X|) \\ &= \frac{\sum_{t_j \in DB_X} |t_j|}{|DB_X|} - |X| \\ &= \left(\operatorname*{avg}_{t_j \in DB_X} |t_j| \right) - |X| \\ &\leq \left(\max_{t_j \in DB_X} |t_j| \right) - |X| \; = \; Pop(X, t^{maxTL(X)}). \end{aligned} \tag{6}$$

Hence, this proved that $Pop(X) \leq Pop(X, t^{maxTL(X)})$. □

Lemma 2. *For $X \subseteq X'$, $Pop(X')$ cannot exceed $maxTL(X) - |X'|$.*

Proof. Recall from Equation (4) that $Pop(X) = \frac{sumTL(X)}{sup(X, \text{TDB})} - |X|$. Knowing that $sumTL(X)$ satisfies the download closure property, we get $sumTL(X') \leq sumTL(X)$ for $X \subseteq X'$ as shown in Equation (5). Then, we get the following:

$$\begin{aligned} Pop(X') &= \frac{sumTL(X')}{sup(X', \text{TDB})} - |X'| \\ &\leq maxTL(X') - |X'| \\ &\leq maxTL(X) - |X'| \end{aligned} \tag{7}$$

Hence, this proved that $Pop(X') \leq maxTL(X) - |X'|$. □

Based on the above two lemmas, the following equation provides us with an upper bound of the popularity $Pop(X')$ of a pattern X' (in terms of $maxTL(X)$), where $X \subseteq X'$:

$$Pop^{UB}(X') \leq maxTL(X) - |X'|. \tag{8}$$

Based on Equation (8), we can first calculate the popularity upper bound of a pattern X' from $maxTL(X)$ where (i) $X \subseteq X'$, and (ii) $|X'| \geq |X|+1 = k+1$). We can then prune unpopular patterns. We call this *super-pattern popularity check.*

Similar to FP-tree [17], each node of a Pop-tree contains the parent and child pointers as well as horizontal node traversal pointers. To facilitate popular pattern mining, we keep (i) an item x, (ii) support of $Y \cup \{x\}$, (iii) $sumTL(Y \cup \{x\})$, and (iv) $maxTL(Y \cup \{x\})$, where Y represents the set of items above x (i.e., ancestor nodes of x).

To construct a Pop-tree, we scan the TDB to find the $support(x)$, maximum transaction length $maxTL(x)$ and the popularity $Pop(x)$ for each singleton x in the TDB. Then, we perform the super-pattern popularity check and safely delete a pattern x if $Pop^{UB}(x') < minpop$ (where x' is an extension of x). We then scan the TDB the second time to insert each transaction into the Pop-tree in a similar fashion as the insertion process of FP-tree.

Example 5. Let us show how to construct a Pop-tree for the TDB shown in Table 1 with $minpop$=2.4. With the first database scan, we obtain the following information in the form of

$$\langle x\text{: } sup(x, \text{TDB}),\ maxTL(x),\ Pop(x)\rangle$$

for each of the m=9 domain items: $\langle a$:3,3,1.66$\rangle$, $\langle b$:4,6,3.0$\rangle$, $\langle c$:3,6,4.0$\rangle$, $\langle d$:4,6,2.25$\rangle$, $\langle e$:3,6,3.33$\rangle$, $\langle f$:2,6,4.5$\rangle$, $\langle g$:2,5,3.5$\rangle$, $\langle h$:3,6,4.0$\rangle$, $\langle i$:1,3,2.0$\rangle$. This information is useful as follows. (i) Based on the obtained $Pop(x)$ values, we noticed that all items—except a, d &i—are popular (i.e., with popularity at least 2.4). Although $\{a\}$, $\{d\}$ &$\{i\}$ are unpopular, their super-patterns may be popular. Hence, we cannot delete them without performing the super-pattern popularity check. So, (ii) the obtained $maxTL(x)$ values are used for super-pattern popularity check. The popularity upper bounds of the extensions of $\{a\}$, $\{d\}$ &$\{i\}$ are at most 3−2=1, 6−2=4 &3−2=1, respectively. As the value for $Pop^{UB}(\{d\})$ is greater than $minpop$, we keep $\{d\}$ but safely delete $\{a\}$ &$\{i\}$. Finally, (iii) we sort and insert items b, c, d, e, f, g &h into a header table (H-table) in the descending order of the obtained $sup(x, \text{TDB})$ values: $\langle b, d, c, e, h, f, g\rangle$.

We then scan the TDB the second time. We compute the length of each transaction, remove all items that are not in the H-table, and sort the remaining

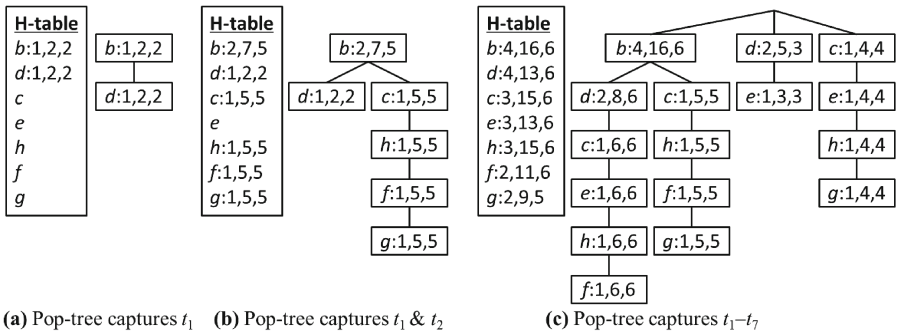


Fig. 1. The Pop-tree construction

items in each transaction according to the H-table order. Figure 1(a) shows the contents of the H-table in the form of

$$\langle x\text{: } sup(x, \text{TDB}),\ sumTL(x),\ maxTL(x)\rangle,$$

and the Pop-tree structure after inserting t_1 of TDB. Because t_1 and t_2 share a common prefix (i.e., $\{b\}$), we increase the occurrence count of the common node $\{b : 1, 2, 2\}$ by one, its total transaction length ($sumTL$) by the transaction length of t_2 (i.e., $|t_2|$=5), and update its $maxTL$. For the remaining (uncommon) nodes of t_2, we set *support*=1, $sumTL$=$|t_2|$ and $maxTL$=$|t_2|$. The contents of the Pop-tree after insertion of t_2 are shown in Fig. 1(b). The final Pop-tree after capturing all the transactions in the TDB is shown in Fig. 1(c). □

Let $I(t_j)$ be the set of items in transaction t_j that pass through the first database scan. Based on the above Pop-tree construction procedure, we observed several important properties of Pop-trees listed as follows.

