Anytime Algorithms for Mining Groups with Maximum Coverage

Satya Gautam Vadlamudi **Partha Pratim Chakrabarti** **Sudeshna Sarkar**

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, West Bengal 721302, India
Email: {satya,ppchak,sudeshna}@cse.iitkgp.ernet.in

Abstract

Mining maximal groups from spatio-temporal data of mobile users is a well known problem. However, number of such groups mined can be very large, demanding further processing to come up with a readily usable set of groups. In this paper, we introduce the problem of mining a set of K maximal groups which covers maximum number of users. Such a set of groups can be useful for businesses which plan to distribute a set of K offers targeting groups of users such that a large number of users are covered. We propose efficient methods to solve this hard problem, which do not mine the total set of groups apriori (avoiding the large amount of time consumed upfront), instead intelligently decide during their execution as to which area of the search space is to be explored to mine the next group so that a set of K groups covering large number of users is quickly produced, and then improve the K-set as time progresses (anytime nature). Experimental results on several synthetic spatio-temporal datasets as well as real datasets that are publicly available show the efficacy and scalability of the proposed methods across various parametric inputs.

Keywords: Anytime Algorithms, Coverage, Spatio-Temporal Data Mining, Top K, Valid Groups

1 Introduction

Spatio-temporal mobility data of users consisting of ⟨*time*, *location*⟩ values for each member is of great value and several kinds of interesting information has been mined using such data like groups (Wang et al. 2003), trajectories (Lee et al. 2009), events and social-networks (Lauw et al. 2005), association rules and significant locations (Verhein & Chawla 2006), moving clusters (Kalnis et al. 2005), etc. Specifically, mining group patterns from mobility data can be very useful in target marketing (Schafer et al. 2001), social network analysis (Forsyth 2006), crime investigation (Xu & Chen 2005), habitat monitoring (Davis et al. 2004), and various other applications.

A criterion for a set of users to be designated as a *valid group* is proposed in the work (Wang et al. 2003), based on the proximity in their spatio-temporal mobility (see Section 2 for full details). Later, methods for mining all *maximal* valid groups are proposed in order to avoid redundancy (Wang et al. 2008) which arises because all subsets of a valid group also become valid groups. Although the amount of redundancy is greatly reduced by mining only the maximal valid groups instead of mining all the valid groups, the number of groups mined still remains very large to be used readily (order of *one million* for an instance with $6,000$ users, see (Wang et al. 2008)). Also, mining all maximal valid groups can be extremely time consuming; in our experiments, for an instance with $5,000$ users, VGBK (Wang et al. 2008), an efficient algorithm, could not terminate with the set of all maximal valid groups even after sixty hours.

In this scenario, we propose the problem of finding a set (of cardinality K) of maximal valid groups which covers maximum number of users.

Such a set could be extremely useful in reaching out to a large number of users through a limited set of groups. For example, a business owner or a political outfit can advertise to a large number of people through a limited set of attractive offers. Note that, even if the offer itself does not involve all the members of a group rather only one person per group is chosen, it is still extremely likely that the advertisement reaches/influences maximum number of people by virtue of the groupings. Given the set of all maximal valid groups, the problem is well known as maximum coverage problem which is NP-hard (Hochbaum 1997) (the hardness of the maximum coverage problem holds in our case as well since there exist no generic relations between the maximal valid groups which can trivialize its complexity). Therefore, the search space for finding the set of groups with maximum coverage is exponential in the size of the set of all maximal valid groups.

Such a huge search space poses challenge in all aspects: time, memory, and solution quality. Therefore methods which can work within the given amount of memory and can give a good quality solution within the given amount of time are needed. None of the existing methods can be directly used for this purpose and substantial adaptation is needed if they were to be used. In this work, we explore several algorithms which are adaptations of existing mining techniques, and also develop new methods based on depth-first and best-first search techniques which are of low memory footprint and yet give good anytime performance. *Anytime* (Dean & Boddy 1988), meaning, they readily produce a K-set of groups as soon as they can and improve upon the K-set as time progresses, thereby making a good quality K-set available to use at any time.

Note that, our problem has two aspects to it: 1) mining the maximal valid groups, and 2) choosing the K-set efficiently. For addressing the first aspect, efficient algorithms were proposed in the literature (Wang et al. 2008), and for addressing the second aspect, there are no clear recommendations in the literature, which need to be explored. In this paper, we identify that best-first search algorithms such as the ones based on A* (Hart et al. 1968) which are used for combinatorial optimization can be of help in this case since the mining algorithms proceed in a search algorithm like manner as well. Hence, we explore two types of algorithms, ones which are mining oriented which we adapt for optimization (BKC, DFRC; explained in Section 3), and ones which are optimization oriented which we adapt for mining (ItBC, MAC; explained in Section 3). We observe that each of these algorithms has its strengths and weaknesses and is suited for a particular type of situation. We categorized such cases and identified the corresponding best algorithms in each case, which can be suitably picked as per the needs of the application.

