Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 57 (2006) 240–282
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Efficient mining of group patterns from user movement data

Yida Wang [a], Ee-Peng Lim [a,*], San-Yih Hwang [b]

[a] *Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Blk N4, 2a-32, Nanyang Avenue, Singapore 639798, Singapore*
[b] *Department of Information Management, National Sun Yat-Sen University, Kaohsiung 80424, Taiwan*

Received 3 February 2005; received in revised form 3 February 2005; accepted 27 April 2005
Available online 31 May 2005

Abstract

In this paper, we present a new approach to derive groupings of mobile users based on their movement data. We assume that the user movement data are collected by logging location data emitted from mobile devices tracking users. We formally define *group pattern* as a group of users that are within a distance threshold from one another for at least a minimum duration. To mine group patterns, we first propose two algorithms, namely AGP and VG-growth. In our first set of experiments, it is shown when both the number of users and logging duration are large, AGP and VG-growth are inefficient for the mining group patterns of size two. We therefore propose a framework that summarizes user movement data before group pattern mining. In the second series of experiments, we show that the methods using location summarization reduce the mining overheads for group patterns of size two significantly. We conclude that the cuboid based summarization methods give better performance when the summarized database size is small compared to the original movement database. In addition, we also evaluate the impact of parameters on the mining overhead.

Keywords: Group pattern mining; Mobile data mining; Location summarization

* Corresponding author. Tel.: +65 852 2609 8246; fax: +65 852 2603 5505.
E-mail addresses: wyd66@pmail.ntu.edu.sg (Y. Wang), aseplim@ntu.edu.sg (E.-P. Lim), syhwang@mis.nsysu.edu.tw (S.-Y. Hwang).

0169-023X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2005.04.006

1. Introduction

1.1. Group pattern mining

People form groups due to many different reasons. Within an organization, formal groups are formed to carry out some designated tasks or assignments. Group members have well-defined roles and the groups may exist till the tasks or assignments are completed. Informal groups, on the other hand, are formed due to emotional, social or psychological needs. Group members have less defined roles and the memberships are less stable. Group dynamics and its influence on individual decision making have been well studied by sociologists [7], and it has been shown that peer pressure and group conformity can affect the behaviors of individuals. Such group behaviors can be very useful to different applications. For example, by knowing the groups a customer belongs to, retailers can derive common buying interests among customers and develop group-specific pricing models and marketing strategies. Group discounts and product recommendation can be introduced to encourage more purchases that lead to higher sales. In fighting against terrorism, analyzing user group patterns is one of the important tasks that help to reveal the links between terrorists and their roles in the group.

In the past, different ways to discover groups using clustering techniques have been proposed [22]. Very often, they are based on different definitions of similarity measure to represent the closeness between users. For example, users may be grouped by their common interests, job features, education level, and other attributes. Users can also be grouped based on the transactions they perform. For example, Amazon.com groups users together by the common books they purchase. However, in many cases, these methods suffer from a common pitfall, i.e., members in a derived group may not even know one another. Such kind of group derivation approaches are therefore not suitable to many applications that require group members to be acquaintances.

In our research, we propose a new way to derive grouping knowledge by performing data mining on user movement log data. These movement data are assumed to be generated by mobile devices that track the locations of their owners as they move from one place to another. These devices are equipped with GPS (Global Positioning Systems) and other related positioning technologies. GPS can achieve positioning errors ranging from 10 to 20 m [5,6,32], while the Assisted-GPS technology further reduces errors to between 1 and 10 m [8]. There are also terrestrial-based positioning technologies on the popularly used cellular networks, such as AOA, DOA, and TOA, which can achieve positioning accuracy around 50–100 m [33]. In the indoor environment, users can also be tracked by RFIDs by having RFID receivers at different locations sensing the signals from RFID tags.

We also assume that each user location, in the form of x-/y-/z-coordinates, can be logged at regular intervals over a period of time. In practice, the assumption may not hold as mobile devices may experience failures. They may be switched off by their owners from time to time, and the data collection time may not be synchronized across users. These assumptions nevertheless are reasonable if considering data cleaning or data transformation to be performed before applying our proposed mining algorithms. To keep a focused discussion, we shall keep the privacy and legal issues out the scope of this paper. As logging user locations over time can affect the privacy of users, we believe that these issues should be addressed within a legal framework which is beyond the scope of this paper. Furthermore, for several practical situations related to safety and security, user movement logging is considered necessary and has already been done.

Mining user groups from movement data is a kind of spatio-temporal data mining. That is, we are interested to discover groupings of users such that members in the same group are spatially close to one another for significant amount of time. Such user groupings are also known as *group patterns* [27]. The grouping knowledge derived in this way is unique compared to the other approaches due to the following:

- *Physical proximity between group members:* The group members are expected to be physically close to one another when they act as a group. Such characteristics are common among many types of groups, e.g., shopping pals, game partners, etc.
- *Temporal proximity between group members:* The group members are expected to stay together for some meaningful duration when they act as a group. Such characteristics distinguishes an ad hoc cluster of people who are physically close but unaware of one another from a group of people who come together for some planned activity(ies).

Intuitively, people who spent significant time together are expected to be *aware* of one another and they should maintain regular *contact*. Hence, the group members are expected to exert much stronger influence on one another.

1.2. Research objectives and contributions

Our research aims to formalize the concept of group pattern based on user movement. In this paper, we formally define the notion of *group pattern*. We introduce *max_dis* and *min_dur* as the maximum physical distance threshold among group members and the minimum duration threshold for group members to stay together respectively for deriving group patterns. In addition, we define the *weight* of a group pattern as a measure of its interestingness. With a *min_wei* threshold, we define the notion of *valid group pattern* and the valid group pattern mining problem.

In the following, we summarize our main research contributions as follows:

- *Algorithms for valid group pattern mining*. Two algorithms AGP and VG-growth are developed to mine valid group patterns. While the AGP algorithm is derived from the Apriori algorithm for classical association rule mining, VG-growth adopts a mining strategy similar to FP-growth algorithm and is based on a novel data structure known as *VG-graph*.
- *Location summarization based algorithms for mining valid 2-groups*. We observe in our experiments that the time taken by AGP and VG-growth to mine valid group patterns of size two (also known as valid 2-groups) dominates the total mining time, because both algorithms require large number of user pairs to be examined, especially when the number of users is large. We therefore propose a group pattern mining framework that can accommodate different location summarization methods. Four different location summarization methods have been proposed to reduce the overhead of mining valid 2-groups.
- *Performance evaluation of group pattern mining algorithms*. We conduct comprehensive experiments to evaluate the performance of all the proposed algorithms using datasets synthetically generated by IBM City Simulator [14]. We observe that VG-growth is much faster than AGP for mining valid k-groups, where $k > 2$, while the location summarization based algorithms are much more efficient than AGP and VG-growth for mining valid 2-groups.

In our previous work [27,28], we have proposed our novel algorithms AGP and VG-growth for mining valid group patterns and one location summarization method SLS for efficiently mining valid 2-groups. In this paper, we provide the formal definition of conditional VG-graph and give the correctness and completeness proofs of the VG-growth algorithm. Several other location summarization methods are also included to further reduce the overhead of mining valid 2-groups. Comprehensive experiments are conducted to compare the performances of different location summarization based algorithms and the influences of relevant parameters.

1.3. Paper outline

The rest of the paper is organized as follows. The formal definitions of group pattern mining problem is given in Section 2. Section 3 describes two valid group pattern mining algorithms, AGP and VG-growth, and their performance results. In Section 4, we introduce a general framework to incorporate location summarization into valid 2-group mining. Our location summarization methods are introduced in Section 5. In Section 6, we present an experimental study on the location summarization based algorithms. We look at some related work in Section 7. Finally, we draw conclusion in Section 8.

2. Problem definition

2.1. Preliminaries

Group pattern mining is to be conducted on a user movement database defined by $D = (D_1, D_2, \ldots, D_M)$, where D_i is a time series of tuples $(t, (x, y, z))$ denoting the x-, y- and z-coordinates of user u_i at time t. We assume that there are N time points in the time series ranging from 0 to $N-1$. For simplicity, we denote the location of a user u_i at time t by $u_i[t].p$, and his/her x-, y-, and z-values at time t by $u_i[t].x$, $u_i[t].y$ and $u_i[t].z$ respectively. A very small user movement database example is shown in Table 1.

Definition 1. Given a set of users G, a maximum distance threshold *max_dis*, and a minimum time duration threshold *min_dur*, a set of consecutive time points $[t_a, t_b]$ is called a *valid segment* of G, if

(1) $\forall u_i, u_j \in G$, $t_a \leqslant t \leqslant t_b$, $d(u_i[t].p, u_j[t].p) \leqslant max\_dis$.
(2) If $t_a > 0$, $\exists u_i, u_j \in G$, $d(u_i[t_a - 1].p, u_j[t_a - 1].p) > max\_dis$.
(3) If $t_b < N - 1$, $\exists u_i, u_j \in G$, $d(u_i[t_b + 1].p, u_j[t_b + 1].p) > max\_dis$.
(4) $(t_b - t_a + 1) \geqslant min\_dur$.

In other words, within a valid segment of a set of users G, all members must be close to one another for at least a minimum time duration (*min_dur*). The function, $d()$, returns the distance between two points.[1] Furthermore, valid segments are maximal as no two valid segments of

[1] In this paper, Euclidean distance is adopted. However, other distance metrics can be used for different applications, such as Manhattan distance and *weighted* Euclidean distance, as long as they satisfy the following four properties: (1) $d(i,j) \geqslant 0$; (2) $d(i,i) = 0$; (3) $d(i,j) = d(j,i)$; and (4) $d(i,j) \leqslant d(i,h) + d(h,j)$.

Table 1
User movement database D

t	x	y	z
u_1			
0	92.6	22.36	7.92
1	93.44	23.12	7.92
2	97.12	19.88	7.92
3	94.52	20.2	9.06
4	94.52	20.2	9.06
5	95.12	22.84	9.06
6	95.68	22.44	9.06
7	97.72	26.16	9.06
8	94.16	27.88	9.06
9	90.2	31.72	9.06
u_2			
0	97.36	28.56	5.66
1	94.6	24.68	5.66
2	92.08	22.88	5.66
3	93.8	25.88	5.66
4	96.72	29.8	5.66
5	97.32	30	5.66
6	93.84	32.48	5.66
7	92.92	30.52	5.66
8	95.68	29.12	5.66
9	95.56	29.2	5.66
u_3			
0	94.36	12.96	10.4
1	95.4	13.28	10.4
2	92.52	13.04	10.4
3	96.04	14.56	10.4
4	96.96	13.56	10.4
5	97.68	12.04	10.4
6	98.56	15.64	10.4
7	100.16	17.04	10.4
8	100.16	17.04	10.4
9	98.2	17.32	10.4
u_4			
0	91.2	31.12	9.06
1	90	31.2	9.06
2	91.16	28.6	7.92
3	92.36	29.32	7.92
4	93.12	26.12	7.92
5	95.28	26.24	7.92
6	97.12	29.08	7.92
7	97.44	29.04	7.92
8	98.92	25.4	7.92
9	96.64	25.4	7.92

Table 1 (*continued*)

t	x	y	z
u_5			
0	102.16	27.32	7.92
1	102.16	27.32	7.92
2	101.96	26.44	7.92
3	99.48	28.64	7.92
4	98.04	31.8	7.92
5	101.36	28.32	7.92
6	101.36	28.32	7.92
7	100.6	30	7.92
8	99.88	30.92	7.92
9	98.84	33.48	7.92
u_6			
0	93.96	30.52	12.46
1	90.48	29.6	12.46
2	91.32	29.8	12.46
3	91.04	29.56	12.46
4	89.68	26.72	12.46
5	90.64	29	12.46
6	89.84	32.84	12.46
7	86.84	33.44	12.46
8	86.84	33.44	12.46
9	87.28	33.24	12.46

the same set of users can overlap each other. The thresholds, *max_dis* and *min_dur*, are used to define the spatial and temporal proximity requirements between members of a group. In particular, the *min_dur* threshold helps to weed out short-lived or accidental closeness between users.

Consider the user movement database in Table 1. For *min_dur* = 3 and *max_dis* = 10, $[0,3]$ is a valid segment of the set of users, $\{u_1, u_2\}$.

Definition 2. Given a set of users G, thresholds *max_dis* and *min_dur*, we say that G, *max_dis* and *min_dur* form a *group pattern*, denoted by $P = \langle G, max\_dis, min\_dur\rangle$, if G has a valid segment.

The valid segments of a group pattern P are those of its G component. We also call a group pattern with k users a *k-group pattern*.

Definition 3. Given two group patterns, $P = \langle G, max\_dis, min\_dur\rangle$ and $P' = \langle G', max\_dis, min\_dur\rangle$, P' is called a *sub-group pattern* of P if $G' \subseteq G$.

In a movement database, a group pattern may have multiple valid segments. The combined length of these valid segments is called the *weight-count* of the pattern. We therefore measure the significance of the pattern by comparing its weight-count with the overall time duration.

Definition 4. Let P be a group pattern with valid segments $s_1, \ldots, s_n$, the **weight-count** and **weight** of P are defined as

$$\textit{weight-count}(P) = \sum_{i=1}^{n} |s_i| \tag{1}$$

$$\textit{weight}(P) = \frac{\textit{weight-count}(P)}{N} = \frac{\sum_{i=1}^{n} |s_i|}{N} \tag{2}$$

Since weight represents the *proportion* of the time points a group of users stay close together, the larger the weight is, the more significant (or interesting) the group pattern is. Furthermore, if the weight of a group pattern exceeds a threshold *min_wei*, we call it a **valid group pattern**, and the corresponding group of users a **valid group**. For example, suppose *min_wei* = 50%. The group pattern $\langle \{u_2, u_4, u_5\}, 10, 3 \rangle$ is valid, since it has a valid segment $\{[3, 9]\}$ and a weight of $7/10 \geqslant 0.5$.

Definition 5. Given the thresholds *max_dis*, *min_dur*, and *min_wei*, the problem of *finding all the valid group patterns* (*or simply valid groups*) is known as **valid group (pattern) mining**.

2.2. Discussions

There are some similarities between group pattern mining and the classical association rule mining. In the latter problem, the goal is to discover all frequent itemsets, which is defined as a set of items, with support exceeding a minimal support threshold. There are however several key differences that render the direct application of association rule mining methods not feasible in valid group pattern mining.

- There is no explicit concept of *transaction* in a movement database. The movement database consists of multiple time series of locations, one for each user. One can try to organize the locations recorded at one time point into some kind of transactions with each transaction representing a set of locations that are not more than *max_dis* apart at that time point. For example, 12 transactions can be derived based on the locations at time point 3 in Table 1: $\{u_1, u_2\}$, $\{u_1, u_3\}$, $\{u_1, u_4\}$, $\{u_1, u_5\}$, $\{u_2, u_4\}$, $\{u_2, u_5\}$, $\{u_2, u_6\}$, $\{u_4, u_5\}$, $\{u_4, u_6\}$, $\{u_5, u_6\}$, $\{u_1, u_4, u_5\}$, and $\{u_2, u_4, u_5\}$. However, this grouping of user locations at each time point into transactions can lead to extremely large number of transactions, especially for a large population. This transactionizing overhead can be prohibitive if there are many time points and users.
- The *weight* defined for valid group pattern mining is very different from the *support* defined in association rule mining. By transactionizing the movement database, it does not address the weight counting problem. For transactions derived from a single time point, we should not double count the transactions that have the same set of users. In the above example, user pair $\{u_4, u_5\}$ is contained in three derived transactions $\{u_4, u_5\}$, $\{u_1, u_4, u_5\}$ and $\{u_2, u_4, u_5\}$. But the weight-count of $\{u_4, u_5\}$ should be only incremented by 1, instead of 3, since all of the three transactions occur at the same time point 3.

Therefore, it is necessary to design new algorithms for mining valid group patterns.

3. Group pattern mining algorithms

In [27], we proposed two algorithms to mine group patterns, known as the *Apriori-like Group Pattern mining* (*AGP*) algorithm and *Valid Group-Growth* (*VG-Growth*) algorithm. The former explores the Apriori property of valid group patterns and extends the Apriori algorithm [3] to mine valid group patterns. The latter is based on idea similar to the FP-growth algorithm. Both Apriori and FP-growth algorithms are originally designed for association rule mining. In the following, we present the proposed AGP and VG-growth algorithms.