Property 1. A Pop-tree registers the projection of $I(t_j)$ for t_j in the TDB only once.

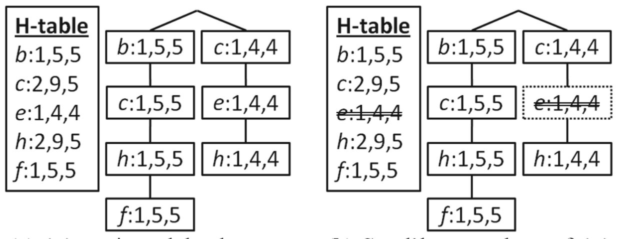
Property 2. The total transaction length $sumTL$ in a node x in a Pop-tree captures the sum of lengths of all transactions that pass through, or end at, the node for all the nodes in the path from x up to the root.

Property 3. The total transaction length $sumTL$ of any node in a Pop-tree is greater than or equal to the sum of transaction lengths of its children.

Properties 2 and 3 are the result of sharing common prefixes by different transactions, which allow our Pop-tree to be compact. Based on following lemma, one can observe that a Pop-tree is a highly compact tree structure.

Lemma 3. *The size of a Pop-tree on a TDB for minpop is bounded above by* $\sum_{t_j \in \text{TDB}} |I(t_j)|$.

Proof. Recall that $I(t_j)$ denotes the set of items in transaction t_j that pass through the first database scan. In other words, $I(t_j)$ represents the set of individually popular items in t_j. During the Pop-tree construction, these items in t_j are inserted as a tree path into a Pop-tree for popular pattern mining. Thus, as we insert every transaction t_j in the TDB, the size of the resulting Pop-tree—-in terms of the total number of tree nodes—would be equal to the total number of individually popular items in all the transactions, i.e., $\sum_{t_j \in \text{TDB}} |I(t_j)|$. This would be the worst case scenario, in which there is no tree path sharing (i.e., no common nodes can be merged). Fortunately, in most cases, some paths are in common and can thus be merged. In those cases, the size of a Pop-tree—in terms of the total number of tree nodes—would be lower than the total number of individually popular items in all transactions. Hence, this proved that the size of a Pop-tree on a TDB for *minpop* is bounded above by $\sum_{t_j \in \text{TDB}} |I(t_j)|$. □



(a) $\{g\}$-projected database **(b)** Cond'l pattern base of $\{g\}$

Pattern	Popularity
$\{b,g\}$	$(5/1)-2 = 3$
$\{f,g\}$	$(5/1)-2 = 3$
$\{c,g\}$	$(9/2)-2 = 2.5$
$\{g,h\}$	$(9/2)-2 = 2.5$

(c) Popular patterns with g

Fig. 2. The Pop-tree mining

Lemma 4. *Given a TDB and minpop, the complete set of all popular patterns can be obtained from a Pop-tree for the minpop on the TDB.*

Proof. Given a TDB and *minpop*, the Pop-tree keeps every item x with $Pop(x) \geq minpop$ (i.e., keeps all individually popular items). Every possible pattern X with $Pop^{UB}(X) \geq minpop$ can then be generated from the Pop-tree. As $Pop^{UB}(X) \geq Pop(X)$ for any pattern X, this implies that all patterns X with $Pop(X) \geq minpop$ (i.e., true positives) can be generated. As a by-product, some patterns Y with $Pop^{UB}(Y) \geq minpop > Pop(Y)$ (i.e., false positives) could be generated from the Pop-tree. As a preview, the corresponding Pop-growth algorithm would prune these false positives as its last step and would return only those true positives to the user. Hence, given a TDB and *minpop*, the complete set of all popular patterns can be obtained from a Pop-tree for the minpop on the TDB. $\square$

We can justify the completeness of a Pop-tree for mining popular patterns by Lemma 4. Based on this lemma, popular patterns can be found by mining our Pop-tree.

4.2 Finding Popular Patterns from the Pop-Tree

Recall that, to mine popular patterns, the Pop-growth algorithm applies two key procedures: (i) construction of a Pop-tree and (ii) mining of popular patterns from the Pop-tree. The Pop-growth finds popular patterns from the Pop-tree, in which each tree node captures its occurrence count, total transaction length, and maximum transaction length. The algorithm finds popular patterns by constructing the projected database for potential popular patterns and recursively mining their extensions.

While constructing the conditional pattern base from a projected database, we perform a super-pattern popularity check for extensions of any unpopular item, and delete the item only when it fails the check. We call such pruning technique the *lazy pruning*.

The lazy pruning technique ensures that no popular patterns (having unpopular subsets) will be missed by Pop-growth. The following example illustrates how Pop-growth mines popular patterns from the Pop-tree.

Example 6. Let us continue Example 5. In other words, let us mine popular patterns from the Pop-tree shown in Fig. 1(c) constructed for the TDB shown in Table 1 with *minpop*=2.4.

Recall that the Pop-growth recursively mines the projected databases of all items in H-table. Before constructing the projected database for an item x in H-table, we output the item as a popular pattern if its popularity is at least *minpop*. The conditional pattern base for the $\{g\}$-projected database (i.e., $DB_{\{g\}}$), as shown in Fig. 2(a), is constructed by accumulating the contents in the tree paths: $\langle b{:}1,5,5\quad c{:}1,5,5\quad h{:}1,5,5\quad f{:}1,5,5\rangle$ and $\langle c{:}1,4,4\quad e{:}1,4,4\quad h{:}1,4,4\rangle$. The header table for $DB_{\{g\}}$, as shown in Fig. 2(a), contains all items that co-occur with g in the Pop-tree. It also contains the corresponding *support*, $sumTL$ and $maxTL$ of each item in $DB_{\{g\}}$. We then compute the exact popularity of each item in $DB_{\{g\}}$ by using Eq. (4).

The conditional tree for any conditional pattern base of a pattern X may contain two types of items: (i) items that are popular in DB_X and (ii) items that are unpopular in DB_X but have potentially popular super-patterns. Other items are deleted from the projected database. To find unpopular items that having potentially popular super-patterns, we apply the lazy pruning technique and Eq. (8).

Based on Eq. (4), the popularity of items in the H-table of $DB_{\{g\}}$ can be computed: $Pop(\{b,g\}) = \frac{5}{1} - 2 = 3$, $Pop(\{f,g\}) = \frac{5}{1} - 2 = 3$, $Pop(\{c,g\}) = \frac{9}{2} - 2 = 2.5$, $Pop(\{g,h\}) = \frac{9}{2} - 2 = 2.5$ and $Pop(\{e,g\}) = \frac{4}{1} - 2 = 2$. All items except e are popular together with g. By applying the lazy pruning technique, the popularity upper bound $Pop^{UB}(\{e,g\})$ for e with g can be calculated as at most $4-2 = 2$, which is less than *minpop*. Hence, we can safely delete e from the projected database of g. The conditional tree for the projected database of g is presented in Fig. 2(b).