A conceptually closely related and well-studied problem is that of mining frequent itemsets from a given database with a given minimum support and confidence (Agrawal & Srikant 1994). Since the number of such frequent itemsets are usually very large, mining of *closed* frequent itemsets (Pei et al. 2000), *non-derivable* frequent itemsets (Calders & Goethals 2002), and *maximal* frequent itemsets (Gouda & Zaki 2005) were explored. Additional criteria such as support constraints (Wang et al. 2000), item constraints (Srikant et al. 1997), etc. were used to narrow down the number of itemsets. Mining top-k frequent itemsets was proposed to find the most significant itemsets from a database (Han et al. 2002). Redundancy-aware top-k patterns mining (Xin et al. 2006) was proposed to increase the diversity among the top-k patterns while maintaining significance. Compression of itemsets was done by using condensed representations (Pei et al. 2002). Recent work on frequent itemsets called *summarization* focuses on obtaining a representative set of frequent itemsets which can act as a summary of the whole set of frequent itemsets (Afrati et al. 2004). Also, work has been carried out on mining high utility itemsets (Tseng et al. 2010) where cost of different items, their count in a transaction, and the number of transactions they are involved in are taken into account.

Translation of our objective in the context of frequent itemset mining gives rise to mining K maximal frequent itemsets which cover maximum number of items. Perhaps the closest work in spirit to this is of mining redundancy-aware top-k patterns (Xin et al. 2006) where diverse itemsets are covered via the incorporation of pattern redundancy measure. The proposed coverage problem may be very helpful in coming up with a limited number of recommendations/offers that span a wide variety of items. However, in this paper, we restrict ourselves towards mining the user groups with maximum coverage. Also note that, mining group patterns is different from clustering (Ng & Han 1994) and often has relatively more complex criteria.

This paper makes the following key contributions:

1. We introduce the problem of mining K groups such that maximum number of users are covered and explain the utility and importance of such a set.

2. We propose efficient algorithms for solving this problem which are of low memory footprint and which give good anytime performance.

3. We present extensive empirical analysis using synthetic as well as real spatio-temporal datasets which are publicly available. We vary the different mining parameters, the simulator parameters in case of synthetic data, K value, and data size. We analyze the anytime performance (solution quality, time taken) as well as memory consumption. Thereby thoroughly testing the efficacy and scalability of the proposed methods.

The rest of the paper is organized as follows: Section 2 presents the required background on the valid group definition, existing algorithms for mining all maximal valid groups, and the best first search algorithms. In Section 3, we present the proposed methods for mining groups with maximum coverage along with their properties. In Section 4, we present the empirical results demonstrating the efficacy of the proposed methods. We conclude in Section 5.

2 Background

In this section, we briefly present the required background on the definition of valid group, algorithms for mining maximal valid groups, and some best first search approaches that are used later.

Valid group (Wang et al. 2003). The following definitions related to a set of users G lead to the definition of a valid group.

Valid Segment: A valid segment is a set of consecutive timepoints $[t_a, t_b]$ where the set of users G are within *max_dis* distance of each other at each timepoint and the length of the segment is at-least *min_dur* (also the segment has to be maximal, meaning, all users of G are not together at $t_a - 1$ and $t_b + 1$).

Group Pattern: A set of users G, thresholds *max_dis*, *min_dur* form a group pattern if G has a valid segment.

Weight: Weight of a group pattern is the sum of the lengths of all valid segments of that group.

Valid Group: A set of users G is called a valid group if they are part of a group pattern whose weight exceeds a threshold *min_wei*.

Maximal Valid Group: A valid group is called a maximal valid group if it is not a subset of any other valid group.

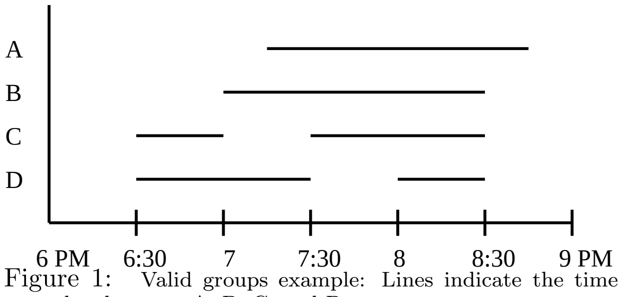

Figure 1: Valid groups example: Lines indicate the time spent by the users A, B, C, and D at a restaurant.