3.1. Apriori-liked group pattern mining (AGP) algorithm

AGP algorithm is built upon the Apriori property that also holds for group patterns.

Property 1 (Apriori property of group patterns). *If a group pattern is valid, then all of its sub-group patterns are valid as well.*

Proof. Given *min_wei*, and a group pattern $P = \langle G, min\_dur, max\_dis \rangle$, if P is a valid group pattern, then $\frac{\sum |s_i|}{N} \geqslant \min\_wei$, where s_i's are valid segments of P. Let P' denote any sub-group pattern of P. Note that, for each valid segment s_i of P, there must exist a valid segment s_i' of P' such that $|s_i| \subseteq |s_i'|$ and so $|s_i'| \geqslant |s_i|$. Hence, we have $\frac{\sum |s_i'|}{N} \geqslant \frac{\sum |s_i|}{N} \geqslant \min\_wei$. That is, the sub-group pattern P' is also a valid group pattern. □

The AGP algorithm is shown in Fig. 1. We use C_k to denote the set of candidate k-groups, and $\mathbb{G}_k$ to denote the set of valid k-groups. The AGP algorithm starts by mining $\mathbb{G}_1$, the set of all distinct users. It then uses $\mathbb{G}_1$ to find $\mathbb{G}_2$, which in turn is used to find $\mathbb{G}_3$. The process repeats until no more valid k-groups can be found. In each iteration, Apriori property is used to generate candidate groups of larger size, and to prune the unpromising candidate groups. However, there are two key differences between AGP and the classical Apriori algorithm:

(1) Instead of examining whether a transaction contains a candidate itemset, the AGP algorithm tests whether users in a candidate group are *close* to one another at a given time point.
(2) Instead of simply incrementing support counts, AGP algorithm accumulates the lengths of all valid segments so as to compute the weight of a candidate group.

For example, suppose we want to mine valid group patterns from D (see Table 1) with *max_dis* = 10, *min_dur* = 3, and *min_wei* = 50%. $\mathbb{G}_1$ is first assigned the set $\{\{u_1\}, \{u_2\}, \{u_3\}, \{u_4\}, \{u_5\}, \{u_6\}\}$. We then generate C_2 by a *join* operation, which is the same as that in Apriori algorithm.

$$C_2 = \{\{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_1, u_5\}, \{u_1, u_6\}, \{u_2, u_3\}, \{u_2, u_4\}, \{u_2, u_5\}, \{u_2, u_6\}, \{u_3, u_4\}, \{u_3, u_5\}, \{u_3, u_6\}, \{u_4, u_5\}, \{u_4, u_6\}, \{u_5, u_6\}\}.$$

Then we scan D to compute the weights for each candidate 2-group and select the valid ones for $\mathbb{G}_2$:

Algorithm: AGP
Input: D, max_dis, min_dur, and min_wei
Output: $\mathbb{G}$ (the set of valid groups)

Method:

```
01  G = ∅;
02  G_1 = the set of all distinct users;
03  for (k = 2; G_{k-1} ≠ ∅; k++)
04      C_k = Generate_Candidate_Groups(G_{k-1});
05      for (t = 0; t < N; t++)
06          for each candidate k-group c_k ∈ C_k
07              if Is_Close(c_k, t, max_dis) then
08                  c_k.cur_seg++;
09              else
10                  if c_k.cur_seg ≥ min_dur then
11                      c_k.weight += c_k.cur_seg;
12                  c_k.cur_seg = 0;
13      G_k = {c_k ∈ C_k | c_k.weight ⩾ min_wei · N};
14      G = G ∪ G_k;
15  return G;

procedure Generate_Candidate_Groups(G_{k-1})
01  for each (k-1)-group g_{k-1} ∈ G_{k-1}   // g_{k-1} = (u_1, ..., u_{k-1})
02      for each (k-1)-group h_{k-1} ∈ G_{k-1}   // h_{k-1} = (v_1, ..., v_{k-1})
03          if (u_1 = v_1) ∧ ··· ∧ (u_{k-2} = v_{k-2}) ∧ (u_{k-1} < v_{k-1}) then
04              c_k = (u_1, ..., u_{k-2}, u_{k-1}, v_{k-1});   // join step
05              if Has_Invalid_Subgroup(c_k, G_{k-1}) then
06                  delete c_k;   // prune step
07              else
08                  c_k.cur_seg = 0;
09                  c_k.weight = 0;
10                  add c_k to C_k;
11  return C_k;

procedure Has_Invalid_Subgroup(c_k, G_{k-1})
01  for each (k-1)-subgroup c_{k-1} of c_k
02      if c_{k-1} ∉ G_{k-1} then
03          return TRUE;
04  return FALSE;

procedure Is_Close(c_k, t, max_dis)
01  for each pair of users (u_i, u_j) in c_k
02      if d(u_i[t].p, u_j[t].p) > max_dis then
03          return FALSE;
04  return TRUE;
```

Fig. 1. Algorithm AGP.

$$\mathbb{G}_2 = \{\{u_1, u_2\}, \{u_1, u_4\}, \{u_1, u_5\}, \{u_2, u_4\}, \{u_2, u_5\}, \{u_2, u_6\}, \{u_4, u_5\}, \{u_4, u_6\}\}.$$

From $\mathbb{G}_2$, we generate C_3:

$$C_3 = \{\{u_1, u_2, u_4\}, \{u_1, u_2, u_5\}, \{u_1, u_4, u_5\}, \{u_2, u_4, u_5\}, \{u_2, u_4, u_6\}, \{u_2, u_5, u_6\}\{u_4, u_5, u_6\}\}.$$

Note that $\{u_2, u_5, u_6\}$ and $\{u_4, u_5, u_6\}$ are subsequently pruned from C_3 since they have an invalid sub-group ($\{u_5, u_6\}$) which is not in $\mathbb{G}_2$. After scanning D again to compute the weights, we obtain $\mathbb{G}_3$:

$$\mathbb{G}_3 = \{\{u_1, u_4, u_5\}, \{u_2, u_4, u_5\}\}.$$

The algorithm terminates here and the discovered valid groups are $\mathbb{G}_2 \cup \mathbb{G}_3$.

Time Complexity Analysis. In the *Generate_Candidate_Groups* procedure, each call to *Has_Invalid_Subgroups* procedure (Line 05) requires $O(k \cdot |\mathbb{G}_{k-1}|)$ time in the worst case. With the two loops in Lines 01 and 02, the time complexity of the *Generate_Candidate_Groups* procedure is $O(k \cdot |\mathbb{G}_{k-1}|^3)$. In the main algorithm, Lines 05–12 scan the database to compute the weight, which costs[2] $O(M \cdot N, N \cdot |C_k| \cdot \binom{k}{2}) = O(M \cdot N, N \cdot |C_k| \cdot k_2)$, where M is the number of distinct users and N is the whole time span of D. Line 13 selects the valid groups, which costs $O(|C_k|)$.

In total, the time cost of AGP algorithm is $O(\Sigma_k\{k \cdot |\mathbb{G}_{k-1}|^3, M \cdot N, N \cdot |C_k| \cdot k^2\})$.

This is a main memory based analysis, which does not consider the disk access overhead. That is, both the movement database and the candidate sets are assumed to reside in main memory. Note that, the $(N \cdot |C_k| \cdot k^2)$ component represents the overheads of scanning D to check the distance between every two users of every candidate group. This is the main bottleneck of all Apriori-like algorithms and we therefore develop the VG-growth algorithm to reduce such bottleneck.

3.2. VG-growth: an algorithm based on valid group graph

AGP algorithm, like the original Apriori algorithm, involves much overhead in candidate k-group generation and multiple database scans. In [11], Han et al. proposed a novel data structure known as *FP-tree* and a divide-and-conquer algorithm, *FP-growth*, that mines association rules without the above overhead. In this section, we will borrow the idea and develop the *Valid Group Graph* data structure and *VG-growth* algorithm.

In a FP-tree (Frequent Pattern tree), each node represents a frequent item, and the frequent items are ordered in support descending order so that the more frequently occurring items are more likely to be shared and thus located closer to the top of the FP-tree. The FP-growth method starts from a frequent item (as an initial *suffix* pattern), examines only its *conditional pattern base* (a "sub-database" which contains the set of frequent items co-occurring with the suffix pattern), constructs its *conditional FP-tree*, and performs mining recursively with such a tree. The major operations of mining are *count accumulation* and *prefix count adjustment*, which are usually much less costly than candidate generation and pattern matching operations performed in the Apriori-like algorithm.

[2] We use $O(A, B, C, \ldots)$ to denote $max\{O(A), O(B), O(C), \ldots\}$.

As we extend the FP-tree structure and FP-growth algorithm to valid group pattern mining, the key differences between association rule mining and valid group pattern mining have to be considered. Moreover, some basic concepts adopted by FP-tree and FP-growth will have to change as described below.

- Each node in a FP-tree is a frequent item, which is also the smallest unit in association rule mining. In valid group pattern mining, the smallest unit is a valid 2-group. A direct construction of FP-tree-like structure based on valid 2-groups will however lead to excessive number of nodes in the tree.
- The *weight* used in valid group pattern mining is more complicated than the *support* measure. Hence, it is necessary to store the list of valid segments for each valid 2-group so as to derive the weight of valid groups of larger sizes.

3.2.1. Valid group (VG) graph

In this section, we define Valid group graph on which our proposed VG-growth algorithm will operate to mine valid group patterns.

Definition 6. A *valid group graph* (*VG-graph*) is a weighted directed graph (V, E, s), where

(1) Each vertex in V represents a user who participates in some valid 2-group, i.e., $V = \{u | u \in G, G \in \mathbb{G}_2\}$. This set of users is also known as *valid users*.
(2) Each weighted directed edge in E represents a valid 2-group and the direction of an edge always origins from the vertex with a smaller user id.
(3) s is a weighting function that maps each edge in E to its valid segments.

Consider D in Table 1. Assume that $max\_dis = 10$, $min\_dur = 3$ and $min\_wei = 50\%$. We construct the VG-graph of D using a modified AGP algorithm that stores valid segments as it computes $\mathbb{G}_2$. $\mathbb{G}_2$ is shown in Table 2 and the constructed VG-graph is shown in Fig. 2. Note that a VG-graph can be constructed for a movement database by first deriving the set of all valid 2-groups, which requires only *one* scan of the movement database.

Table 2
$\mathbb{G}_2$ and the valid segments

Valid 2-groups	Valid segment lists
$\{u_1, u_2\}$	$s(u_1, u_2) = \{[0,3],[7,9]\}$
$\{u_1, u_4\}$	$s(u_1, u_4) = \{[3,9]\}$
$\{u_1, u_5\}$	$s(u_1, u_5) = \{[1,3],[5,9]\}$
$\{u_2, u_4\}$	$s(u_2, u_4) = \{[0,9]\}$
$\{u_2, u_5\}$	$s(u_2, u_5) = \{[3,9]\}$
$\{u_2, u_6\}$	$s(u_2, u_6) = \{[0,3],[5,7]\}$
$\{u_4, u_5\}$	$s(u_4, u_5) = \{[3,9]\}$
$\{u_4, u_6\}$	$s(u_4, u_6) = \{[0,6]\}$

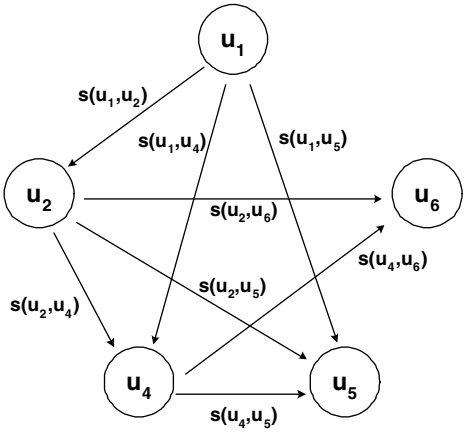

Fig. 2. The VG-graph for Table 1.

Assuming an adjacency list representation, the space required for storing a VG-graph can be determined by:

$$|VG\text{-}graph| = \alpha|V| + \beta|E| + |vsl| \tag{3}$$

where α and β are the space required for storing a vertex and an edge respectively, and $|vsl|$ is the space required for storing the valid segment lists. Note that, $|V| \leqslant M$, $|E| = |\mathbb{G}_2|$ and $|vsl| \leqslant 2\delta|\mathbb{G}_2|\lfloor\frac{N}{\min\_dur+1}\rfloor$, where δ is the space required for storing one time stamp, and the number of valid segments for a valid 2-group is at most $\lfloor\frac{N}{\min\_dur+1}\rfloor$. In addition, we define the *compression ratio* of a VG-graph as:

$$compression\ ratio\ of\ VG\text{-}graph = \frac{|VG\text{-}graph|}{|D|} \tag{4}$$

where $|D|$ denotes the size of the original movement database.

Property 2. Given a movement database D and thresholds *max_dis*, *min_dur* and *min_wei*, its corresponding VG-graph contains the complete information of D relevant to valid group pattern mining.

Proof. In the VG-graph construction process, all the valid 2-groups, associated with their valid segments, are stored in the VG-graph. From Property 1, we know that if a k-group ($k \geqslant 2$) pattern is valid, then all of its 2-subgroup patterns are valid as well. That is to say, each valid k-group can be generated from some valid 2-groups. Moreover, we can check the validity of a k-group by examining the intersections among the valid segments of all its 2-subgroups. Thus the property holds. □

3.2.2. VG-growth algorithm

In this subsection, we present the VG-growth algorithm that uses the compact information in VG-graph for mining the complete set of valid groups.

Definition 7. If $(u \to v)$ is a directed edge in a VG-graph, u is called the *prefix-neighbor* of v.

For example, in Fig. 2, u_1 and u_2 are the prefix-neighbors of u_4.

Definition 8. A *suffix-group*, denoted by H, is an ordered list of k ($k \geqslant 0$) valid users in user id descending order.

In particular, the suffix-group is empty at the beginning of VG-growth algorithm (see Line 05 in Fig. 4).

Definition 9. The conditional VG-graph of a suffix-group H containing k ($k \geqslant 0$) users is called an *k-order conditional VG-graph*, denoted by $VG^{(k)}(H) = (V^k, E^k, s^k)$, and can be constructed as follows.

(1) When $k = 0$, $VG^{(0)}(\varnothing)$ (i.e., the 0-order conditional VG-graph) is the VG-graph constructed from $\mathbb{G}_2$.
(2) When $k \geqslant 1$, let $H = \{u_{a_1}, u_{a_2}, \ldots, u_{a_k}\}$, where $a_1 > a_2 > \cdots > a_k \geqslant 1$. Then, $VG^{(k)}(H) = (V^k, E^k, s^k)$ can be constructed from $VG^{(k-1)}(H - \{u_{a_k}\}) = (V^{k-1}, E^{k-1}, s^{k-1})$ as:

$$\begin{aligned} V^k &= \{u \mid u \in V^{k-1}, (u \to u_{a_k}) \in E^{k-1}\} \\ E^k &= \{(u_i \to u_j) | (u_i \to u_j) \in E^{k-1}, \quad u_i \in V^k, \ u_j \in V^k, \ |s^k(u_i, u_j)| \geqslant \min\_wei \cdot N\} \end{aligned} \tag{5}$$

where

$$s^k(u_i, u_j) = \{s | s \in s_\cap, |s| \geqslant \min\_dur\} \tag{6}$$

and

$$s_\cap = s^{k-1}(u_i, u_j) \cap s^{k-1}(u_i, u_{a_k}) \cap s^{k-1}(u_j, u_{a_k}). \tag{7}$$

The VG-growth algorithm conducts a traversal on the VG-graph, visiting vertices according to their vertex ids. We illustrate the algorithm using the VG-graph in Fig. 2. The vertices are visited as follows:

Vertex u_1: Select the set of prefix-neighbors of vertex u_1, denoted by V_{u_1}. Since V_{u_1} is empty, the mining process for u_1 terminates with no valid group generated.