The mining for each extension (i.e., for f, h, c &b) of g is performed recursively. The set of patterns generated from the projected database of g is shown in Fig. 2(c). The mining process terminates when we reach the top of H-table of the Pop-tree. □

The Pop-growth mining technique is efficient because it applies a pattern-growth based mining technique on a Pop-tree. Moreover, the lazy pruning technique further reduces the mining cost for unpopular items whose super-patterns cannot be popular.

5 Discussion: An Application on Mining Popular Friends from Social Networks

In the previous section, we described how our proposed Pop-growth mines frequent patterns from transactional databases. This algorithm builds a Pop-tree structure to capture important contents of the transactional databases and recursively mines popular patterns from the Pop-tree. In this section, we extend the proposed technique and apply it to mine a special type of frequent patterns—popular friends—from social networks.

Recent advances in technology and successes in online digital media sites have led the surge of interest in social computing and its applications. Social computing enables users to intersect social behaviour with computing systems and to create social conventions as well as social contexts through the use of software and technology. This explains why, over the past few years, various research works on the analytics, mining and visualization of complex social networks have been proposed. In general, social networks are structures made of social entities (e.g., individuals, corporations, collective social units, or organizations) that are linked by some specific types of interdependencies (e.g., kinship, friendship, common interest, beliefs, or financial exchange). These dependencies among linked entities in the social networks present an opportunity to further infer different properties of individuals. Because a social entity is connected to another social entity as his next-of-kin, friend, collaborator, co-author, classmate, co-worker, team member and/or business partner, identifying social entities or groups of entities that have connections with a large number of other social entities may provide useful knowledge to the user. For example, among the friends of p, some of them may be very popular in the sense that they have many connections. Discovering these popular friends provides useful knowledge to p because they may have high social connectivity and/or could have strong influence on members of their social groups. Similarly, a newcomer (to a city, company, or profession) may want to be introduced to individuals having high social connectivity so that he can get to know more people faster. Similar comments apply to users in other social networking sites.

Note that the task of finding popular friends can be more complicated when we do not have access to these lists of connections. For example, due to various reasons (e.g., privacy setting), connection lists of some social entities in the social network may not be accessible to unauthorized users. Fortunately, members of interest groups (especially, open public groups) are usually visible. In these situations, given a social network containing all members of these interest groups, we can adapt our proposed Pop-growth to find popular users or a popular group of friends. Specifically, to adapt our proposed Pop-growth algorithm for mining popular friends from social networks, we treat (i) each interest group list like a transaction and (ii) each social user/member in an interest group list like an item in a transaction. With this adaption and setting of a user-specified *minpop*, we find popular friends from a sample social network as illustrated in Example 7.

Example 7. Consider a collection of n=7 interest groups involving m=9 users (namely, Alice, Bob, Cathy, Don, Ed, Fank, Gary, Helen, and Irene) as shown in Table 2, which may represent a subset of a large social network. To adapt our proposed Pop-growth algorithm for mining popular friends from social networks, we treat (i) each interest group list like a transaction and (ii) each social user/member in an interest group list like an item. With this adaption and setting of a user-specified *minpop*=2.4, we first scan the collection once to find *individually popular users*: Bob, Cathy, Ed, Frank, Gary, or Helen, with their popularity values of 3, 4, 3.33, 4.5, 3.5, or 4, respectively.

Table 2. A sample social network

Interest group list ID	Members in the interest group list
L_1 on ballet	{Bob, Don}
L_2 on baseball	{Bob, Cathy, Frank, Gary, Helen}
L_3 on curling	{Bob, Cathy, Don, Ed, Frank, Helen}
L_4 on football	{Cathy, Ed, Gary, Helen}
L_5 on hockey	{Alice, Don}
L_6 on lacrosse	{Alice, Bob, Irene}
L_7 on soccer	{Alice, Don, Ed}

Note that, among the m=9 users, only six of them are popular. As for the three unpopular users (Alice, Don, and Irene), their super-pattern may still be popular. Hence, we cannot delete them without performing the super-pattern popularity checks. The checks reveal that, when X={Alice}, the popularity upper bound $Pop^{UB}(X')$ of its extension X' (where $X' \supseteq$ {Alice}; e.g., X' = {Alice, Bob}) is at most 3−2=1 < *minpop*. Similarly, when X={Irene}, the popularity upper bound $Pop^{UB}(X')$ of its extension X' is also at most 3−2=1 < *minpop*. In contrast, when X={Don}, the popularity upper bound $Pop^{UB}(X')$ of its extension X' is at most 6−2=4 $\geq$ *minpop*. Hence, we keep Don but safely delete Alice and Irene.

To find popular groups of users, we build a Pop-tree by sorting and inserting the six individual popular users (Bob, Cathy, Ed, Frank, Gary, and Helen) together with this undeleted user (Don) into a header table (H-table) in descending order of their support values: ⟨Bob, Don, Cathy, Ed, Helen, Frank, Gary⟩. We then scan the collection of n=7 interest group lists the second time. During that, we compute the length of each interest group list (e.g., $|L_1| = 2, |L_3| = 6$), remove all uses who are not in the H-table (e.g., remove Alice from L_5 to make the resulting list become {Don}, remove both Alice and Irene from L_6 to make it become {Bob}, remove Alice from L_7 to make it become {Don, Ed}), and sort the remaining users in each interest group list according to the H-table order. When inserting users into a Pop-tree, if two interest group lists share a common prefix (e.g., Bob appears in both L_1 and L_2), then the prefix is merged. Figure 3 shows the Pop-tree after capturing all the interest group lists in the social collection.