Figure 1 shows the time spent by four users at a restaurant. Assume that the restaurant is identified as a single location (they will be within *max_dis* whenever they are at the restaurant) and the values of *min_dur* and *min_wei* to be 30 Min. and 1 Hour respectively. In such a scenario, the maximal valid groups are: {A,B,C}, {B,C}, {B,D}, and {C,D} (note that, B, C, and D are not all together for *min_wei* time for them to become a valid group).

Mining maximal valid groups (Wang et al. 2008). Two efficient methods, namely, VGMax and VGBK were proposed for mining all the maximal valid groups.

VGMax works as follows: A graph known as VG-graph is formed where the vertices represent the users

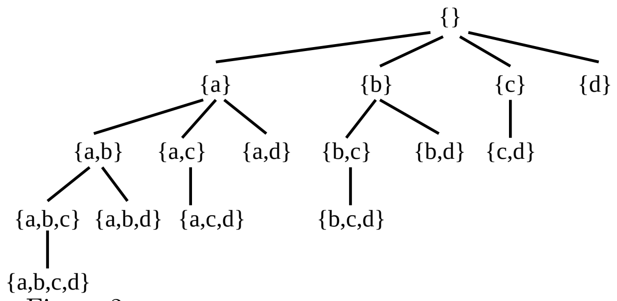

Figure 2: Set enumeration tree consisting of four users.

and each edge represents the set of valid segments between each pair of users. An edge is present between a pair of vertices in VG-graph *iff* the corresponding users form a valid group (i.e., the total length of their valid segments must be at-least *min_wei*). Next, a vertex is extracted from the graph, and a new graph called conditional VG-graph is generated. The conditional VG-graph contains those vertices which form a valid group with the extracted vertex/user. Edges are updated to contain those sets of valid segments which are also shared by the extracted user. This process is repeated in a depth-first manner to generate conditional VG-graphs by extracting vertices one at a time (using the recent conditional VG-graph) until a conditional VG-graph that contains no edges is generated. Whenever a vertex is generated in the conditional VG-graph which does not have any edges incident on it, the extracted set of users and that vertex/user constitute a potential maximal valid group. Since the algorithm traverses the set enumeration tree (Rymon 1992), subgroups of a maximal group will also be recognized as potential maximal valid groups. For example, in Figure 2, if {a,b,d} is found to be a maximal valid group, then later on {b,d} will also be recognized as a potential maximal valid group. Therefore, *MaximalityChecking* is performed which checks whether any superset already exists in the solution set before adding a potential maximal group to it.

VGBK, inspired by the Bron-Kerbosch algorithm (Bron & Kerbosch 1973), also works in similar manner as VGMax but gets away with *MaximalityChecking* by carefully keeping track of the users (say S) that are ignored by VGMax in its traversal but can form a valid group with the current extracted set. Before adding a valid group g to the set of maximal valid groups, it checks whether S is empty or not. If it is not empty, it means that there exists users which can form a valid group with g, implying g to be non-maximal, and vice-versa.

Best-first search/Heuristic search. Given a search graph/tree, it can be mainly explored in three ways: depth-first, breadth-first, or best-first. Best-first exploration involves an estimation/value corresponding to each node denoting its promise towards leading to a goal node. Based on such heuristic estimates, most promising node is selected and expanded until a goal node is found, at which point either the search is terminated or continued to find better/other solutions.

There are several best-first search based anytime algorithms (Zhou & Hansen 2005, Aine et al. 2007, Thayer & Ruml 2010, van den Berg et al. 2011, Vadlamudi et al. 2012) that are potential candidates which can be used here. However, our situation demands that the algorithms should be able to work within the given memory on large sized problems in which case the following two methods appear to be more promising (which we will adapt to solve our problem later in the paper):

Beam search (Bisiani 1987). It is one of the simplest heuristic search methods. Given a beam-width b, it expands b most promising nodes at each level of the search tree in breadth-first manner until a goal-node is found. It uses $2 * b$ nodes of memory for its execution on a search tree.

MAWA* (Vadlamudi et al. 2011). Memory-bounded Anytime Window A* is a memory bounded anytime heuristic search algorithm based on Anytime Window A* (AWA*) (Aine et al. 2007) and Memory-bounded A* (MA*) (Chakrabarti et al. 1989). Since A* (Hart et al. 1968) proceeds in a purely best-first manner, it takes a long time for finding a solution. AWA* adds a depth-component to A* using a window based restriction while exploring in a best-first manner so that good quality solutions can be found quickly. However, like A*, AWA* also runs out of memory while exploring large search spaces. MAWA* was developed by effectively combining AWA* and MA* to give good anytime performance while working within the given amount of memory. It was shown to be effective on diverse search spaces, especially, when using very low memory.