Vertex u_2: Select the set of prefix-neighbors of vertex u_2, i.e., $V_{u_2} = \{u_1\}$. For each vertex v in V_{u_2}, we generate a valid 2-group by concatenating v with u_2, i.e., $\{u_2, u_1\}$. Select the set of edges on V_{u_2}, denoted by $E(V_{u_2})$. Here, V_{u_2} contains only one vertex, and $E(V_{u_2}) = \emptyset$. The mining process for u_2 terminates.

Vertex u_4: $V_{u_4} = \{u_1, u_2\}$, which generates two valid 2-groups: $\{u_4, u_1\}$ and $\{u_4, u_2\}$. $E(V_{u_4}) = \{(u_1 \to u_2)\}$ with $s(u_1, u_2) = \{[0,3],[7,9]\}$. Adjust the valid segments of edge $\{(u_1 \to u_2)\}$ against u_4: $s(u_1, u_2) = s(u_1, u_2) \cap s(u_1, u_4) \cap s(u_2, u_4) = \{[7,9]\}$. Since the adjusted valid segments do not meet the *min_wei* requirement, the mining process for u_4 terminates.

Vertex u_5: $V_{u_5} = \{u_1, u_2, u_4\}$. Generate three valid 2-groups: $\{u_5, u_1\}$, $\{u_5, u_2\}$, and $\{u_5, u_4\}$. Next, Select the directed edges on V_{u_5} : $E(V_{u_5}) = \{(u_1 \to u_2), (u_1 \to u_4), (u_2 \to u_4)\}$ with associated segment lists: $s(u_1, u_2) = \{[0,3], [7,9]\}$, $s(u_1, u_4) = \{[3,9]\}$, and $s(u_2, u_4) = \{[0,9]\}$. Now, we adjust the associated segment lists against u_5 as follows:

$$s(u_1,u_2) = s(u_1,u_2) \cap s(u_1,u_5) \cap s(u_2,u_5) = \{[0,3],[7,9]\} \cap \{[1,3],[5,9]\} \cap \{[3,9]\} = \{[3],[7,9]\}$$

$$s(u_1,u_4) = s(u_1,u_4) \cap s(u_1,u_5) \cap s(u_4,u_5) = \{[3,9]\} \cap \{[1,3],[5,9]\} \cap \{[3,9]\} = \{[5,9]\}$$

$$s(u_2,u_4) = s(u_2,u_4) \cap s(u_2,u_5) \cap s(u_4,u_5) = \{[0,9]\} \cap \{[3,9]\} \cap \{[3,9]\} = \{[3,9]\}$$

Edges with adjusted segment lists not meeting the *min_dur* and *min_wei* requirements are removed. The edge $(u_1 \rightarrow u_2)$ is removed in this step.

V_{u_5} and $E(V_{u_5})$ (after segment list adjustment) form the conditional VG-graph of u_5(VG(u_5)), which contains three vertices $\{u_1,u_2,u_4\}$ and two edges $(u_1 \rightarrow u_4)$ and $(u_2 \rightarrow u_4)$ with associated segment lists. u_5 is a suffix-group as it will be incorporated as a suffix to every valid group found in $VG(u_5)$.

We perform mining recursively on $VG(u_5)$. We have $V_{u_5u_1} = V_{u_5u_2} = \emptyset$ and $V_{u_5u_4} = \{u_1,u_2\}$. From $V_{u_5u_4}$, two valid 3-groups: $\{u_5,u_4,u_1\}$ and $\{u_5,u_4,u_2\}$ are derived.

Till now, the mining process for u_5 terminates. The mining process for u_5 is shown in Fig. 3.

Vertex u_6: From $V_{u_6} = \{u_2,u_4\}$, we derive two valid 2-groups: $\{u_6,u_2\}$, $\{u_6,u_4\}$. $E(V_{u_6}) = \{(u_2 \rightarrow u_4)\}$ with $s(u_2,u_4) = \{[0,9]\}$. After adjustment, $s(u_2,u_4) = s(u_2,u_4) \cap s(u_2,u_6) \cap s(u_4,u_6) = \{[0,3],[5,6]\}$. The valid segment $[5,6]$ does not meet the *min_dur* requirement and is removed, leaving $[0,3]$ to be the only valid segment. Since $[0,3]$ does not meet the *min_wei* requirement, this edge $(u_2 \rightarrow u_4)$ is removed. The mining process for u_6 therefore terminates.

After visiting all the vertices, VG-growth terminates.

The complete VG-growth algorithm is shown in Fig. 4.

Lemma 1. *Let* $VG^{(k)}(H)$ *be the conditional VG-graph of a suffix-group* $H = \{u_{a_1},\ldots,u_{a_k}\}$ $(k \geqslant 0, a_1 > a_2 > \cdots > a_k)$, *then every edge* $(u_i \rightarrow u_j)$ *in* $VG^{(k)}(H)$ *represents a valid* $k+2$ *group* $\{u_i,u_j,u_{a_1},\ldots,u_{a_k}\}$.

Proof. We prove this lemma by induction.

[*Base Case*]. When $k = 0$, $VG^{(0)}(\emptyset)$ is the original VG-graph. It is clear that every edge $(u_i \rightarrow u_j)$ in the original VG-graph represents a valid 2-group $\{u_i,u_j\}$ and the base case holds.

[*Inductive Hypothesis*]. Suppose the lemma holds for some n, $0 \leqslant n \leqslant k$. That is, every edge $(u_i \rightarrow u_j)$ in $VG^{(n)}(H) = (V^n,E^n,s^n)$ with $H = \{u_{a_1},\ldots,u_{a_n}\}$ represents a valid $n+2$ group $\{u_i,u_j,u_{a_1},\ldots,u_{a_n}\}$.

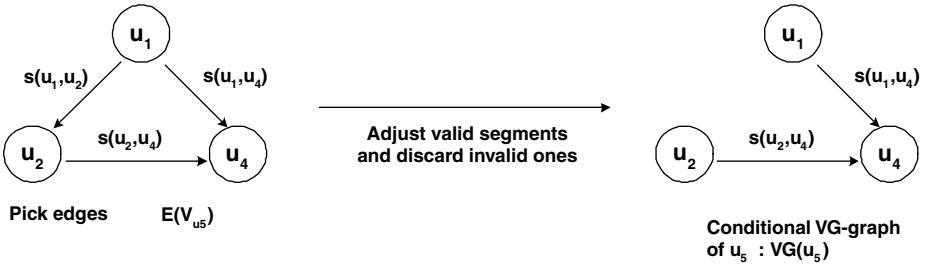

Fig. 3. The mining process for vertex u_5.

Algorithm: VG-growth
Input: D, *max_dis*, *min_dur*, and *min_wei*
Output: all valid groups $\mathbb{G}$

Method:
01 $\mathbb{G} = \varnothing$;
02 $\mathbb{G}_2$ = the set of valid 2-groups generated by a modified AGP algorithm;
03 vsl = the set of valid segment lists associated with $\mathbb{G}_2$;
04 VG-graph = the VG-graph generated from $\mathbb{G}_2$ and vsl;
05 **VG-growth**(VG-graph, null, $\mathbb{G}$);
06 return $\mathbb{G}$;

Procedure: VG-growth(Graph, H, $\mathbb{G}$) // H is the *suffix-group*
01 **for** each vertex u in Graph
02 $H' = H \cup \{u\}$;
03 $V_{H'}$ = the set of prefix-neighbors of u;
04 **if** $V_{H'} \neq \varnothing$ **then**
05 **for** each vertex v in $V_{H'}$
06 obtain a valid group: $G = H' \cup \{v\}$;
07 $\mathbb{G} = \mathbb{G} \cup G$;
08 $E(V_{H'})$ = the set of directed edges on $V_{H'}$;
09 **if** $E(V_{H'}) \neq \varnothing$ **then**
10 **for** each directed edge $(v_i \rightarrow v_j)$ in $E(V_{H'})$
11 $s(v_i, v_j) = s(v_i, v_j) \cap s(v_i, u) \cap s(v_j, u)$;
12 **if** $s(v_i, v_j)$ doesn't satisfy *min_dur*, *min_wei* **then**
13 remove edge $(v_i \rightarrow v_j)$ from $E(V_{H'})$;
14 **if** $E(V_{H'}) \neq \varnothing$ then
15 $VG(H')$ = the conditional valid group graph of H';
16 **VG-growth**$(VG(H'), H', \mathbb{G})$;

Fig. 4. VG-growth algorithm.

[*Inductive Step*]. Consider $VG^{(n+1)}(H \cup \{u_{a_{n+1}}\}) = (V^{n+1}, E^{n+1}, s^{n+1})$. Based on the definition of conditional VG-graph, for each edge $(u_i \rightarrow u_j) \in E^{n+1}$, there must exist $(u_i \rightarrow u_j)$, $(u_i \rightarrow u_{a_{n+1}})$, and $(u_j \rightarrow u_{a_{n+1}})$ in E^n. Given the inductive hypothesis, these three edges represent three valid $n+2$ groups: $\{u_i, u_j, u_{a_1}, \ldots, u_{a_n}\}$, $\{u_i, u_{a_{n+1}}, u_{a_1}, \ldots, u_{a_n}\}$, and $\{u_j, u_{a_{n+1}}, u_{a_1}, \ldots, u_{a_n}\}$, denoted by G^{I}, G^{II}, and G^{III} respectively. In addition, the valid segments of G^{I}, G^{II}, and G^{III} are $s^n(u_i, u_j)$, $s^n(u_i, u_{a_{n+1}})$, and $s^n(u_j, u_{a_{n+1}})$ respectively.

Next, from Eqs. (6) and (7), we know that $s^{n+1}(u_i, u_j)$ satisfies *min_dur* and thus $s^{n+1}(u_i, u_j)$ is the valid segments of group $G^{\mathrm{IV}} = G^{\mathrm{I}} \cup G^{\mathrm{II}} \cup G^{\mathrm{III}}$. From Eq. (5), $s^{n+1}(u_i, u_j)$ also satisfies *min_wei*, thus, G^{IV} is valid. That is, each edge $(u_i \rightarrow u_j)$ in $VG^{(n+1)}(H \cup \{u_{a_{n+1}}\})$ represents a valid $n+3$ group $\{u_i, u_j, u_{a_{n+1}}, u_{a_1}, \ldots, u_{a_n}\}$.

Thus, the lemma holds for $n+1$. □

Theorem 1 (Correctness). *VG-growth only generates valid groups.*

Proof. Without loss of generality,[3] let $G = \{u_{a_1}, \ldots, u_{a_k}\}$ be a valid group generated by VG-growth, where $a_1 > a_2 > \cdots > a_k$. According to the mining process of VG-growth, G is generated when visiting vertex $u_{a_{k-1}}$ in $VG^{(k-2)}(H)$ where $H = \{u_{a_1}, \ldots, u_{a_{k-2}}\}$, and there is an edge $(u_{a_k} \rightarrow u_{a_{k-1}})$ in $VG^{(k-2)}(H)$. Based on Lemma 1, we know that this edge represents a valid k-group $\{u_{a_1}, \ldots, u_{a_k}\}$. Thus, this property is proved. □

Theorem 2 (Completeness). *VG-growth generates all valid groups.*

Proof. Let $G = \{u_{a_1}, \ldots, u_{a_k}\}$ be *any* valid group, where $a_1 > a_2 > \cdots > a_k$ and $k \geqslant 2$. We need to prove that VG-growth will generate G as a valid group. We prove this in two cases: (1) $k = 2$; and (2) $k \geqslant 3$ as follows:

Case 1: when $k = 2$. G will be generated during mining the set of valid 2-groups using AGP (see line 02 in Fig. 4).
Case 2: when $k \geqslant 3$. Given that $G = \{u_{a_1}, \ldots, u_{a_k}\}$ is a valid group, $u_{a_1}, \ldots, u_{a_k}$ are valid users in the original VG-graph. Based on the definition of conditional VG-graph, there must exist a complete subgraph formed by vertices $u_{a_{i+1}}, \ldots$, and u_{a_k} in $VG^{(i)}(H)$ with $H = \{u_{a_1}, \ldots, u_{a_i}\}$, $\forall i, 2 \leqslant i < k$.

Considering the case when $i = k - 2$, there must be an edge $(u_{a_k} \rightarrow u_{a_{k-1}})$ in $VG^{(k-2)}(\{u_{a_1}, \ldots, u_{a_{k-2}}\})$. Therefore, when visiting vertex $u_{a_{k-1}}$, G will be generated as a valid group by VG-growth. Thus, this Theorem is proven. □

3.3. Performance evaluation of AGP and VG-growth

In this section, we evaluate and compare the performance of AGP and VG-growth. The experiments have been conducted using movement databases generated by IBM City Simulator [14] on a Pentium-IV machine with a CPU clock rate of 2.4 GHz and 1 GB of main memory. Note that both AGP and VG-growth were implemented assuming that the movement database resides in main memory. This reduces the required time for our experiments.

City Simulator can generate realistic three-dimensional user movement over city layout that includes streets and buildings. A dataset M1kN1k that contains 1000 users and 1000 time points was generated, covering a 1000 m × 1500 m × 100 m area of 48 roads and 72 buildings with different heights (the highest is around 90 m).

We recorded the total execution time (T), the time for mining valid 2-groups (T_2), and the time for mining all other valid groups (T_k) for different *min_wei* values ranging from 1% to 10%. $T = T_2 + T_k$ as both AGP and VG-growth find valid 2-groups first before the rest. The *max_dis* and *min_dur* thresholds were 30 and 4 respectively. In the experiments, the unit of measurement adopted for distance was meter, and the interval between every two consecutive time points represents 10 min. Thus, $N = 1000$ and *min_dur* = 4 represent about one week and 40 min respectively. Table 3 summarizes the parameters used in this set of experiments.

[3] Although there is no implicit ordering among the users of a group, we can always sort them by user ids.

Table 3
Performance comparison between AGP and VG-growth

Dataset	*M*	*N*	Dataset size (MB)	Thresholds
M1kN1k	1000	1000	12	*max_dis* = 30, *min_dur* = 4 *min_wei* = 1%, 2%, 4%, 6%, 10%

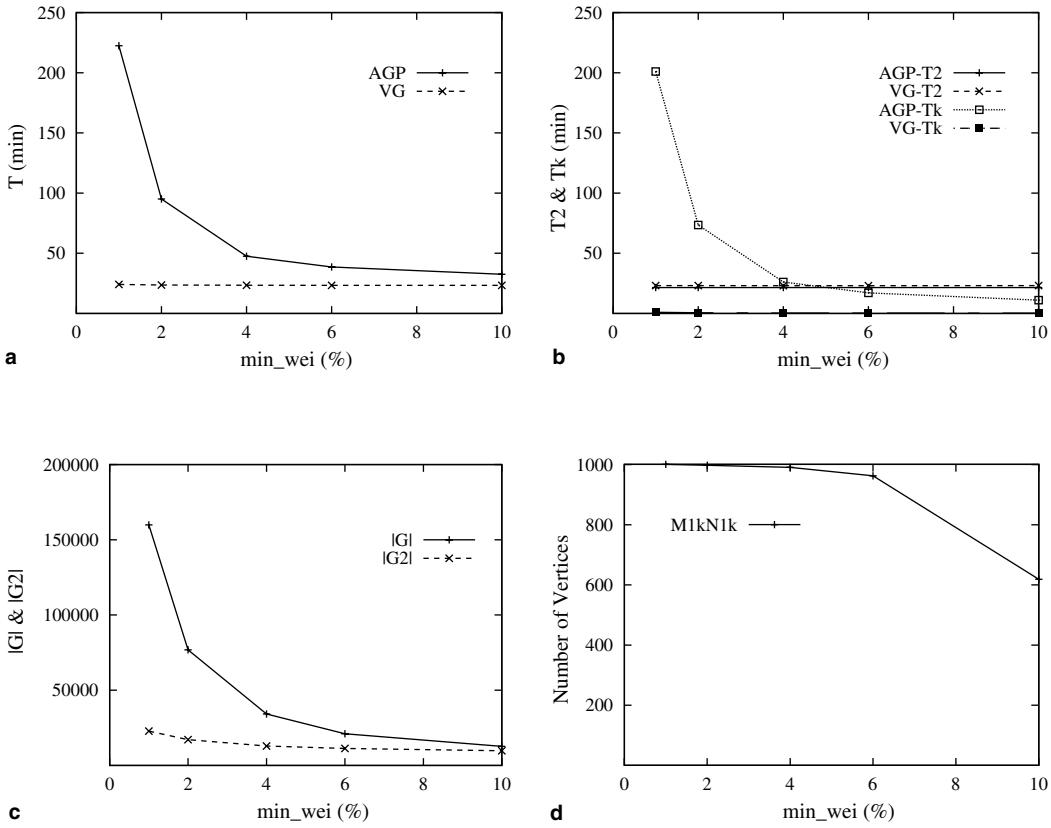

Fig. 5. Experiment results: AGP vs. VG-growth: (a) T (M1kN1k), (b) T_2 and T_k (M1kN1k), (c) $|\mathbb{G}|$ and $|\mathbb{G}_2|$ (M1kN1k), (d) number of vertices in VG-graph.