Once the Pop-tree is built, we call Pop-growth to recursively mine the projected databases of all users in H-table. Before constructing the projected database for a user (e.g., {Gary}) in H-table, we output it as a popular user if its popularity is at least *minpop*. The conditional pattern base for the {Gary}-projected database (i.e., $DB_{\{\text{Gary}\}}$) is constructed by accumulating the contents in the tree path ⟨Bob:1,5,5 Cathy:1,5,5 Helen:1,5,5 Frank:1,5,5⟩ and ⟨Cathy:1,4,4 Ed:1,4,4 Helen:1,4,4⟩. The header table for $DB_{\{\text{Gary}\}}$ contains all users that share a common interest with Gary in the Pop-tree. It also contains the

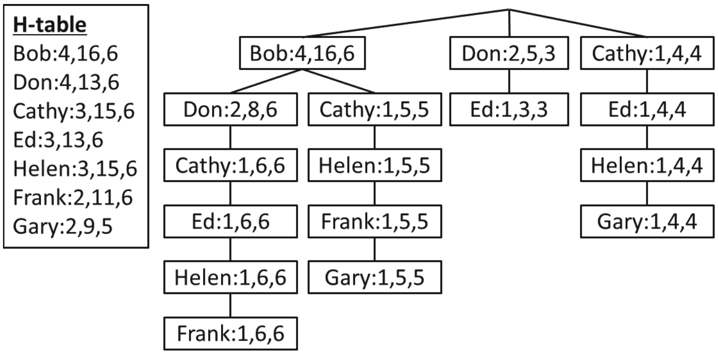


Fig. 3. The Pop-tree capturing interest group lists of a sample social network

corresponding *support*, *sumTL* and *maxTL* of each user in $DB_{\{\text{Gary}\}}$. We then compute the exact popularity of each item in $DB_{\{\text{Gary}\}}$ by using Eq. (4).

As the conditional tree for any conditional pattern base of a group X of users contains (i) those who are popular in DB_X and (ii) those who are unpopular in DB_X but have potentially popular super-groups, we apply the lazy pruning technique and Eq. (8) to prune out those unpopular users who have potentially popular super-groups. For example, we found popular groups {Bob, Gary} and {Frank, Gary} (both with popularity value of 3), as well as {Cathy, Gary} and {Gary, Helen} (both with popularity value of 2.5). Similar steps can be applied to other paths in a Pop-tree to find all other popular friends or groups of friends from social networks. □

6 Pop-Streaminng: Mining Popular Patterns from Dynamic Data Streams

In Sect. 4, we mined popular patterns from static transactional databases. The corresponding Pop-growth algorithm works well when handling static data. However, there are situations in which we need to deal with dynamic streaming data. In this section, we propose another algorithm for handling dynamic data. The corresponding algorithm—called **Pop-streaming**—mines popular patterns from dynamic data streams in a *landmark model* environment.

When using the landmark model for processing data streams, transactions in each batch (regardless of whether they are historical or recent data) are treated equally. As such, all batches (regardless of whether they are old or recent) are assigned the same weights. To mine popular patterns from dynamic data streams, our proposed Pop-streaming algorithm first calls Pop-growth (Sect. 4) to find popular patterns from the current batch of transactions in the streams (using a threshold called *preMinpop*, which is defined to be $\leq$ *minpop*). Note that, although users are interested in truly popular patterns (i.e., patterns with popularity $\geq$ *minpop* $>$ *preMinpop*), *preMinpop* is used in attempt to avoid pruning a pattern too early. This is important because data in the continuous streams are not necessarily uniformly distributed.

Once the popular patterns for a batch of streaming data are found, the next step is to construct a **Pop-stream structure** to capture the mined popular patterns. Each node in this tree-based Pop-stream structure corresponds to a popular pattern. Nodes that correspond to the popular patterns sharing common prefix are merged. In addition to the popular pattern X (or more precisely, the suffix item in X), each node stores additional information. So, on the surface, this Pop-stream structure may seem to be similar to that of the UF-stream structure [21] used in frequent pattern mining. As such, it was tempting to keep X and its popularity value $Pop(X)$ in each node. However, a closer look reveals that, while frequency (or support) of patterns is additive, popularity of patterns is *not*. See Example 8.

Table 3. A data stream

Batch ID	Transaction ID	Transaction
B_1	t_1	$\{b, d\}$
	t_2	$\{b, c, f, g, h\}$
B_2	t_3	$\{b, c, d, e, f, h\}$
	t_4	$\{c, e, g, h\}$

Example 8. Consider two batches of streaming data as shown in Table 3. The support of $\{c, h\}$ in Batch B_1 is 1, and that in Batch B_2 is 2. So, $sup(\{c, h\}, B_1 \cup B_2) = sup(\{c, h\}, B_1) + sup(\{c, h\}, B_2) = 1 + 2 = 3$. However, the popularity of $\{c, h\}$ in Batch B_1 is $\frac{1}{|\{t_2\}|}Pop(\{c, h\}, t_2) = 3$, and that in Batch B_2 is $\frac{1}{|\{t_3,t_4\}|}(Pop(\{c, h\}, t_3) + Pop(\{c, h\}, t_4)) = \frac{1}{2}(4 + 2) = 3$. So, the sum of these two popularity values becomes 3+3 = 6, which is *not* equal to the popularity of $\{c, h\}$ in these two batches. Mathematically, $Pop(\{c, h\}, B_1 \cup B_2) = \frac{1}{|\{t_2,t_3,t_4\}|}(Pop(\{c, h\}, t_2)+Pop(\{c, h\}, t_3)+Pop(\{c, h\}, t_4)) = \frac{1}{3}(3+4+2) = 3.\square$

Recall from Definition 5 that the popularity $Pop(X)$ of a pattern X can be computed in terms of (i) $sumTL(X)$ and (ii) $sup(X, B_i)$. Moreover, both (i) $sumTL(X)$ and (ii) $sup(X, B_i)$ are additive. For example, $sumTL(\{c, h\})$ in B_1 is 5, $sumTL(\{c, h\})$ in B_2 is 6+4 = 10, whereas $sumTL(\{c, h\})$ in the first two batches is 5+(6+4) = 15. Similarly, $sup(\{c, h\}, B_1)$ is 1, $sup(\{c, h\}, B_2)$ is 2, whereas $sup(\{c, h\}, B_1 \cup B_2)$ is 1+2 = 3. Hence, instead of storing the popularity value of a popular pattern, we store (i) $sumTL(X)$ and (ii) $sup(X, B_i)$ values so that we can compute the popularity of X based on these two values.

As we are dealing with batches of streaming data, the Pop-stream structure needs to be updated. Hence, we need to store multiple pairs of $sumTL(X)$ and $sup(X, B_i)$ values (one pair for each batch) In other words, we need to store w pairs of $sumTL(X)$ and $sup(X, B_i)$ values when handling w batches of streaming data Fortunately, when using the landmark model, all data are of the same weights. Hence, we only need keep *one* pair of $sumTL(X)$ and $sup(X, B_i)$ values for each node representing a popular pattern X:

$$\text{(i) } X, \text{ (ii) } sumTL(X), \text{ and (iii) } sup(X, \bigcup_i B_i).$$

When a new batch B_j flows in, if X does not exist in the Pop-stream structure, our Pop-streaming algorithm inserts $\langle X, sumTL(X), sup(X, B_j)\rangle$ into the Pop-stream structure. Otherwise (i.e., X exists in the Pop-stream structure), we need to update the stored information as follows:

1. add the new $sumTL(X)$ to the existing $sumTL(X)$, and
2. add the new $sup(X, B_j)$ to the existing $sup(X, \bigcup_{i=1}^{j-1} B_i)$.