As shown in Fig. 5(a), VG-growth outperformed AGP in execution time T, especially when *min_wei* is small (<4%). In particular, when *min_wei* = 1%, VG-growth runs 10 times faster than AGP. Fig. 5(b) shows that the execution time differences come from T_k, since AGP and VG-growth share the same procedure of mining valid 2-groups. For small *min_wei*, VG-growth was more efficient than AGP due to larger number of valid groups with size >2 that can be mined using VG-graph. In contrast, AGP suffered from large number of candidate groups, causing large overhead in database scans and validity checking of candidate groups.

As the *min_wei* increased, T_2 began to dominate T due to larger proportions of valid 2-groups, as shown in Fig. 5(c). For example, when *min_wei* = 10%, most valid groups were of size

2 and both AGP and VG-growth spent almost all the time finding valid 2-groups. In other words, the VG-graph has little use to improve the execution time. This also motivates our proposed summarization approach to mine valid 2-groups which will be described in Sections 4 and 5.

In the experiment, we implemented the VG-graph using adjacency list structure. Each vertex in the VG-graph was stored with a list of its prefix-neighbors and the corresponding valid segment lists. Vertex ids and the time stamps were represented as 4 bytes integers and each list pointer required 4 bytes. The byte size of the VG-graph was obtained accordingly. As shown in Fig. 5(d), the number of vertices in the VG-graph was the about same as the number of users when *min_wei* was less than 6%. It decreased with increasing *min_wei*. For example, when *min_wei* = 10%, there were only 618 valid users (i.e., vertices) among the 1000 users in M1kN1k. Fig. 6(a) shows the size of VG-graph in KB for different *min_wei* values. Our experiments have shown that the compression ratio of VG-graph for dataset M1kN1k, which occupies 12 MB space on hard disk, was between 1% and 6%. This indicates very good compression ratios achieved by VG-graph as it only contains the set of valid users and only stores the valid segments of each valid 2-group rather than the actual location records of each user.

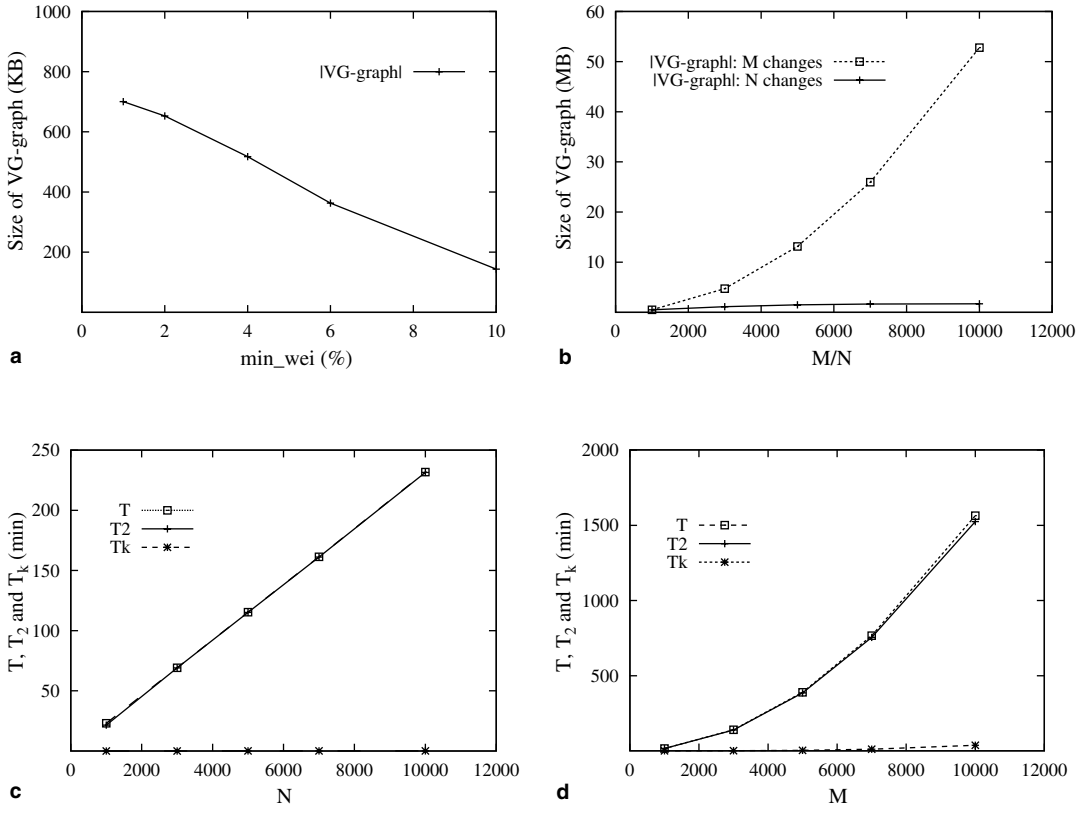

Fig. 6. Experiment results: AGP vs. VG-growth: (a) size of VG-graph, (b) size of VG-graph: M/N changes, (c) scale-up with N (VG-growth), (d) scale-up with M (VG-growth).

Table 4
Scalability of VG-growth

M (thousands)	N (thousands)	Dataset size (MB)	Thresholds
1	1/3/5/7/10	12–120	*max_dis* = 30, *min_dur* = 4
1/3/5/7/10	1	12–120	*min_wei* = 1%, 2%, 4%, 6%, 10%

The scalability results of VG-growth with respect to M and N are shown in Fig. 6(b)–(d). We measured both the execution time and the size of VG-graph for different numbers of users (M) and time points (N) as shown in 4.

We only provide the curves for *min_wei* = 4%, since the curves for other *min_wei*s have the same trends. Fig. 6(b) shows the scalability of VG-graph when M or N changes. The size of VG-graph increased almost quadratically with M but increased very little with N. This is because the increase of M leads to not only more vertices in the VG-graph but also more valid segments. On the other hand, the increase of N only causes more valid segments while the number of vertices increases only a little. From Fig. 6(c) and (d), we find that T increased almost linearly with N and almost quadratically with M. This is due to the dominating T_2 which has $O(N \cdot M^2)$ time complexity.

4. Framework for mining valid 2-groups using location summarization

In this section, we propose to address the overhead of mining valid 2-groups. We first describe a common framework to incorporate different location summarization methods into valid 2-group mining. Subsequently, we present several location summarization methods that adopt different summarization models and assumptions on the summarization parameters.

The proposed framework consists of two steps: *preprocessing* and *mining*. In the preprocessing step, each user's movement data are first divided into time windows of the same size and locations within each time window are summarized using a *summarization model*. After that, the upper bounds of weight-count and valid segment length for each user pair are computed based on the summarized location data.

In the mining step, we first generate a set of candidate 2-groups based on the upper bound information. This set of candidate 2-groups is expected to be smaller than all possible 2-groups. Moreover, instead of scanning the large movement database, the much smaller summarized location database is scanned to check the validity of each candidate 2-group. Only when valid segments could not be determined based on the summarized data, the original database will then be accessed. We elaborate the details in the following subsections.

4.1. Preprocessing of user movement data

Let D'_i denote the summarized data of user u_i, in which the number of time points in the original movement data of u_i, D_i, is reduced to $N' = \lfloor \frac{N}{w} \rfloor$, where w is the time window size and N is the number of time points in D_i. For simplicity, we assume that $\frac{N}{w}$ is a whole number. Note that a time point t' in the summarized database D'_i corresponds to a time window $[t' \cdot w, (t'+1) \cdot w)$ in D_i. We use $u_i[t'].P$ to denote $\{u_i[t].p | t' \cdot w \leqslant t < (t'+1) \cdot w\}$. Based on a *summarization model* (SM), which is some 3D geometry shape such as sphere, cube, etc., $u_i[t'].P$ is summarized to an *instance* of the

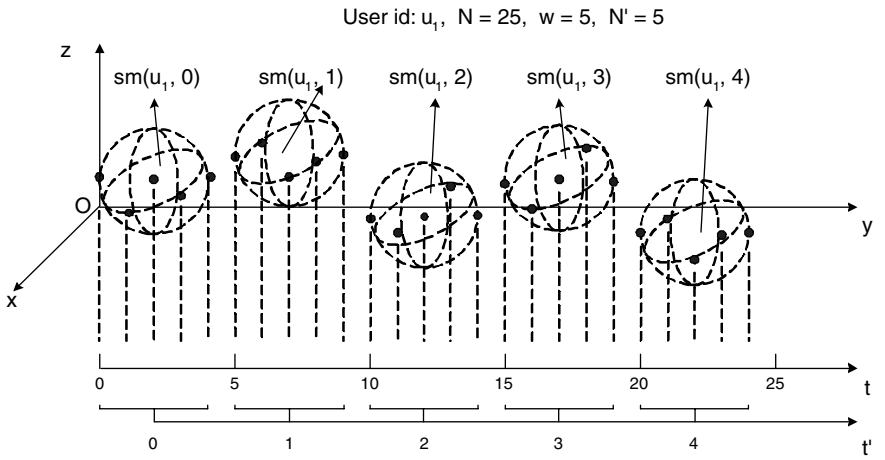

Fig. 7. Example of instances of summarization model.

corresponding SM, denoted by $sm(u_i, t')$. D'_i is therefore $\{sm(u_i,0), sm(u_i,1), \ldots, sm(u_i,N'-1)\}$. In addition, we define the *summarization ratio* of a summarized database as $\frac{|D'|}{|D|}$, where $|D'|$ and $|D|$ are the sizes of D' and D respectively.

For example, Fig. 7 illustrates the instances of summarization model, where N and w are 25 and 5 respectively. Note that, the location points within each time window are summarized into a sphere, i.e., an instance of the sphere summarization model, which will be described in detail later. The summarized database contains five spheres, each represented by a center and a radius.

In this paper, we will examine four different SMs, namely:

- Sphere location summarization method (SLS);
- Cuboid location summarization method (CLS);
- Grid-sphere location summarization method (GSLS);
- Grid-cuboid location summarization method (GCLS).

These methods will be further described in Section 5. Extensions to GSLS and GCLS to consider maximum speed constraint on user movement are given in Appendix A. These extensions however yield performance results similar to GSLS and GCLS. Hence, we do not report their results in the paper.

With D', the number of time points to be scanned are reduced from N to $\frac{N}{w}$. However, this does not address the problem of scanning D' for large number of candidate 2-groups. Thus, in the preprocessing step, we pre-compute the *upper bounds* of weight-count and valid segment length for each user pair based on D'. The pre-computation is carried out under the assumption that the upper bound of *max_dis*, denoted by $\overline{max\_dis}$, is given.

Definition 10. Let t' be a time point in the summarized database D'. Let $\overline{max\_dis}$ be the *upper bound* of *max_dis* (i.e., $\overline{max\_dis} \geqslant max\_dis$). Then $sm(u_i, t')$ and $sm(u_j, t')$ are said to be **possibly close**, if:

$$MinDistance(sm(u_i, t'), sm(u_j, t')) \leqslant \overline{max\_dis} \tag{8}$$

where MinDistance($sm(u_i, t'), sm(u_j, t')$) is a function returning the minimum distance between $sm(u_i, t')$ and $sm(u_j, t')$.

Definition 11. Given a summarized database D', a user pair $\{u_i, u_j\}$, and $\overline{max\_dis}$, a set of consecutive time points $[t'_a, t'_b]$ is called a *possibly close segment* (*PCS*) of $\{u_i, u_j\}$, if:

(1) $\forall t' \in [t'_a, t'_b]$, $sm(u_i, t')$ and $sm(u_j, t')$ are possibly close.
(2) If $t'_a > 0$, $sm(u_i, t'_a - 1)$ and $sm(u_j, t'_a - 1)$ are not possibly close.
(3) If $t'_b < N'$, $sm(u_i, t'_b + 1)$ and $sm(u_j, t'_b + 1)$ are not possibly close.

We use $S(\{u_i, u_j\})$ to denote the set of PCSs of $\{u_i, u_j\}$, i.e.,

$$S(\{u_i, u_j\}) = \{[t'_a, t'_b] \mid [t'_a, t'_b] \subseteq [0, N'), [t'_a, t'_b] \text{ is a PCS of } \{u_i, u_j\}\} \tag{9}$$

Property 3. $\forall s \in s(u_i, u_j)$, $\exists [t'_a, t'_b] \in S(\{u_i, u_j\})$ such that $s \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$.

Proof. Recall that $s(u_i, u_j)$ is the set of valid segments of $\{u_i, u_j\}$. Given any valid segment s of $\{u_i, u_j\}$, $s = [t_p, t_q]$ $(0 \leqslant t_p < t_q < N)$, $[t_p, t_q]$ must lie *within* one time window of size w, or *across* more than one time windows, denoted by $[t'_m, t'_n]$. We have $t_p \geqslant (t'_m \cdot w)$ and $t_q < (t'_n + 1) \cdot w$. Since u_i and u_j are not more than $\overline{max\_dis}$ apart $\forall t \in [t_p, t_q]$, $sm(u_i, t'_k)$ and $sm(u_j, t'_k)$ should be possibly close at t'_k $(m \leqslant k \leqslant n)$, since $\overline{max\_dis} \geqslant max\_dis$. Hence, we have (1) $[t'_m, t'_n]$ itself is a PCS of $\{u_i, u_j\}$, or (2) $[t'_m, t'_n]$ is covered by a PCS (say, $[t'_p, t'_q]$) of $\{u_i, u_j\}$. Let $[t'_a, t'_b]$ be $[t'_m, t'_n]$ (for case 1), or $[t'_p, t'_q]$ (for case 2). In both cases, we have $[t'_a, t'_b] \in S(\{u_i, u_j\})$, and $s = [t_p, t_q] \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$. Therefore, this property holds. □

The above property says $S(\{u_i, u_j\})$ consists of possibly close segments (in D') that cover all the valid segments of $\{u_i, u_j\}$ in D. This property provides the foundation of the correctness and completeness for the summarization based algorithms.

Definition 12. Given a user pair $\{u_i, u_j\}$, the *longest possibly close segment length* of $\{u_i, u_j\}$ is defined as:

$$Q(\{u_i, u_j\}) = w \cdot \max_{[t'_a, t'_b] \in S(\{u_i, u_j\})} (t'_b - t'_a + 1) \tag{10}$$

Property 4. $\forall s \in s(u_i, u_j)$, $Q(\{u_i, u_j\}) \geqslant |s|$.

Proof. Let $s_{\max}$ be the longest valid segment of $\{u_i, u_j\}$. We want to show that $Q(\{u_i, u_j\}) \geqslant |s_{\max}|$. Due to Property 3, there exists a PCS: $[t'_a, t'_b] \in S(\{u_i, u_j\})$ such that $s_{\max} \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$. Since $[t'_a t'_b] \in S(\{u_i, u_j\})$, $Q(\{u_i, u_j\}) \geqslant w \cdot (t'_b - t'_a + 1) \geqslant |s_{\max}|$. Thus, the property is proven. □

This property asserts that the longest possibly close segment length of a user pair is an upper bound of the valid segment length of this pair of users.

Definition 13. The **upper bound weight-count** of $\{u_i, u_j\}$ is defined as:

$$R(\{u_i, u_j\}) = w \cdot \sum_{[t'_a, t'_b] \in S(\{u_i, u_j\})} (t'_b - t'_a + 1) \tag{11}$$

Property 5. $R(\{u_i, u_j\}) \geqslant$ weight-count $(\{u_i, u_j\})$.

Proof. Recall that *weight-count*$(\{u_i, u_j\}) = \sum_{i=1}^{n} |s_i|$, where $s_i \in s(u_i, u_j)$. Let $S(\{u_i, u_j\})$ be the set of PCSs of $\{u_i, u_j\}$. Note that, for any PCS $\in S(\{u_i, u_j\})$, there are two possible cases: (1) this PCS covers one or more valid segment(s); or (2) this PCS does not cover any valid segment, since $\overline{max\_dis} \geqslant max\_dis$. Let $S'(\{u_i, u_j\})$ denote the set of PCSs that covers one or more valid segment(s). Obviously, $S'(\{u_i, u_j\}) \subseteq S(\{u_i, u_j\})$.

Next, from Property 3, we know that, for each valid segment s_i, there exists a PCS covering s_i. Thus,

$$\sum_{i=1}^{n} |s_i| \leqslant w \cdot \sum_{\text{PCS} \in S'(\{u_i, u_j\})} |\text{ PCS }|$$

From the definition of upper bound weight-count, we know:

$$R(\{u_i, u_j\}) = w \cdot \sum_{\text{PCS} \in S(\{u_i, u_j\})} |\text{PCS}| \geqslant w \cdot \sum_{\text{PCS} \in S'(\{u_i, u_j\})} |\text{PCS}|$$

Therefore, $R(\{u_i, u_j\}) \geqslant$ *weight-count*$(\{u_i, u_j\})$. Thus, the property is proven. □

This property asserts that the upper bound weight-count of a user pair is indeed the upper bound on the weight-count for this pair of users.

Let $\mathbb{P}$ denote the set of all user pairs together with their longest possibly close segment length and upper bound weight-count, i.e.,

$$\mathbb{P} = \{(\{u_i, u_j\}, Q(\{u_i, u_j\}), R(\{u_i, u_j\})) | 1 \leqslant i < j \leqslant M\} \tag{12}$$

where M is the number of distinct users.

$\mathbb{P}$ contains the pre-computed upper bounds information about the valid segment length and the weight-count for each user pair. To efficiently find candidate 2-groups that satisfy the *min_dur* requirement, we sort $\mathbb{P}$ by Q value in *descending* order. We also use $(\mathbb{P}_k.c_2, \mathbb{P}_k.Q(c_2), \mathbb{P}_k.R(c_2))$ to denote the kth tuple in $\mathbb{P}$.

The detailed algorithm for the preprocessing step is shown in Fig. 8.

4.2. Mining of valid 2-groups

After the summarized database D' and precomputed upper bound information $\mathbb{P}$ are constructed, the mining step can be carried out to find the set of valid 2-groups, as shown in Fig. 9. User specified *max_dis*, *min_dur* and *min_wei* are input to the mining step.

From $\mathbb{P}$, we first determine a set of candidate 2-groups, C_2, such that for each $c_2 \in C_2$, $Q(c_2) \geqslant min\_dur$ and $R(c_2) \geqslant min\_wei \cdot N$.

Next, we compute the weight-count of each $c_2 \in C_2$ by scanning the summarized database D'. We classify the closeness of two instances of SM at a summarized time point into three cases:

Input: D, w, and $\overline{max\_dis}$;
Output: D' and $\mathbb{P}$.

Method:

```
01  for each user pair c2 = {ui, uj}
02      Q(c2) = R(c2) = sum(c2) = 0;
03  for (t' = 0; t' < N/w; t'++)
04      for each user ui
05          ui[t'].P = {ui[t].p | t'·w ≤ t < (t'+1)·w };
06          sm(ui, t') = SummarizeLocation(ui[t'].P);
07          add sm(ui, t') into D';
08      for each user pair c2 = {ui, uj}
09          if MinDistance(sm(ui, t'), sm(uj, t')) ⩽ max_dis then
10              R(c2) = R(c2) + w;
11              sum(c2)= sum(c2) + w;
12          else
13              if sum(c2) > Q(c2) then
14                  Q(c2) = sum(c2);
15              sum(c2) = 0;
16  sort ℙ by Q(c2) in decreasing order;
17  return D' and ℙ;
```

Fig. 8. Preprocessing step of framework.

- Case 1: all location points within the two instances of SM are no more than *max_dis* apart (see lines 06 and 07 in Fig. 9).
- Case 2: all location points within the two instances of SM are more than *max_dis* apart (see lines 10–13 in Fig. 9).
- Case 3: otherwise, i.e., only some location points inside the two instances of SM are less than *max_dis* (see line 15 in Fig. 9).

Should case 3 arises, the corresponding time window in the original movement database D will be examined to determine the exact weight-count.

Time complexity analysis. In Fig. 9, line 02 generates the set of candidate 2-groups based on *min_dur* and *min_wei*. The time complexity of procedure *GetCandidate2Groups* is $O(k)$, where k is the number of pairs with $Q(c_2) \geqslant min\_dur$. Note that, $|C_2| = k'$ ($k' \leqslant k$), where k' is the number of pairs that satisfies both $Q(c_2) \geqslant min\_dur$ and $R(c_2) \geqslant min\_wei \cdot N$.

Lines 03–15 compute the weight-count for each candidate 2-group. The time cost of lines 03–15 is: $n_1 \cdot T_{\mathrm{Max}} + n_2 \cdot (T_{\mathrm{Max}} + T_{\mathrm{Min}}) + n_3 \cdot (T_{\mathrm{Max}} + T_{\mathrm{Min}} + T_{\mathrm{COD}})$, where n_1, n_2 and n_3 are the number of times when the above three cases are encountered respectively. T_{Max}, T_{Min}, and T_{COD} are the time costs of procedures *MaxDistance*, *MinDistance* and *CheckOriginalDB* respectively. Note that, $n_1 + n_2 + n_3 = N' \cdot |C_2|$ as there are altogether $N' \cdot |C_2|$ iterations and $T_{\mathrm{COD}} = O(w)$. Thus, the time complexity of lines 03–15 is $O(n_1 + n_2 + w \cdot n_3)$.

Lines 16–17 select and output the set of valid 2-groups, which costs $O(|C_2|) = O(k')$.

```
Input:   D, D', max_dis, min_dur, min_wei, ℙ, and w;
Output:  𝔾2, the set of valid 2-groups.

Method:
01  𝔾2 = ∅;
02  C2 = GetCandidate2Groups(ℙ, min_dur, min_wei);
03  for (t' = 0; t' < N/w; t'++)
04      for each candidate 2-group c2 ∈ C2, c2 = {ui, uj}
05          max_ij_dis = MaxDistance(sm(ui, t'), sm(uj, t'));
06          if max_ij_dis ⩽ max_dis then  // ui and uj must be close
07              c2.cur_seg += w;
08          else
09              min_ij_dis = MinDistance(sm(ui, t'), sm(uj, t'));
10              if min_ij_dis > max_dis then  // ui and uj must be far apart
11                  if c2.cur_seg ⩾ min_dur then
12                      c2.weight += c2.cur_seg;
13                  c2.cur_seg = 0;
14              else
15                  CheckOriginalDB(D, t', w, c2);
16  𝔾2 = {c2 ∈ C2 | c2.weight ⩾ min_wei · N};
17  return 𝔾2;

procedure GetCandidate2Groups (ℙ, min_dur, min_wei)
01  for (i = 0; i < |ℙ|; i++)
02      if ℙi.Q(c2) ⩾ min_dur then
03          if ℙi.R(c2) ⩾ min_wei · N then
04              add ℙi.c2 into C2;
05      else
06          break;
07  return C2;

procedure CheckOriginalDB(D, t', w, c2)
01  for (t = t' · w; t < (t' + 1) · w; t++)
02      if d(ui[t].p, uj[t].p) ⩽ max_dis then
03          c2.cur_seg++;
04      else
05          if c2.cur_seg ⩾ min_dur then
06              c2.weight = c2.weight + c2.cur_seg;
07          c2.cur_seg = 0;
```

Fig. 9. Mining step of framework.

Thus, the total time cost is

$$O(k + n_1 + n_2 + w \cdot n_3 + k').$$

In the best case, $n_3 = 0$, the total time cost becomes

$$O(k + N' \cdot k' + k') = O\left(\frac{N}{w} \cdot k'\right).$$

In the worst case, $n_3 = N' \cdot |C_2| = N' \cdot k'$, the total time cost becomes

$$O(k + N' \cdot k' \cdot w + k') = O(N \cdot k').$$

As mentioned before, without the location summarization, the time cost for finding all valid 2-groups is $O\left(N \cdot \binom{M}{2}\right)$. Note that, here $k' \leqslant k \leqslant \binom{M}{2}$, which indicates the location summarization based algorithms always outperform AGP and VG-growth for mining valid 2-groups.

5. Location summarization methods

In the following, we will introduce four location summarization methods. Each method has its own SM and the corresponding *SummarizeLocation*, *MinDistance*, and *MaxDistance* procedures.

5.1. Sphere location summarization (SLS) method

The sphere location summarization method adopts a *sphere* as the SM. Each sphere is represented by (p_c, r), where p_c is the center and r is the radius. Given w location values from a time window, $u_i[t'].P$, we compute the *minimal* and *maximal* x-, y-, z-values, denoted by $u[t'].x_{\min}$, $u[t'].x_{\max}$, $u[t'].y_{\min}$, $u[t'].y_{\max}$, $u[t'].z_{\min}$, and $u[t'].z_{\max}$. The center and radius of the sphere at time t' are determined respectively by:

$$p_c = \left(\frac{u[t'].x_{\min} + u[t'].x_{\max}}{2}, \frac{u[t'].y_{\min} + u[t'].y_{\max}}{2}, \frac{u[t'].z_{\min} + u[t'].z_{\max}}{2}\right) \tag{13}$$

$$r = \max_{p \in u_i[t'].P} d(p, p_c) \tag{14}$$

Given two spheres (p_{c_i}, r_i) and (p_{c_j}, r_j), the minimum and maximum distances between them can be easily computed as follows:

$$Mindistance = d(p_{c_i}, p_{c_j}) - (r_i + r_j) \tag{15}$$

$$Maxdistance = d(p_{c_i}, p_{c_j}) + (r_i + r_j) \tag{16}$$

5.2. Cuboid location summarization (CLS) method

As shown in Fig. 10, instead of using sphere, the cuboid location summarization (CLS) method uses a *cuboid* to represent locations within a time window. Considering the fact that most users travel larger distance in x–y plane than in the z-dimension, CLS may be more compact than SLS.[4] A more precise summarization will result in: (1) fewer calls to *CheckOriginalDB* procedure; (2) fewer candidate 2-groups to be generated.

[4] In cuboid location summarization, the edges of a cuboid are parallel to the axes.

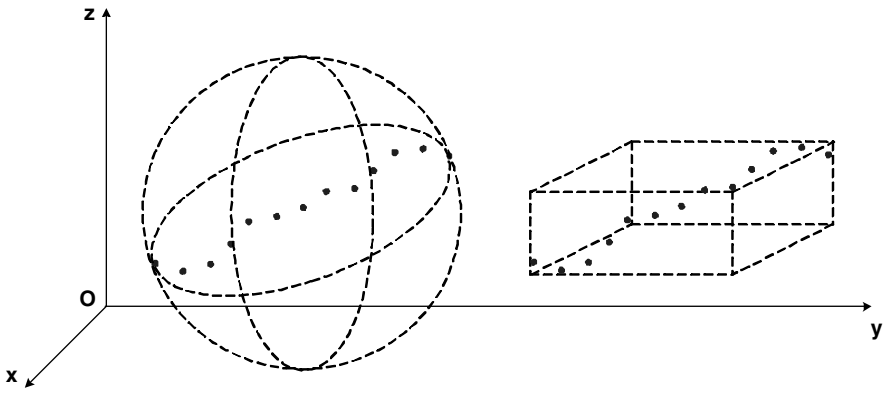

Fig. 10. SLS and CLS models.

The cuboid is defined such that its edges are parallel to the x-/y-/z-axes and is represented by $(v_{\min}, v_{\max})$, where $v_{\min}$ is the corner with minimum x-/y-/z-values, and $v_{\max}$ is that with maximum x-/y-/z-values.

The minimum and maximum distances between two cuboids $(v_{\min_i}, v_{\max_i})$ and $(v_{\min_j}, v_{\max_j})$ are determined as follows:

$$Mindistance = \sqrt{(d_x^{\min})^2 + (d_y^{\min})^2 + (d_z^{\min})^2} \tag{17}$$

where

$$d_x^{\min} = \begin{cases} 0 & \text{if } [v_{\min_i}.x, v_{\max_i}.x] \cap [v_{\min_j}.x, v_{\max_j}.x] \neq \emptyset \\ \max(v_{\min_i}.x, v_{\min_j}.x) - \min(v_{\max_i}.x, v_{\max_j}.x) & \text{otherwise} \end{cases} \tag{18}$$

The $d_y^{\min}$ and $d_z^{\min}$ values are defined similarly.

$$Maxdistance = \sqrt{(d_x^{\max})^2 + (d_y^{\max})^2 + (d_z^{\max})^2} \tag{19}$$

where

$$\begin{aligned} d_x^{\max} &= \max((v_{\max_i}.x - v_{\min_j}.x), (v_{\max_j}.x - v_{\min_i}.x)) \\ d_y^{\max} &= \max((v_{\max_i}.y - v_{\min_j}.y), (v_{\max_j}.y - v_{\min_i}.y)) \\ d_z^{\max} &= \max((v_{\max_i}.z - v_{\min_j}.z), (v_{\max_j}.z - v_{\min_i}.z)) \end{aligned} \tag{20}$$

5.3. Grid-sphere location summarization (GSLS) method

In both SLS and CLS, the actual coordinates and radius are used in location summarization. To further reduce the size of SM, grid based location summarization methods are introduced. One of the four grid based methods to be introduced in this paper is grid-sphere location summarization (GSLS), which is a grid extension to SLS.

Grid based method partitions the space into a set of **cells**, or cubes, of length l, as shown in Fig. 11. Each cell in the grid is given a unique id, which is an integer starting from 0. Given a location (x, y, z), the id of the corresponding cell c can be determined by

$$c.id = c.x' + N_x \cdot c.y' + (N_x \cdot N_y) \cdot c.z' \tag{21}$$

where $c.x' = \lfloor \frac{x}{l} \rfloor$, $c.y' = \lfloor \frac{y}{l} \rfloor$, $c.z' = \lfloor \frac{z}{l} \rfloor$, $N_x = \lceil \frac{XMAX}{l} \rceil$, $N_y = \lceil \frac{YMAX}{l} \rceil$ and $XMAX/YMAX$ are the maximum values in x-/y-dimensions respectively. For example, suppose the space under consideration is $100 \times 100 \times 100$ and the cell length is 10, i.e., $XMAX = YMAX = 100$, $l = 10$. Thus, $N_x = N_y = 10$. Given a 3D point $(12, 35, 70)$, it falls into the cell with $id = \lfloor \frac{12}{10} \rfloor + 10 \cdot \lfloor \frac{35}{10} \rfloor + 10 \cdot 10 \cdot \lfloor \frac{70}{10} \rfloor = 1 + 10 \cdot 3 + 100 \cdot 7 = 731$. That is, instead of storing three integers 12, 35 and 70, we use only cell id 731 to represent the grid cell containing the 3D point $(12, 35, 70)$.