This insertion (of new popular patterns) and update (of existing popular patterns) step is repeated for each batch.

Note that, during the mining process, our proposed Pop-streaming algorithm updates the $sumTL(X)$ and $sup(X)$ values stored in the Pop-stream structure whenever a batch of streaming data flows in. However, the algorithm does not repeatedly update $Pop(X)$. It uses the delay mode for mining: It only computes $Pop(X)$ based on the updated $sumTL(X)$ and $sup(X)$ values when the user needs the results. See Fig. 4 for a skeleton of the Pop-streaming algorithm.

```
Algorithm Pop-streaming
1. For each batch $B_j$ do
    2. Apply Pop-growth to $B_j$ to find popular patterns from $B_j$.
    3. Insert each mined popular pattern $X$ into the Pop-stream structure and update
       its $sumTL(X)$ and $sup(X)$ values:
        3a. If the nodes corresponding to $X$ do not exist in the Pop-stream struc-
            ture, then create new nodes (each of which keeps $sumTL(X)$ and $sup(X)$
            values) for $X$;
        3b. else, add its $sumTL(X)$ and $sup(X)$ values in $B_j$ to the existing
            $sumTL(X)$ and $sup(X)$ values.
4. When a user requests the mining results (i.e., popular patterns) do
    5. Compute $Pop(X)$ based on the updated $sumTL(X)$ and $sup(X)$ values stored
       in the Pop-stream structure .
    6. If $Pop(X) \geq minpop$, then return $X$ to the user.
```

Fig. 4. A skeleton of the Pop-streaming algorithm

7 Experimental Results

For experiments, we mostly use those datasets commonly used in frequent pattern mining experiments because characteristics of those transactional datasets are well known (see Table 4). More specially, we used (i) IBM synthetic datasets (e.g., T10I4D1M, T10I4D100K, T20I4D100K). and (ii) real datasets (e.g., chess, mushroom, connect-4) from the Frequent Itemset Mining Dataset Repository http://fimi.ua.ac.be/data/. We obtained consistent results for all of these datasets. Hence, to avoid repetition, we report here the experimental results on only a subset of these datasets in the remainder of this section.

Table 4. Dataset characteristics

Dataset	#transactions	#items	maxTL	avgTL	Data density
T10I4D100K	100,000	870	29	10.10	Sparse
T20I4D100K	99,996	871	42	19.81	Sparse
mushroom	8,124	119	23	23.00	Dense

All programs were written in C and run on UNIX with a quad-core processor with 1.3 GHz. The runtime specified indicates the total execution time (i.e., CPU and I/Os). The reported results are based on the average of multiple runs for each case. In all of the below experiments, Pop-trees were constructed using descending order of occurrence counts of items.

To the best of our knowledge, our Pop-tree is the first approach to mine popular patterns from transactional databases. Here, we first present the performance of our Pop-tree structure and Pop-growth algorithm when varying the mining parameters such as popularity threshold and dataset characteristics.

7.1 Runtime of Pop-Growth

In this section, we report the execution time that the Pop-growth requires for mining popular patterns over datasets of different types and changes in *minpop*. The execution time includes all the steps of H-table construction, the Pop-tree building and the corresponding mining. The results on one sparse dataset (e.g., T20I4D100K) and one dense dataset (e.g., mushroom) are presented in Fig. 5.

To observe the effect of mining on the variation in size of such datasets, we performed popular pattern mining while increasing the size of both of the

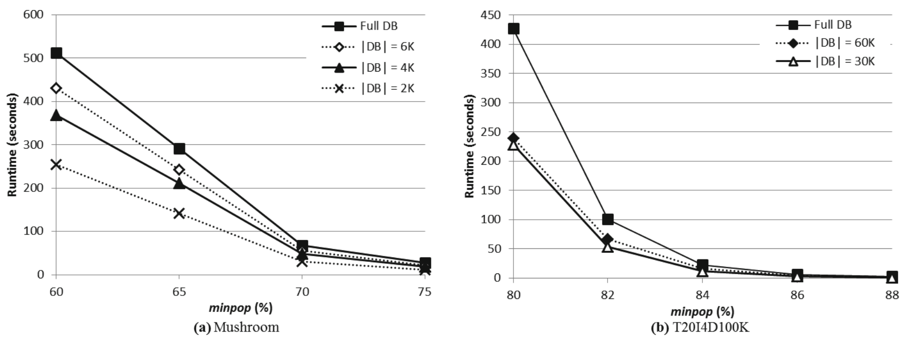
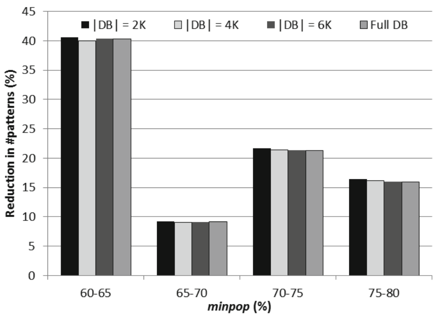


Fig. 5. Runtime of Pop-growth in mining transactional databases

datasets: (i) From 2 K to full for the mushroom dataset and (ii) from 30 K to full for T20I4D100K. Thus, the series for "Full DB" represent the results for the full size of datasets. Both datasets required more execution time when mining larger datasets. As the database size increased and *minpop* decreased, the tree structure size and number of popular patterns increased. Hence, a comparatively longer time was required to generate large numbers of popular patterns from large trees. Although the mushroom dataset is smaller in size, the transaction lengths of all transactions are the same (i.e., 23). Hence, the Pop-tree mining took a longer time when compared to a dataset with variable length such as T20I4D100K. The experimental results show that mining the corresponding Pop-tree for popular patterns is time efficient for both sparse and dense datasets.

7.2 Reduction on the Number of Patterns When Changing *minpop*

Similar to the previous experiment, we also examined the number of patterns generated by our Pop-growth when we varied the dataset size and *minpop*. Figure 6 shows the reduction in the number of patterns in percentage when increasing the *minpop* values in both the mushroom and T20I4D100K datasets with different dataset size. Each data point in the x-axes of the graphs reports the change of *minpop* from a low to a high value, while the y-axes indicate the percentage change in the number of patterns generated from a low to a high *minpop* value.



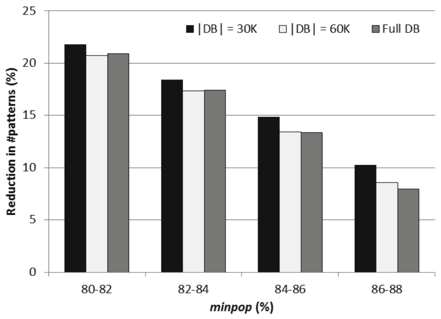


Fig. 6. Reduction on the number of patterns when changing *minpop*

Note that, depending on dataset characteristics, the reduction rate varied. For example, for the mushroom dataset, the reduction rate dropped sharply when *minpop* was changed from 60 %–65 % to 65 %–70 %, but the reduction rate rose when *minpop* was changed to 70 %–75 %. In contrast, T20I4D100K showed a consistent reduction rate when lowering the *minpop* value. However, as observed from the graphs for both datasets, the number of patterns reduced when increasing the *minpop* values. For example, for the mushroom dataset, the reduction rate was around 40 % when increasing the threshold from 60 % to 65 %. For 30 K of T20I4D100K, the reduction rate was around 21 % when increasing

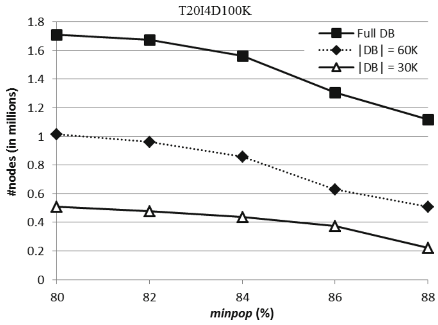


Fig. 7. Compactness of the Pop-tree: node count on T20I4D100K

the threshold from 80 % to 82 %. It is also interesting to note that the pattern count reduction rate was very similar irrespective of the database size.

We observed that the pattern generation characteristics of the proposed popular pattern mining algorithm were consistent with the variation of *minpop* and database size.

7.3 Compactness of the Pop-Tree

Here, we report the compactness of a Pop-tree in terms of number of Pop-tree nodes. Note that, as the mushroom dataset has a fixed transaction length, the maximum transaction length for every possible pattern in the dataset is always the same. Consequently, every item in the dataset passes the lazy pruning phase and contributes to the tree. Hence, for a particular portion of the mushroom dataset, the tree size (i.e., number of nodes) is the same with the variation of *minpop*. However, the number of nodes varied from 34523 (when $|TDB| = 2\,K$) to 91338 (when $|TDB| = 6\,K$). For the full dataset, it is around 100 K. The compactness of Pop-tree on different portion of T20I4D100K is presented in Fig. 7. The size of the tree structure gradually reduced in T20I4D100K with the increase of *minpop*.

As expected, in both datasets, the number of nodes increased with the increase in size of database. However, as far as the total number of nodes is concerned, one can observe that, irrespective of fixed or variable transaction length, a Pop-tree structure is compact enough to fit into a reasonable amount of memory.

7.4 Scalability of Pop-Growth

To study the scalability of Pop-growth mining technique, we further ran our algorithm on T10I4D100K, which is sparser than T20I4D100K. Figure 8 presents the results on scalability tests on the variation of *minpop* and required number of nodes on the dataset. Clearly, as the *minpop* decreases, the overall tree construction and mining time (Fig. 8(a)), and required memory (Fig. 8(b)) increase.

However, the Pop-tree shows a stable performance with a linear increase in runtime and memory consumption as the *minpop* decreased for the dataset. Moreover, the results demonstrate that, the Pop-tree can mine the set of popular patterns on this dataset for a reasonably small value of popularity threshold with a considerable amount of execution time and memory.

To recap, the above experimental results show that the proposed Pop-tree can mine the set of popular patterns in both time and memory efficient manner over different types of dataset. Furthermore, the Pop-tree structure and Pop-growth algorithm are scalable for popularity threshold values and memory.

7.5 Mining Popular Friends from Social Networks

The aforementioned results show the time-efficiency of our proposed Pop-growth algorithm and the space-efficiency of our proposed Pop-tree structure for mining popular patterns from transactional data. Here, we experimented the efficiency of the Pop-growth algorithm when adapted to mine popular friends from social networks. To conduct this experiment, we used the social network datasets (e.g., Facebook, Twitter) from Stanford Large Network Dataset Collection https://snap.stanford.edu/data/. For example, when *minpop* was set to 1043, Pop-growth only took 43 s to find about 384 K popular friend groups from the Facebook dataset (where $maxTL = 1045$). As another example, when *minpop* was set to 1203, Pop-growth took 70 s to find more (e.g., around 484 K) popular friend groups from another dataset—namely, the Twitter dataset (where $maxTL$ is higher and with a length of 1205).

7.6 Runtime of Pop-Streaming

After performing a series of experiments on popular pattern mining from static transactional databases or static social networks, we conduct experiments on popular pattern mining from dynamic data streams. Here, we divided datasets into multiple batches. We report the execution time that the Pop-streaming

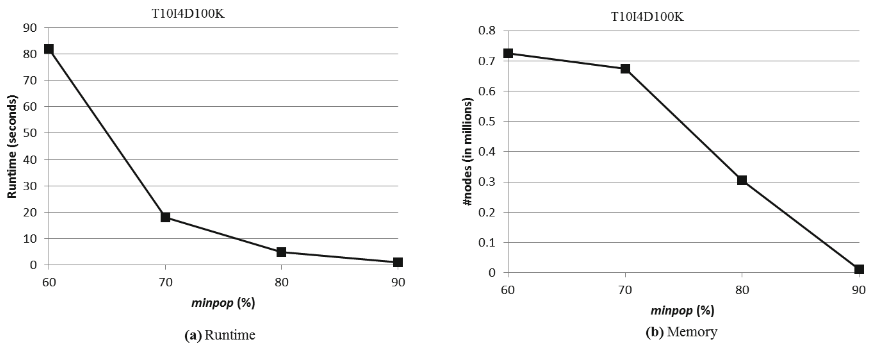


Fig. 8. Scalability on Pop-growth

requires for mining popular patterns over batches of streaming data of different types and changes in *minpop*. The execution time includes all the steps of the application of Pop-growth, the construction of the Pop-stream structure, and the corresponding mining. The results on dense streams (e.g., mushroom) and sparse streams (e.g., IBM) are presented in Fig. 9. Consistent with the runtime results on transactional database mining, Pop-streaming required shorter runtimes when *preMinpop* increased.