Conversely, given a cell index $c.id$, the corresponding x', y', z' index values can be obtained by a **Reverse** function, i.e., $(c.x', c.y', c.z') = Reverse(c.id, N_x, N_y)$.

$$\begin{aligned} c.z' &= \left\lfloor \frac{c.id}{N_x \cdot N_y} \right\rfloor \\ c.y' &= \left\lfloor \frac{c.id - N_x \cdot N_y \cdot c.z'}{N_x} \right\rfloor \\ c.x' &= c.id - N_x \cdot N_y \cdot c.z' - N_x \cdot c.y' \end{aligned} \tag{22}$$

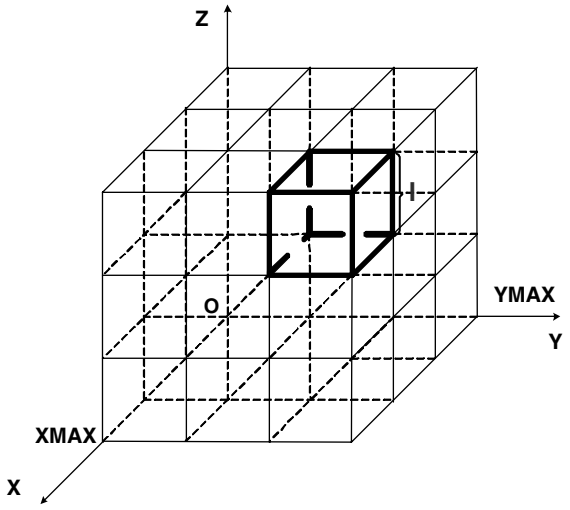

Fig. 11. Partition 3D space into cells.

Using the above example, we apply the Reverse function and get $c.z' = \lfloor \frac{731}{100} \rfloor = 7$, $c.y' = \lfloor \frac{731-100\cdot 7}{10} \rfloor = 3$, and $c.x' = 731 - 100 \cdot 7 - 10 \cdot 3 = 1$. Therefore, we can get the coordinates of the center of cell 731, i.e., $(1 \cdot 10 + 5, 3 \cdot 10 + 5, 7 \cdot 10 + 5) = (15, 35, 75)$. That is, the original 3D point is replaced by the center of the cell containing it. The offset between the original 3D point and the cell center is bounded by $\frac{\sqrt{3}}{2} l$. The smaller the cell length, the smaller the offset is.

The main idea of grid based method is to use a single *c.id* instead of the actual coordinates to represent a location point. This results in large storage saving for D'. Furthermore, the tradeoff between size and accuracy of summarization can be easily tuned by cell length.

In GSLS, the locations within a time window are summarized into a sphere represented by $(p_c.id, \bar{r})$, where $p_c.id$ is the *center cell id* and $\bar{r}$ is the *discretized radius*. The center cell id refers to the cell in which the actual center of the sphere is found. The discretized radius $\bar{r}$ is defined by $\lceil \frac{r}{\gamma} \rceil$, where r is the original radius and γ is the *radius scale unit* for discretizing radius. Such a sphere is also known as the *grid-sphere*.

Given two grid-spheres $(p_{c_i}.id, \overline{r_i})$ and $(p_{c_j}.id, \overline{r_j})$, the minimum and maximum distances between them can be determined by first deriving two larger non-grid spheres (p'_{c_i}, r'_i) and (p'_{c_j}, r'_j) that contain the two original grid-spheres, i.e.,

$$\begin{aligned} p'_{c_i} &= l \cdot \left(p_{c_i}.x' + \frac{1}{2}, p_{c_i}.y' + \frac{1}{2}, p_{c_i}.z' + \frac{1}{2} \right) \\ p'_{c_j} &= l \cdot \left(p_{c_j}.x' + \frac{1}{2}, p_{c_j}.y' + \frac{1}{2}, p_{c_j}.z' + \frac{1}{2} \right) \\ r'_i &= \overline{r_i} \cdot \gamma + \frac{\sqrt{3}}{2} l \\ r'_j &= \overline{r_j} \cdot \gamma + \frac{\sqrt{3}}{2} l \end{aligned} \tag{23}$$

That is, the centers of the two larger non-grid spheres, i.e., p'_{c_i} and p'_{c_j}, are the geometrical centers of the two cells $p_{c_i}.id$ and $p_{c_j}.id$ respectively. The radii r'_i and r'_j are derived from the discretized radii and the radius scale unit. Note that, the derived radii are augmented by $\frac{\sqrt{3}}{2} l$ in order to ensure that the non-grid spheres cover the original grid-spheres, since the maximal offset between the old and new centers is the distance from the new center to the corner of the cell, which is just $\frac{\sqrt{3}}{2} l$. In addition, the x', y', z' index values can be obtained by Reverse function.

Then, the minimum and maximum distances are computed based on (p'_{c_i}, r'_i) and (p'_{c_j}, r'_j), i.e.,

$$\begin{aligned} Mindistance &= d(p'_{c_i}, p'_{c_j}) - (r'_i + r'_j) \\ Maxdistance &= d(p'_{c_i}, p'_{c_j}) + (r'_i + r'_j) \end{aligned} \tag{24}$$

5.4. Grid-cuboid location summarization (GCLS) method

Similar to GSLS, GCLS summarizes the locations within each time window into a cuboid such that the cuboid consists of grid cells and is represented by the ids of the two cells that contains the

Table 5
Summary of location summarization methods

Method	No. of users	No. of time points	SM	Summarized database size (bits)
SLS	M	$\frac{N}{w}$	(p_c, r)	$(3\eta + \lvert r\rvert)M\frac{N}{w}$
CLS			$(v_{\min}, v_{\max})$	$6\eta M\frac{N}{w}$
GSLS			$(p_c.id, \bar{r})$	$(\lvert id\rvert + \lvert\bar{r}\rvert)M\frac{N}{w}$
GCLS			$(id^{\min}, id^{\max})$	$2\lvert id\rvert M\frac{N}{w}$
AGP&VG-growth	M	N	$\langle x, y, z\rangle$	$3\eta MN$

locations with the minimum x-/y-/z-values and maximum x-/y-/z-values. This cuboid is also known as the *grid-cuboid*. Each grid-cuboid is denoted by $\langle id^{\min}, id^{\max}\rangle$. For user u_i at t', the corresponding $id^{\min}$ and $id^{\max}$ are determined as:

$$id^{\min} = \left\lfloor \frac{u_i[t'].x_{\min}}{l} \right\rfloor + N_x \cdot \left\lfloor \frac{u_i[t'].y_{\min}}{l} \right\rfloor + (N_x \cdot N_y) \cdot \left\lfloor \frac{u_i[t'].z_{\min}}{l} \right\rfloor \tag{25}$$

$$id^{\max} = \left\lfloor \frac{u_i[t'].x_{\max}}{l} \right\rfloor + N_x \cdot \left\lfloor \frac{u_i[t'].y_{\max}}{l} \right\rfloor + (N_x \cdot N_y) \cdot \left\lfloor \frac{u_i[t'].z_{\max}}{l} \right\rfloor \tag{26}$$

Similar to GSLS, given two grid-cuboids $\langle id_i^{\min}, id_i^{\max}\rangle$ and $\langle id_j^{\min}, id_j^{\max}\rangle$, the minimum and maximum distances between them can be determined by first deriving two larger non-grid cuboids, $(v'_{i_{\min}}, v'_{i_{\max}})$ and $(v'_{j_{\min}}, v'_{j_{\max}})$, containing the two original grid-cuboids, i.e.,

$$\begin{aligned}
v'_{i_{\min}} &= l \cdot (id_i^{\min}.x', id_i^{\min}.y', id_i^{\min}.z') \\
v'_{i_{\max}} &= l \cdot (id_i^{\max}.x' + 1, id_i^{\max}.y' + 1, id_i^{\max}.z' + 1) \\
v'_{j_{\min}} &= l \cdot (id_j^{\min}.x', id_j^{\min}.y', id_j^{\min}.z') \\
v'_{j_{\max}} &= l \cdot (id_j^{\max}.x' + 1, id_j^{\max}.y' + 1, id_j^{\max}.z' + 1)
\end{aligned} \tag{27}$$

The x', y', z' index values can be obtained by Reverse function. Then, the minimum and maximum distances are computed based on $(v'_{i_{\min}}, v'_{i_{\max}})$ and $(v'_{j_{\min}}, v'_{j_{\max}})$ by using Eqs. (17)–(20).

5.5. Summary of location summarization methods

We summarize the location summarization methods in Table 5, in which the columns from left to right in turn are the location summarization method, the number of users in the summarized database, the number of time points in the summarized database, the summarization model, and the size of the summarized database (in terms of bits). Note that, the last row in Table 5 represents the movement database used by AGP or VG-growth without location summarization. In Table 5, η, $|r|$, $|\bar{r}|$, and $|id|$ denote the number of bits required to represent a coordinate value, a radius, a discretized radius, and a cell id respectively.

6. Performance evaluation of location summarization based algorithms

In this set of experiments, we evaluate the performance of the location summarization based algorithms for mining valid 2-groups using different summarization methods. We first evaluate the impact of summarization parameters on the performance. We also compare the performance of the different algorithms when the summarized database size is fixed. Finally, we compare the performance between AGP and the location summarization based algorithms. Throughout all the experiments, the dataset used is M1kN10k, which contains 1000 users and 10,000 time points and we fixed *max_dis* = 30, *min_dur* = 4, and *min_wei* = 1%.

Recall that, in the experiments described in Section 3.3, the movement database was loaded into main memory and mining did not involve hard disk access. In this set of experiments, however, only the summarized database was loaded into main memory, while the original movement database resided on the hard disk. The intention is to simulate the situation where the movement database is too large to be loaded into memory and to investigate the performance of the location summarization methods under such a disadvantaged condition. It is obvious that if the movement database is also loaded into memory, the performance of the location summarization based algorithms will be even better than that presented here.

6.1. Impact of time window size

In this experiment, we studied the influence of the time window size w on the execution time of the group pattern mining algorithm using different location summarization methods.

Different w valued were chosen: 4, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 100, 200, and 400. The upper bound of *max_dis* threshold (i.e., $\overline{max\_dis}$) was chosen as 40. For SLS and CLS, w is the only summarization parameter. On the other hand, GSLS and GCLS may have multiple combinations of $|id|$ and $|\bar{r}|$ values for a single w value. We therefore ran GSLS and GCLS with different combinations of parameters for each w and reported the ones that gave the smallest T_2. The results are shown in Fig. 12(a), in which we only plot the T_2 curves of SLS and CLS as the curve of GSLS is almost identical to that of SLS. The same applies to GCLS and CLS.

Intuitively, if w is very small, each instance of SM becomes more precise because of fewer location points within a time window. However, a large $N' = \frac{N}{w}$ results in more overhead to scan the summarized database. On the other hand, when w is very large, $N' = \frac{N}{w}$ becomes smaller. But the summarization is relatively coarse and the minimum distance between two instances of SM at any summarized time point is likely to be less than $\overline{max\_dis}$. This causes large number of candidate 2-groups to be generated and more mining overhead. Therefore, there exist some optimal time window size between the two extremes.

As expected, we can see from Fig. 12(a), the performance of SLS and CLS does not scale up linearly with w. In fact, T_2 decreases first when w increases from 4 to around 30 for CLS and 20 for SLS. After that, T_2 increases when w increases further. Each summarization method has an optimal w value. For SLS and GSLS, such an optimal w is around 24. For CLS and GCLS, the optimal w is around 32.

In order to understand the reason behind this observation, we decompose T_2 into $T_{\text{case1,2}}$ and T_{case3} as follows: $T_2 = T_{\text{case1,2}} + T_{\text{case3}}$, where $T_{\text{case1,2}}$ is the time spent on Case 1 and Case 2 (see page 235) and T_{case3} is the time spent on Case 3. We have:

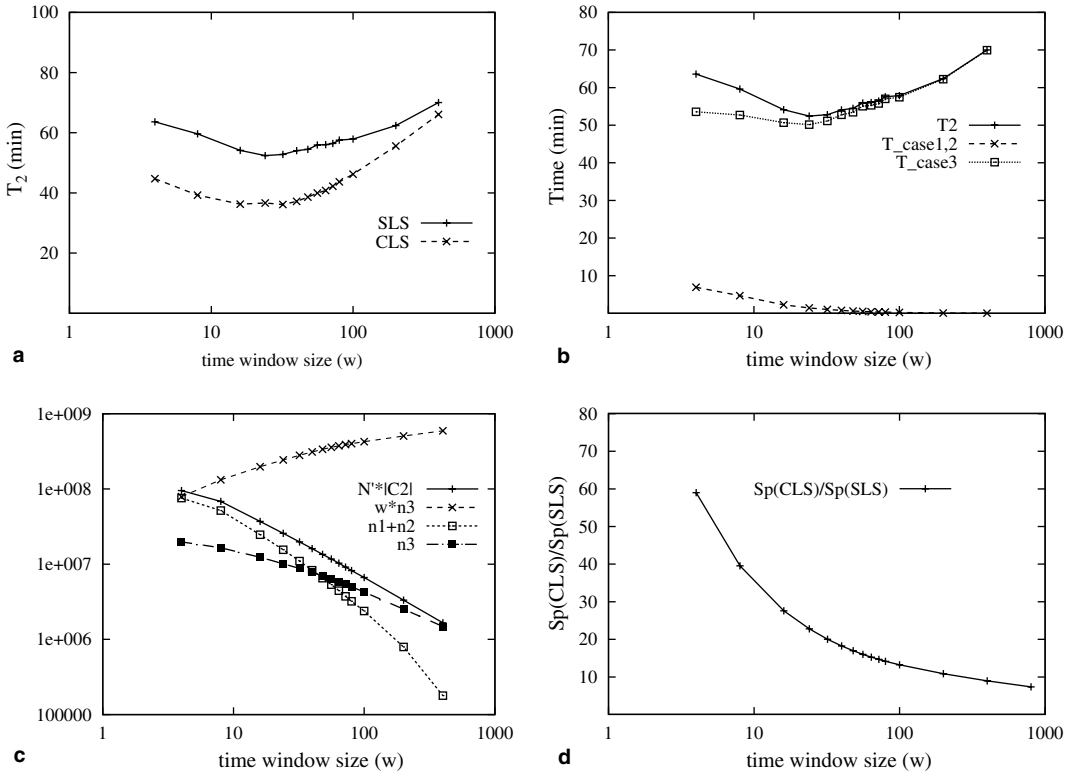

Fig. 12. Impact of time window size.

$$T_{\text{case1,2}} = n_1 \cdot T_{\text{Max}} + n_2 \cdot (T_{\text{Max}} + T_{\text{Min}}) = O(n_1 + n_2)$$
$$T_{\text{case3}} = n_3 \cdot (T_{\text{Max}} + T_{\text{Min}} + T_{\text{COD}}) = O(n_3 + w \cdot n_3).$$

Note that, $T_{\text{case1,2}}$ is solely memory based while T_{case3} consists of two parts. The first part "n_3" represents the memory based cost and the second part "$w \cdot n_3$" represents the hard disk based cost, i.e., the time used for checking the original database which resides on hard disk. Intuitively, the hard disk accessing cost would dominate T_2.

Fig. 12(b) shows $T_{\text{case1,2}}$ and T_{case3} for SLS. The other summarization methods produce similar curves and we therefore chose not to show them. As w increases, $T_{\text{case1,2}}$ becomes negligible, while T_{case3} becomes almost identical to T_2, since T_{case3} dominates T_2. Furthermore, from Fig. 12(c), we can see those components affecting $T_{\text{case1,2}}$ and T_{case3}. Note that, both the x- and y-axes have logarithmic scale. As w increases, the total number of iterations for mining valid 2-groups, $N' \cdot |C_2|$, decreases. Both $n_1 + n_2$ and n_3 also decrease with increasing w. The former results in smaller $T_{\text{case1,2}}$, since $T_{\text{case1,2}} = O(n_1 + n_2)$. However, although n_3 decreases, $w \cdot n_3$ increases with the increasing of w. Recall that they represent memory based cost and hard disk based cost respectively. Hence, it is clear that the memory based cost overrides the hard disk based cost before reaching the optimal w, and reversely beyond the optimal w.

From this experiment, we also found that, with the same w, the two cuboid based methods, i.e., CLS and GCLS, always outperform the two sphere based methods, i.e., SLS and GSLS. This is because, cuboid based methods provide more precise summarizations than sphere based methods. This leads to smaller number of candidate 2-groups, $|C_2|$, and fewer accesses to the original database. We define the *summarization precision*, denoted by S_p, as

$$S_p = \frac{1}{w \cdot \sum_{i=1}^{M} \sum_{t'=1}^{N'} V_{sm(u_i,t')}} \tag{28}$$

where $V_{sm(u_i,t')}$ is the *volume* of the geometrical shape formed by $sm(u_i,t')$.