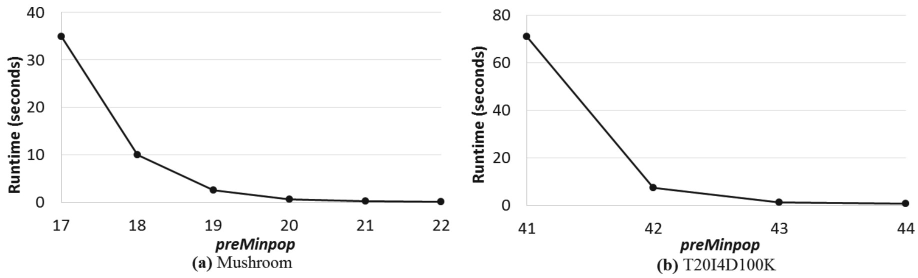


Fig. 9. Runtime of Pop-streaming in mining batches of data streams

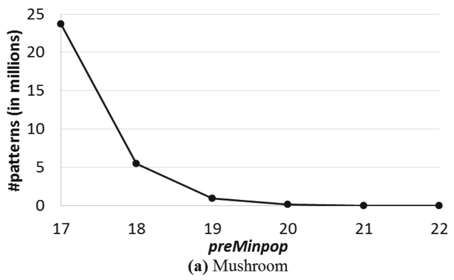
7.7 Compactness of the Pop-Stream Structure

Next, we report the compactness of the Pop-stream structure in terms of number of nodes, which corresponds to the number of popular patterns mined from batches of streaming data. Figure 10 shows that, when *preMinpop* increased, the number of mined popular patterns decreased and thus reducing the size (i.e., reducing the number of nodes) of the Pop-stream structure.

7.8 Memory Consumption for the Pop-Streaming Algorithm

Recall that the first step of the Pop-streaming algorithm is to call Pop-growth for finding popular patterns from each batch of streaming data. When the Pop-growth algorithm is called, it builds a Pop-tree to capture important contents of transactions in the batch. When mining from w batches of the streaming data, the Pop-growth algorithm may be called w times. The size of the ing Pop-tree may vary from one batch to another batch. Figure 11 shows the maximum memory consumption among w Pop-trees.

Once the popular patterns are mined from a batch of streaming data, these mined patterns are then stored in the Pop-stream structure. Recall that, in Sect. 7.7, we measured the compactness of the Pop-stream structure. Note that memory consumption of the Pop-streaming algorithm mainly depends on that of the Pop-tree (measured in this section) and that of the Pop-stream structure (measured in Sect. 7.7). Hence, based on the experimental results from these two sections, we gained some insight about the amount of memory space required by the Pop-streaming mining process.



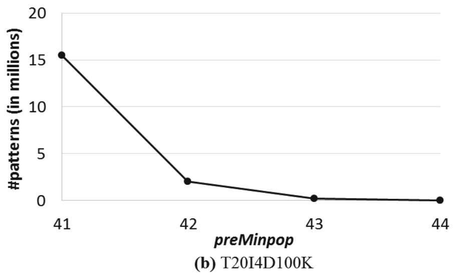
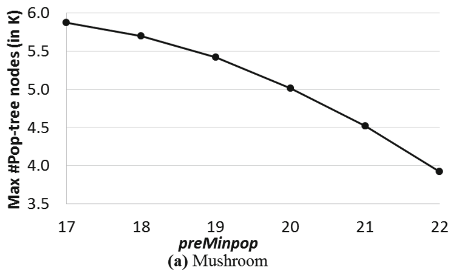


Fig. 10. Compactness of the Pop-stream structure: node count



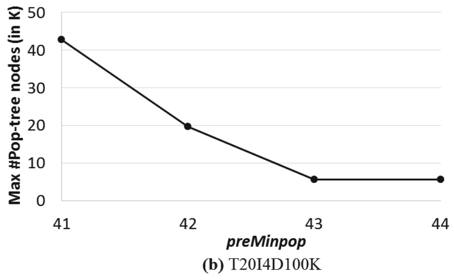


Fig. 11. Compactness of the Pop-tree structure in mining data streams: node count

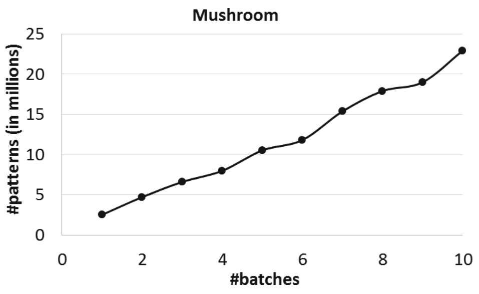


Fig. 12. Compactness of the Pop-stream structure when varying #batches

7.9 Scalability of the Pop-Stream Structure

Finally, we studied the scalability of the Pop-stream structure. In particular, Fig. 12 shows that, when the number of batches increased, the number of patterns to be stored in the Pop-stream structure gradually increased. Hence, the total number of nodes in the Pop-stream structure increased accordingly. The increase in the number of mined popular patterns (i.e., in the number of stored nodes) is proportional to the number of batches in the streaming data.

8 Conclusions and Future Work

In this paper, we introduced a new type of patterns—namely, *popular patterns.* We also proposed the *Pop-tree* (which captures important contents of transactional databases for mining popular patterns) and the *Pop-growth* algorithm (which finds popular patterns by mining the Pop-tree). Although the notion of popularity does not satisfy the downward closure property, we managed to address this issue by using total transaction length ($sumTL$) together with projected databases, which allows lazy pruning. Moreover, we also proposed the *Pop-stream* structure (which captures the popular patterns mined from each batch as well as other auxiliary information for computing the popularity of these patterns) and the *Pop-streaming* algorithm (which finds popular patterns by mining the Pop-stream. Moreover, results also showed that construction of Pop-tree and mining of popular patterns are time efficient. Furthermore, we are not confined with mining popular patterns from static transactional databases; we also mine popular patterns from dynamic data streams. Experimental results showed that both Pop-tree and Pop-stream structures are compact, scalable, and space efficient for both sparse and dense datasets (e.g., IBM synthetic data, real data from FIMI, social network data).

As future work, we plan to further extend our proposed framework as to incorporate novel extensions, precisely targeting several achievements: (i) incorporating the capability of dealing with *Big Data* (e.g., [8]), perhaps by adopting consolidated *data fragmentation approaches* (e.g., [3]), which well-adapt to massive sizes that both transactional databases and multi-rate, heterogeneous data streams may achieve; (ii) incorporating the capability of dealing with *optimization issues* (e.g. [5,6]), perhaps by adopting non-conventional approaches like *topology control* (e.g., [13]), which well-adapts to the graph-based nature of both connected entities in transactional databases and data stream items; (iii) incorporating the capability of dealing with *uncertain and imprecise* transactional databases (e.g., [27]) and data streams (e.g., [4]), perhaps by adopting probabilistic methods (e.g., [7]).

Acknowledgement. This project is partially supported by (i) China Scholarship Council (CSC), (ii) Mitacs (Canada), (iii) Natural Sciences and Engineering Research Council of Canada (NSERC), and (iv) University of Manitoba.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: VLDB 1994, pp. 487–499 (1994)
2. Bailey, J., Manoukian, T., Ramamohanarao, K.: Fast algorithms for mining emerging patterns. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) PKDD 2002. LNCS (LNAI), vol. 2431, pp. 39–50. Springer, Heidelberg (2002)
3. Bonifati, A., Cuzzocrea, A.: Storing and retrieving XPath fragments in structured P2P networks. Data Knowl. Eng. **59**(2), 247–269 (2006)

4. Cuzzocrea, A.: Retrieving accurate estimates to OLAP queries over uncertain and imprecise multidimensional data streams. In: Cushing, J.B., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 575–576. Springer, Heidelberg (2011)
5. Cuzzocrea, A., Furfaro, F., Greco, S., Masciari, E., Mazzeo, G.M., Saccà, D.: A distributed system for answering range queries on sensor network data. In: IEEE PerCom 2005 Workshops, pp. 369–373 (2005)
6. Cuzzocrea, A., Furfaro, F., Masciari, E., Saccà, D., Sirangelo, C.: A distributed system for answering range queries on sensor network data. In: Stefanidis, A., Nittel, S. (eds.) GeoSensor Networks, pp. 53–72. CRC Press (2004)
7. Cuzzocrea, A., Gunopulos, D.: A decomposition framework for computing and querying multidimensional OLAP data cubes over probabilistic relational data. Fundamenta Informaticae **132**(2), 239–266 (2014)
8. Cuzzocrea, A., Saccà, D., Ullman, J.D.: Big data: a research agenda. In: IDEAS 2013, pp. 198–203. ACM (2013)
9. Cameron, J.J., Leung, C.K.-S., Tanbeer, S.K.: Finding strong groups of friends among friends in social networks. In: IEEE DASC 2011, pp. 824–831 (2011)
10. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM 2006, pp. 328–339. SIAM (2006)
11. Castellanos, M., Gupta, C., Wang, S., Dayal, U.: Leveraging web streams for contractual situational awareness in operational BI. In: EDBT/ICDT 2010 Workshops, art. 7. ACM (2010)
12. Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.H.: SpADe: on shape-based pattern detection in streaming time series. In: IEEE ICDE 2007, pp. 786–795 (2007)
13. Cuzzocrea, A., Papadimitriou, A., Katsaros, D., Manolopoulos, Y.: Edge betweenness centrality: a novel algorithm for QoS-based topology control over wireless sensor networks. J. Netw. Comput. Appl. **35**(4), 1210–1217 (2012)
14. Gaber, M.M., Zaslavsky, A.B., Krishnaswamy, S.: Mining data streams: a review. SIGMOD Rec. **34**(2), 18–26 (2005)
15. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining frequent patterns in data streams at multiple time granularities. In: Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y. (eds.) Data Mining: Next Generation Challenges and Future Directions, pp. 105–124. AAAI/MIT Press (2004)
16. Gupta, A., Bhatnagar, V., Kumar, N.: Mining closed itemsets in data stream using formal concept analysis. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2010. LNCS, vol. 6263, pp. 285–296. Springer, Heidelberg (2010)
17. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1–12 (2000)
18. Jiang, N., Gruenwald, L.: Research issues in data stream association rule mining. SIGMOD Rec. **35**(1), 14–19 (2006)
19. Lakshmanan, L.V.S., Leung, C.K.-S., Ng, R.T.: Efficient dynamic mining of constrained frequent sets. ACM Trans. Database Syst. **28**(4), 337–389 (2003)
20. Lee, Y.-K., Kim, W.-Y., Cai, Y.D., Han, J.: CoMine: efficient mining of correlated patterns. In: IEEE ICDM 2003, pp. 581–584 (2003)
21. Leung, C.K.-S., Cuzzocrea, A., Jiang, F.: Discovering frequent patterns from uncertain data streams with time-fading and landmark models. T. Large-Scale Data- and Knowl.-Centered Syst. **8**, 174–196 (2013)
22. Leung, C.K.-S., Hao, B.: Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE 2009, pp. 1663–1670 (2009)
23. Leung, C.K.-S., Jiang, F.: Frequent pattern mining from time-fading streams of uncertain data. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2011. LNCS, vol. 6862, pp. 252–264. Springer, Heidelberg (2011)

24. Leung, C.K.-S., Sun, L.: A new class of constraints for constrained frequent pattern mining. In: ACM SAC 2012, pp. 199–204 (2012)
25. Leung, C.K.-S., Tanbeer, S.K.: Mining popular patterns from transactional databases. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 291–302. Springer, Heidelberg (2012)
26. Leung, C.K.-S., Tanbeer, S.K.: Mining social networks for significant friend groups. In: Yu, H., Yu, G., Hsu, W., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA Workshops 2012. LNCS, vol. 7240, pp. 180–192. Springer, Heidelberg (2012)
27. Motro, A.: Imprecision and uncertainty in database systems. In: Base, P., Kacprzyk, J. (eds.) Fuzziness in Database Management Systems. pp. 3–22. Physica-Verlag (1995)
28. Ng, W., Dash, M.: Discovery of frequent patterns in transactional data streams. T. Large-Scale Data- and Knowl.-Centered Syst. **2**, 1–30 (2010)
29. Rasheed, F., Alshalalfa, M., Alhajj, R.: Efficient periodicity mining in time series databases using suffix trees. IEEE Trans. Knowl. Data Eng. **23**(1), 79–94 (2011)
30. Rashid, M.M., Karim, M.R., Jeong, B.-S., Choi, H.-J.: Efficient mining regularly frequent patterns in transactional databases. In: Lee, S., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012, Part I. LNCS, vol. 7238, pp. 258–271. Springer, Heidelberg (2012)
31. Wu, T., Chen, Y., Han, J.: Re-examination of interestingness measures in pattern mining: a unified framework. Data Min. Knowl. Disc. **21**(3), 371–397 (2010)
32. Xiong, H., Tan, P.-N., Kumar, V.: Hyperclique pattern discovery. Data Min. Knowl. Disc. **13**(2), 219–242 (2006)
33. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data Knowl. Eng. **59**(3), 603–626 (2006)
34. Zhang, M., Kao, B., Cheung, D.W., Yip, K.Y.: Mining periodic patterns with gaprequirement from sequences, ACM Trans. Knowl. Discov. Data **1**(2), art. 7 (2007)