We compared the summarization precision of different location summarization methods. We found that the summarization precision of SLS is slightly higher than that of GSLS. The same observation also applies to CLS and GCLS. Therefore, we only present the results of SLS and CLS here, as shown in Fig. 12(d). Note that, instead of giving the absolute value, we provide the relative ratio, i.e., $\frac{S_p(CLS)}{S_p(SLS)}$. We can see that the summarization precision of CLS is much higher than that of SLS for any w value. In fact, the S_p of CLS is around 10–60 times of that of SLS. The difference becomes smaller when w increases because the compactness of CLS suffers when w becomes very large.

6.2. Impact of $\overline{max\_dis}$

In this experiment, we studied the effect of $\overline{max\_dis}$ on the mining overhead. For each time window size in the above experiments, $\overline{max\_dis}$ was chosen as 30, 40, 60, 100 and 500. Fig. 13(a) gives the curves for SLS and CLS for different $\overline{max\_dis}$ values, in which we only show the curves for $w = 32$ as the curves for other w values have the same trend. Note that the curve of GSLS is almost identical to that of SLS. The same also applies to GCLS and CLS.

We can see that T_2 increases very little when $\overline{max\_dis}$ increases. This may sounds counterintuitive because we expect a larger $\overline{max\_dis}$ to generate more candidate 2-groups that may significantly increase the mining overhead. Indeed, $|C_2|$ increases dramatically with $\overline{max\_dis}$ increasing, as shown in Fig. 13(b). However, a large $|C_2|$ alone is not sufficient to increase T_2 much, as T_2 is dominated by T_{case3}. Recall that $T_{\text{case3}} = O(n_3 + w \cdot n_3)$. With a fixed w, n_3 is a dominant factor. As shown in Fig. 13(c), n_3 is almost constant when $\overline{max\_dis}$ increases. The underlying reason is, although $|C_2|$ increases quickly causing more checking iterations during mining, most of them fall into Case 2 (see page 235), with only very few iterations falling into Case 3.

In conclusion, within a certain range, e.g., less than 60 in our experiments, $\overline{max\_dis}$ does not affect the execution time much for all the methods. Therefore, we simply fixed $\overline{max\_dis}$ as 40 in the following experiments.

6.3. Performance comparison with fixed summarization database size

Cuboid based algorithms outperforms sphere based algorithms with the same w. But, they generate summarized databases of different sizes. Thus, it is not a fair comparison. We therefore need to compare them based on the same summarized database size.

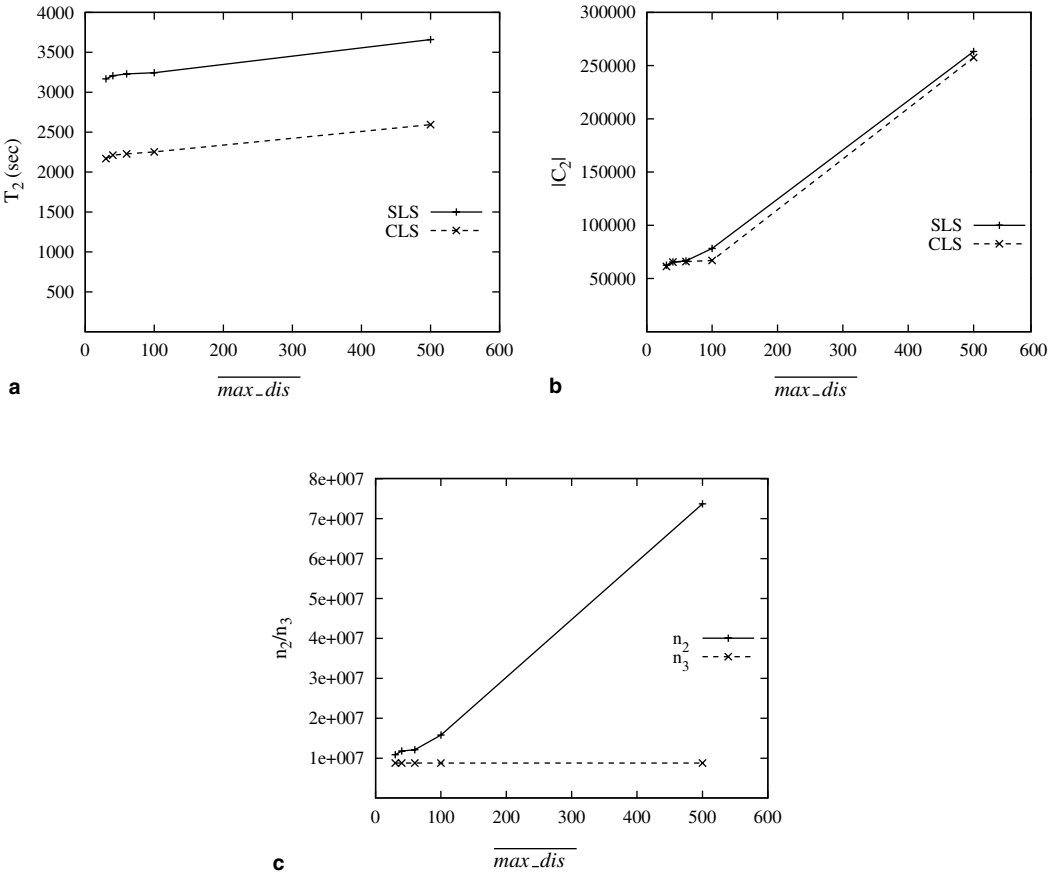

Fig. 13. Impact of $\overline{max\_dis}$. (a) T_2, (b) $|C_2|$ and (c) n_2 and n_3.

In this experiment, we chose seven different $\frac{|D'|}{|D|}$ ratios, ranging from 0.08% to 8.33%. In order to make w a whole number, we chose $\frac{|D'|}{|D|}$ as: $\frac{1}{1200}$, $\frac{1}{120}$, $\frac{1}{60}$, $\frac{1}{48}$, $\frac{1}{36}$, $\frac{1}{24}$, and $\frac{1}{12}$. For each $\frac{|D'|}{|D|}$ ratio, we run each algorithm for several parameter combinations. In particular, for SLS and CLS, $\frac{|D'|}{|D|}$ ratio can be used to determine w. On the other hand, GSLS and GCLS may adopt different combinations of w, $|id|$, and $|\bar{r}|$ values for the same $\frac{|D'|}{|D|}$ ratio. Therefore, for each $\frac{|D'|}{|D|}$ ratio, we run GSLS and GCLS with different combinations of parameters, and reported the ones that give the smallest T_2.

As shown in Fig. 14, the cuboid based methods still enjoy smaller T_2 than the sphere based methods. When the $|D'|$ is relatively large (e.g., $\frac{|D'|}{|D|} > 6\%$), the non-grid based methods perform slightly better than the grid based methods. The reason is, when $\frac{|D'|}{|D|}$ is large, SLS and CLS have more optimal w than GSLS and GCLS. For example, when $\frac{|D'|}{|D|} = \frac{1}{12}$ (8.33%), SLS and CLS have w values as 16 and 24 respectively. On the other hand, GSLS and GCLS share a common w, i.e., 8 (see Table B.2). Note that the optimal w values for sphere and cuboid based methods are around 24 and 32 respectively. For the same reason, when the summarized database is small

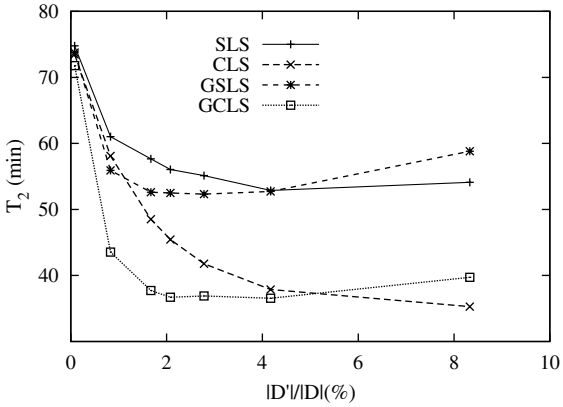

Fig. 14. Performance comparison.

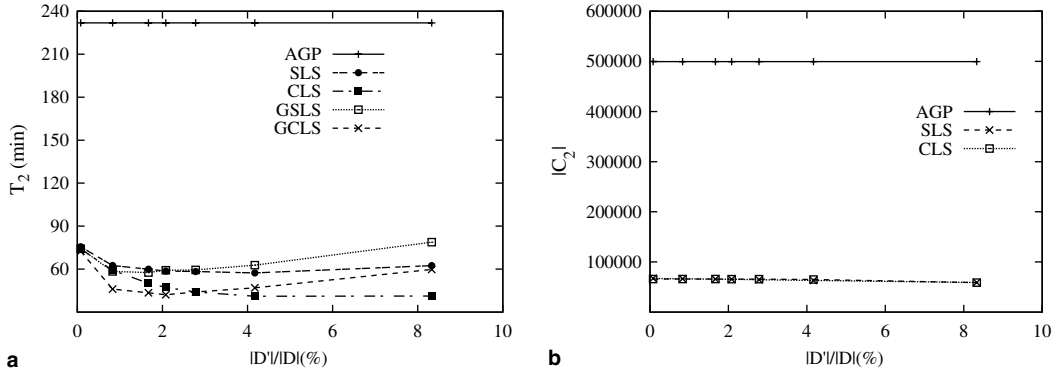

Fig. 15. Performance comparison with AGP: (a) T_2, (b) $|C_2|$.

(e.g., $\frac{|D'|}{|D|} < 4\%$), the grid based algorithms have w values closer to optimal values than the non-grid based algorithms, e.g., when $\frac{|D'|}{|D|} = \frac{1}{120}$ (0.83%) (see Table B.1), leading to smaller T_2.

6.4. Performance comparison between AGP and summarization based mining algorithms

In this subsection, we compare the performance of the algorithms for mining valid 2-groups using location summarization methods with AGP.[5] In particular, we combine the results from Sections 3.3 and 6.3 with the same parameter setting, i.e., dataset M1kN10k, *max_dis* = 30, *min_dur* = 4, and *min_wei* = 1%.

[5] Recall that AGP and VG-growth share the same procedure of finding the set of valid 2-groups.

Fig. 15(a) gives the T_2 curves of the different algorithms. We can see that T_2 values of the location summarization based algorithms are around 10–20% of that of AGP. This illustrates the advantage by using location summarization based algorithms. Recall that AGP runs in main memory, while the location summarization based algorithms need to access the disk-resident original database D. Even so, the latter could still outperform AGP significantly. If the original database was also loaded into main memory, the T_2 of the location summarization based algorithms would be expected to be much smaller.

Fig. 15(b) shows the size of candidate 2-groups generated by AGP and the location summarization based algorithms. Here, we only drew the curves for SLS and CLS, since GSLS and GCLS were very close (slightly below) to those of SLS and CLS respectively. Note that, AGP always generates a constant number of candidate 2-groups, i.e., $\binom{1000}{2} = 499,500$. On the other hand, the location summarization based algorithms generate different sets of candidate 2-groups, which are much smaller than those generated by AGP. In fact, the ratio of $|C_2|/\binom{M}{2}$ for the location summarization based algorithms was around 13%. This significantly reduced the overhead for mining valid 2-groups.

7. Related work

Group pattern mining is a very new data mining research that involves both space and time data. Although there are no existing work dealing with the same problem, there do exist some works that are related.

In mobile computing, in order to reduce the uplink bandwidth consumption during location updating especially for a large population, several group models have been proposed such that location updating is conducted only for each derived group rather than for each individual user [12,13,15,21]. However, the groups defined and discovered in these works are derived only based on physical closeness, without considering time. In addition, there is no interestingness measure for the derived groups, as there is no need to rank the groups in the location update problem.

Group pattern mining deals with time series of user location data. Hence, it is related to the previous works on sequential data mining and time series data mining. Sequential pattern mining was first introduced by Agrawal and Srikant [4] and was generalized by the same authors in [24]. Given a sequence database, in which each sequence is an ordered list of transactions that contains a set of items and has a time stamp, the goal of sequential pattern mining is to find the set of subsequences with support exceeding a minimum support threshold, where the support of a subsequence is the number of sequences containing this subsequence. An Apriori-like algorithm GSP was proposed in [24], which needs to generate and examine a huge number of intermediate candidate subsequences. To break this bottleneck, Han et al. designed an efficient algorithm PrefixSpan for mining large sequence database containing long sequences [18–20]. The PrefixSpan algorithm greatly reduces the efforts of candidate subsequence generation by transforming the sequence database into a set of smaller projected database and performing mining on each projected database. Another efficient sequential pattern mining algorithm, known as SPADE, was proposed by Zaki [31], in which the author adopted vertical format data. Besides, the problem of mining

frequent episodes in event sequences was introduced in [16], in which the episode is defined as a set of events that occur in a given partial order within a time window.

Full periodic patterns was studied in [17], in which the goal is to find all association rules that occur periodically in the whole transaction database. However, partial periodicity is more common in practice since it is likely that only some not all of the time episodes appear periodically, which is the topic studied in [9,10,26]. Furthermore, in [30], the authors considered the situation where the element mutation is allowed and thus proposed a *compatibility matrix* to build probabilistic connections between the observed element and the actual element. In addition, Yang et al. investigated the problem of mining *asynchronous* periodic patterns in time series in [29], considering the case that the presence of periodic patterns may be shifted because of random noise.

There are also some studies on similarity search in time series, such as [1,2]. Among them, Vlachos et al. presented techniques to compute the similarities between trajectories of moving objects in [25]. The trajectory similarity is defined by extending the *longest common subsequence* (LCSS) model [2] to consider the shift in space when comparing two trajectories. Note that, if the spatial shift is not allowed, two moving objects are considered similar if their trajectories are close to each other. Nevertheless, Vlachos et al. focused on designing efficient algorithm to find a trajectory, from a set of trajectories, that is most similar to a given query trajectory. They did not consider the problem of finding *all* the clusters of trajectories that are physically close to one another.

In [23], Shekhar et al. proposed approaches to discover spatial co-location patterns, which is defined as a set of features that are physically close to one another. However, the locations of features are static. This is very different from valid group pattern mining where we deal with users moving continuously.

As discussed in Section 3, if each user is viewed as an item and each group is viewed as an itemset, mining valid groups looks like mining frequent itemsets in association rule mining [3,11]. Unfortunately, we have shown that the existing algorithms in association rule mining cannot be applied directly to valid group mining because of the unique features of the valid group pattern and weight definition.

8. Conclusion

This paper reports a novel approach to mine user group patterns from user movement data. The discovered group patterns, satisfying both spatial and temporal proximity requirements, could potentially be used in target marketing and personalized services. We formally define the valid group pattern mining problem and develop two algorithms (AGP and VG-growth) for mining valid group patterns. The performance of these two algorithms on synthetic movement databases has been reported. It has been shown that VG-growth is a better algorithm and the cost of mining group patterns is mainly due to the mining of valid-2 groups. Since VG-growth and AGP have identical steps for mining valid 2-groups, we further introduce four location summarization methods to reduce the mining overhead. These methods allow valid 2-groups to be mined using smaller number of summarized records and by examining smaller number of candidate 2-groups. We also conducted experiments to evaluate the performance of the four methods. The experiment results have shown that our proposed location summarization methods are much

more efficient than AGP algorithm for mining valid 2-groups. The cuboid based methods also perform better than the sphere based methods.

In our current work, we divide the original database D into different fragments with equal length w. In the future, it is worthwhile to study the possibility of using time windows with different sizes to summarize location information. Besides, we will investigate other possible issues related to valid group pattern mining in our future work, such as efficient approaches to deal with missing data or unsynchronized time stamps in the movement database. Our future research will also consider carrying out data cleaning or data transformation on the movement database before mining. There are also ongoing research extending group pattern mining to dynamic user movement databases.

Appendix A. GSLS and GCLS methods with maximum speed constraint

GSLS2 and GCLS2 are the extensions to GSLS and GCLS that incorporate *the maximum speed constraint on the mobile users*. The maximum distance a user can move within a time window w is denoted by d_w.

For GSLS2, we use d_w to determine a suitable radius scale unit γ. A smaller γ can make the radius of the derived non-grid sphere closer to the original one. Except for the computation of γ, the SM and minimum and maximum distances computations of GSLS2 are the same as GSLS.

Similarly, in GCLS2, by knowing d_w, we can construct a cube with side length equal to d_w. Then, we divide this cube into a set of *sub-cells* with sub-cell length l_{sub}, and give each sub-cell an unique *sub-cell id*. Similar to Eqs. (21) and (22), the computation of sub-cell id and the Reverse function can be carried out based on l_{sub}, $N_{x_{\text{sub}}}$ and $N_{y_{\text{sub}}}$, where $N_{x_{\text{sub}}} = N_{y_{\text{sub}}} = \frac{d_w}{l_{\text{sub}}}$.

As shown in Fig. A.1, the dashed lines represent the sub-cells. Let the cuboid with thickened borders and two corners, $v_{\min}$ and $v_{\max}$, be the summarized cuboid in CLS. Recall that, in GCLS, the two corners are represented by cells containing $v_{\min}$ and $v_{\max}$. In GCLS2, the corner $v_{\min}$ is still represented by the corresponding cell containing it. However, the other corner $v_{\max}$ is represented by the sub-cell containing it, denoted by $sid^{\max}$. That is, in GCLS2, the SM is in

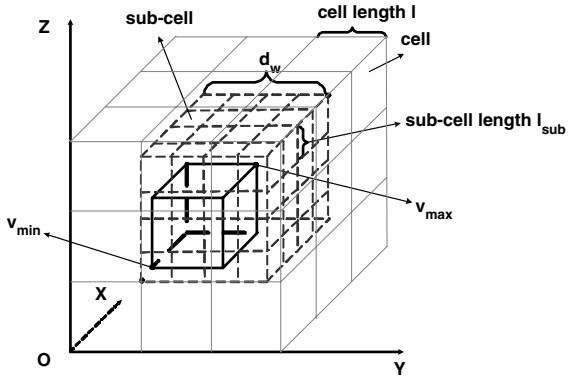

Fig. A.1. Illustration of GCLS2 methods.

the form of $\langle id^{\min}, sid^{\max}\rangle$. For user u_i at t', the corresponding $id^{\min}$ and $sid^{\max}$ can be determined by:

$$id^{\min} = \left\lfloor \frac{u_i[t'].x_{\min}}{l} \right\rfloor + N_x \cdot \left\lfloor \frac{u_i[t'].y_{\min}}{l} \right\rfloor + (N_x \cdot N_y) \cdot \left\lfloor \frac{u_i[t'].z_{\min}}{l} \right\rfloor \tag{A.1}$$

$$\begin{aligned} sid^{\max} = {} & \left\lfloor \frac{u_i[t'].x_{\max} - \left\lfloor \frac{u_i[t'].x_{\min}}{l} \right\rfloor \cdot l}{l_{\text{sub}}} \right\rfloor \\ & + N_{x_{\text{sub}}} \cdot \left\lfloor \frac{u_i[t'].y_{\max} - \left\lfloor \frac{u_i[t'].y_{\min}}{l} \right\rfloor \cdot l}{l_{\text{sub}}} \right\rfloor \\ & + (N_{x_{\text{sub}}} \cdot N_{y_{\text{sub}}}) \cdot \left\lfloor \frac{u_i[t'].z_{\max} - \left\lfloor \frac{u_i[t'].z_{\min}}{l} \right\rfloor \cdot l}{l_{\text{sub}}} \right\rfloor \end{aligned} \tag{A.2}$$

Similar to GCLS, in GCLS2, given two grid-spheres $\langle id_i^{\min}, sid_i^{\max}\rangle$ and $\langle id_j^{\min}, sid_j^{\max}\rangle$, the minimum and maximum distances can be determined by first deriving two larger non-grid cuboids, $(v'_{i_{\min}}, v'_{i_{\max}})$ and $(v'_{j_{\min}}, v'_{j_{\max}})$, containing the two original grid-cuboids, i.e.,

$$\begin{aligned} v'_{i_{\min}} &= (id_i^{\min}.x' \cdot l, id_i^{\min}.y' \cdot l, id_i^{\min}.z' \cdot l) \\ v'_{i_{\max}} &= (sid_i^{\max}.x' \cdot l_{\text{sub}} + l_{\text{sub}} + id_i^{\min}.x' \cdot l, \\ &\quad sid_i^{\max}.y' \cdot l_{\text{sub}} + l_{\text{sub}} + id_i^{\min}.y' \cdot l, \\ &\quad sid_i^{\max}.z' \cdot l_{\text{sub}} + l_{\text{sub}} + id_i^{\min}.z' \cdot l) \\ v'_{j_{\min}} &= (id_j^{\min}.x' \cdot l, id_j^{\min}.y' \cdot l, id_j^{\min}.z' \cdot l) \\ v'_{j_{\max}} &= (sid_j^{\max}.x' \cdot l_{\text{sub}} + l_{\text{sub}} + id_j^{\min}.x' \cdot l, \\ &\quad sid_j^{\max}.y' \cdot l_{\text{sub}} + l_{\text{sub}} + id_j^{\min}.y' \cdot l, \\ &\quad sid_j^{\max}.z' \cdot l_{\text{sub}} + l_{\text{sub}} + id_j^{\min}.z' \cdot l) \end{aligned} \tag{A.3}$$

The x', y', z' index values of $id^{\min}$ can be obtained by Reverse function, while those for $sid^{\max}$ can be obtained by applying Reverse function to the sub-cells, i.e., using $N_{x_{\text{sub}}}$ and $N_{y_{\text{sub}}}$. Then, the extreme distances are computed from $(v'_{i_{\min}}, v'_{i_{\max}})$ and $(v'_{j_{\min}}, v'_{j_{\max}})$ using Eqs. (17)–(20).

Appendix B. Parameter settings for grid based location summarization

We provide the parameter combinations for $\frac{|D'|}{|D|} = \frac{1}{120}$ and $\frac{|D'|}{|D|} = \frac{1}{12}$, as shown in Tables B.1 and B.2 respectively. The parameter combinations in boldface are the ones producing the smallest

Table B.1
Parameter settings for $\frac{|D'|}{|D|} = \frac{1}{120}$ (0.83%)

SLS	$w = 160$					
CLS	$w = 240$					
	$\|id\|$ (bytes)		w			l
GCLS	1		20			84
	2		40			13
	3		60			2
	4		**80**			0.33
	$\|id\|$ (bytes)	$\|\bar{r}\|$ (bytes)	w	d_w	γ	l
GSLS	1	1	20	1806	3.53	84
	2	1	30		3.53	13
		2	40		0.014	
	3	1	40		3.53	2
		2	50		0.014	
		3	60		0.000054	
	4	1	50		3.53	0.33
		2	**60**		0.014	
		3	70		0.000054	
		4	80		0.0000002	
GSLS2	1	1	20	203	0.396	84
	2	1	30	211	0.412	13
		2	40	217	0.0017	
	3	1	40	217	0.423	2
		2	50	219	0.0017	
		3	60	217	0.000007	
	4	1	50	219	0.428	0.33
		2	**60**	217	0.0017	
		3	70	216	0.000007	
		4	80	220	0.00000003	
	$\|id\|$ (bytes)	$\|sid\|$ (bytes)	w	d_w	l_{sub}	l
GCLS2	1	1	20	203	25.25	84
	2	1	30	211	25.91	13
		2	40	217	4.16	
	3	1	40	217	26.4	2
		2	50	219	4.2	
		3	60	217	0.655	
	4	1	50	219	26.56	0.33
		2	60	217	4.16	
		3	**70**	216	0.653	
		4	80	220	0.105	

Table B.2
Parameter settings for $\frac{|D'|}{|D|} = \frac{1}{12}$ (8.33%)

SLS	$w = 16$					
CLS	$w = 24$					
	$\|id\|$ (bytes)		w			l
GCLS	1		2			84
	2		4			13
	3		6			2
	4		**8**			0.33
	$\|id\|$ (bytes)	$\|\bar{r}\|$ (bytes)	w	d_w	γ	l
GSLS	1	1	2	1806	3.53	84
	2	1	3		3.53	13
		2	4		0.014	
	3	1	4		3.53	2
		2	5		0.014	
		3	6		0.000054	
	4	1	5		3.53	0.33
		2	6		0.014	
		3	7		0.000054	
		4	**8**		0.0000002	
GSLS2	1	1	2	40	0.079	84
	2	1	3	56	0.110	13
		2	4	78	0.0006	
	3	1	4	78	0.153	2
		2	5	98	0.0008	
		3	6	117	0.000004	
	4	1	5	98	0.192	0.33
		2	6	117	0.0009	
		3	7	124	0.000004	
		4	**8**	135	0.00000002	
	$\|id\|$ (bytes)	$\|sid\|$ (bytes)	w	d_w	l_{sub}	l
GCLS2	1	1	2	40	6.3	84
	2	1	3	56	8.82	13
		2	4	78	1.94	
	3	1	4	78	12.29	2
		2	5	98	2.44	
		3	6	117	0.434	
	4	1	5	98	15.44	0.33
		2	6	117	2.76	
		3	7	124	0.451	
		4	**8**	135	0.076	

T_2. For example, when $\frac{|D'|}{|D|} = \frac{1}{120}$, GSLS yields the smallest T_2 when $|id| = 4$ bytes, $|\bar{r}| = 2$ bytes, and $w = 60$.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, in: Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms, Chicago, 1993.
[2] R. Agrawal, K.-I. Lin, H. Sawhney, K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, in: Proceedings of VLDB'95, Zurich, Switzerland, 1995.
[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, 1994, pp. 487–499.
[4] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, 1995, pp. 3–14.
[5] B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins, third revised ed.Global Positioning System: Theory and Practice, vol. I, Springer-Verlag Wien, New York, 1994.
[6] G. Chen, D. Kotz, A Survey of Context-Aware Mobile Computing Research, Dartmouth Computer Science Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000.
[7] D. Forsyth, Group Dynamics, third ed., Wadsworth, Belmont, CA, 1999.
[8] G. Giaglis, P. Kourouthanasis, A. Tsamakos, Mobile Commerce: Technology, Theory, and Applications, Idea Group Publishing, 2002, Ch. Towards a Classification Network for Mobile Location Services.
[9] J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, in: Proceedings of the 15th International Conference on Data Engineering, 1999, pp. 106–115.
[10] J. Han, W. Gong, Y. Yin, Mining segment-wise periodic patterns in time-related databases, in: Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, 1998, pp. 214–218.
[11] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of the International Conference on Management of Data, Dallas, TX, 2000.
[12] X. Hong, M. Gerla, Dynamic group discovery and routing in ad hoc networks, in: Proceedings of the First Annual Mediterranean Ad Hoc Networking Workshop (Med-hoc-Net 2002), Sardegna, Italy, 2002.
[13] Y. Huh, C. Kim, Group-based location management scheme in personal communications networks, in: 16th International Conference on Information Networking (ICOIN-16), Korea, 2002.
[14] J. Kaufman, J. Myllymaki, J. Jackson, IBM Almaden Research Center. Available from: <http://www.alphaworks.ibm.com/tech/citysimulator> (December 2001).
[15] G.H.K. Lam, H.V. Leong, S.C.F. Chan, GBL: Group-based location updating in mobile wireless environment, in: 9th International Conference on Database Systems for Advanced Applications (DASFAA 2004), Korea, 2004.
[16] H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery 1 (3) (1997) 259–289.
[17] B. Özden, S. Ramaswamy, A. Silberschatz, Cyclic association rules, in: Proceedings of the 14th International Conference on Data Engineering, Orlando, Florida, USA, 1998, pp. 412–421.
[18] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, PrefixSpan mining sequential patterns efficiently by prefix projected pattern growth, in: Proceedings of the 17th International Conference on Data Engineering (ICDE'Ol), 2001, pp. 215–226. Available from: <citeseer.ist.psu.edu/article/pei01prefixspan.html>.
[19] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: the PrefixSpan approach, IEEE Transactions on Knowledge and Data Engineering 16 (11) (2004) 1424–1440.
[20] J. Pei, J. Han, W. Wang, Mining sequential patterns with constraints in large databases, in: Proceedings of 2002 International Conference on Information and Knowledge Management (CIKM'02), 2002.
[21] G.-C. Roman, Q. Huang, A. Hazemi, Consistent group membership in ad hoc networks, in: International Conference on Software Engineering, 2001, pp. 381–388.
[22] J. Schafer, J. Konstan, J. Riedl, E-commerce recommendation applications, Data Mining and Knowledge Discovery 5 (1/2) (2001) 115–153.
[23] S. Shekhar, Y. Huang, Discovering spatial co-location patterns: a summary of results, Lecture Notes in Computer Science 2121 (2001) 236–256. Available from: <citeseer.ist.psu.edu/article/shekhar01discovering.html>.
[24] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: P.M.G. Apers, M. Bouzeghoub, G. Gardarin (Eds.), Proceedings of the 5th International Conference on Extending

Database Technology, EDBT, vol. 1057, Springer-Verlag, 1996, pp. 3–17. Available from: <citeseer.nj.nec.com/article/srikant96mining.html>.
[25] M. Vlachos, G. Kollios, D. Gunopoulos, Discovering similar multidimensional trajectories, in: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), San Jose, CA, 2002.
[26] W. Wang, J. Yang, P. Yu, Infominer+: mining partial periodic patterns with gap penalties, in: Proceedings of the 2nd IEEE International Conference on Data Mining (ICDM), 2002.
[27] Y. Wang, E.-P. Lim, S.-Y. Hwang, On mining group patterns of mobile users, in: Proceedings of the 14th International Conference on Database and Expert Systems Applications—DEXA 2003, Prague, Czech Republic, 2003.
[28] Y. Wang, E.-P. Lim, S.-Y. Hwang, Efficient group pattern mining using data summarization, in: Proceedings of the 9th International Conference on Database Systems for Advanced Applications—DASFAA 2004, Jeju Island, Korea, 2004.
[29] J. Yang, W. Wang, P. Yu, Mining asynchronous periodic patterns in time series data, IEEE Transaction on Knowledge and Data Engineering (TKDE) 15 (3) (2003) 613–628.
[30] J. Yang, P. Yu, W. Wang, J. Han, Mining long sequential patterns in a noisy environment, in: Proceedings of 2002 ACM-SIGMOD International Conference on Management of Data (SIGMOD'02), 2002.
[31] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences Special Issue on Unsupervised Learning (Doug Fisher, ed.), Machine Learning Journal 42 (2001) 31–60.
[32] P. ZarchanGlobal Positioning System: Theory and Applications, vol. I, American Institute of Aeronautics and Astronautics, 1996.
[33] V. Zeimpekis, G.M. Giaglis, G. Lekakos, A taxonomy of indoor and outdoor positioning techniques for mobile location services, SIGecom Exchanges, ACM vol. 3.4, 2003, pp. 19–27.

.

Yida Wang is a Ph.D. candidate with the School of Computer Engineering at the Nanyang Technological University, Singapore. He obtained his B.Eng. degree from Xian Jiaotong University, China, in 1999. His research interests include data mining, mobile computing, and context aware systems. He has published in international conferences including DEXA, DASFAA, etc.

Ee-Peng Lim is an Associate Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. He obtained his Ph.D. from the University of Minnesota, Minneapolis in 1994. Upon graduation, he started his academic career at the Nanyang Technological University (NTU). In 1997, he established the Centre for Advanced Information Systems and was appointed the Centre Director. He was later appointed a visiting professor at the Chinese University of Hong Kong from December 2001 to June 2003. Upon his return to NTU, he started heading the Division of Information Systems within the School of Computer Engineering. He has published more than 120 referred journal and conference articles in the area of web/data mining, digital libraries and database integration. He is currently an Associate Editor of the ACM Transactions on Information Systems (TOIS) and the International Journal of Digital Libraries (IJDL).

San-Yih Hwang received the B.S. and M.S. degrees from National Taiwan University, Taiwan, and the Ph.D. degree from the University of Minnesota, Minneapolis in 1994, all in computer science. He joined the Department of Information Management at National Sun Yat-sen University, Taiwan, in 1995 and is presently a professor. His current research interests include workflow systems, data mining, and moving object databases.