3 Proposed Methods

In this section, we present the proposed algorithms for mining K maximal valid groups that cover maximum number of users, which guarantee maximal coverage. Firstly, we present a generic routine which is used by all the proposed methods for managing the K-set, then we present the methods based on existing mining techniques, followed by the methods based on depth-first search, followed by the methods based on best-first search.

Maximizing coverage. All algorithms can be viewed to be having two parts in-built, one that mines maximal valid groups, and the other which takes the mined maximal valid groups as they are produced and updates the current best solution (the set of K groups) so as to maximize the coverage. Here, we introduce the latter part which takes in a maximal valid group and updates the current solution set.

The MaximizeCoverage routine (see next page) presents the strategy for maximizing coverage. Initially, the solution set is empty, coverage is zero, and the counts of coverage for each user are all zero. The method keeps on adding a new group (if it is not already present) to the solution set until size K is reached. Then onwards, upon getting invoked with a new maximal group, it efficiently replaces a group from the solution set with the new group such that the overall coverage is maximized. Note that the overall coverage c is only increased (by 1) when an individual user coverage count in A raises from 0 to 1. Similarly, c is only decreased (by 1) when an individual user coverage count in A drops from 1 to 0. c remains unaffected at all other times.

3.1 VGBK based Algorithm

Algorithm BKC. Given the above methodology for obtaining the best possible set of K groups[1], any algorithm mining the maximal valid groups can be linked to it to form a mining algorithm that finds K groups with maximum coverage.

Algorithm 1 presents the routine which combines VGBK and MaximizeCoverage methods to mine K groups with maximum coverage. *MineVGgraph* and

[1]In this paper, we use the terms *group*, *maximal group*, *valid group* and *maximal valid group* synonymously unless otherwise being specified in the context.

MaximizeCoverage

```
1: INPUT :: A maximal valid group $g$, current solution set of maximal valid groups $G$, number of groups K, number of users covered by $G$– $c$, and an array $A$ holding the counts of how many times each user is covered in $G$.
2: if $|G| < K$ then
3:     $G \leftarrow G \cup g$; Update the counts in $A$ and coverage $c$ if $G$ is changed;
4:     Return $G, c, A$;
5: end if
6: $old\_c \leftarrow c$;
7: $G \leftarrow G \cup g$; Update the counts in $A$ and coverage $c$;
8: if $c - old\_c = 0$ then
9:     $G \leftarrow G \setminus g$; Update the counts in $A$;
10:    Return $G, c, A$;
11: end if
12: $max\_gain \leftarrow 0$; $max\_g \leftarrow g$;
13: for each $g' \in G$ other than $g$ do
14:     $G \leftarrow G \setminus g'$; Update the counts in $A$ and coverage $c$;
15:     if $c - old\_c > max\_gain$ then
16:         $max\_gain \leftarrow c - old\_c$; $max\_g \leftarrow g'$;
17:     end if
18:     $G \leftarrow G \cup g'$; Update the counts in $A$ and coverage $c$;
19: end for
20: $G \leftarrow G \setminus max\_g$; Update the counts in $A$ and coverage $c$;
21: Return $G, c, A$;
```

Algorithm 1 VGBK based algorithm for Coverage (BKC)

```
1: INPUT :: A spatio-temporal data set $\langle u_i, t_i, l_i \rangle$, $max\_dis$, $min\_dur$, $min\_wei$, and K.
2: $vg \leftarrow$ MineVGgraph(); $G \leftarrow \phi$; $c \leftarrow 0$; $\forall i A(i) \leftarrow 0$; $old\_c \leftarrow -1$;
3: while $c \neq old\_c$ do
4:     $old\_c \leftarrow c$;
5:     Invoke VGBK() with $vg$ and use MaximizeCoverage() to update $G, c, A$ whenever a new maximal valid group $g$ is mined;
6: end while
7: Return $G, c$;
```

VGBK routines are borrowed from (Wang et al. 2008). VG-graph contains information on all valid groups of size 2. VGBK algorithm involves mining VG-graph as its first step, based on which conditional VG-graphs are generated, used and dissolved in future steps. We separate the VG-graph mining in order to avoid re-generating it in each invocation of *VGBK* routine. G denotes the set of K maximal valid groups with coverage c. We choose VGBK against VGMax as it does not require *MaximalityChecking* (Wang et al. 2008) (which needs all the currently mined maximal valid groups to be stored in the memory which can be very large) used in VGMax in order to produce maximal valid groups.

Since VGBK involves a depth-first like traversal of the generic set enumeration tree (Rymon 1992), the above algorithm may take a long time in order to produce a good quality set G (especially when the groups being mined are of large size which results in large unuseful subset space). Usually, one may expect to complete at-least one full traversal of the set enumeration tree in order to attain good coverage. Also, VG-graph mining can take quite a lot of time for data of large number of users, without which the group mining process does not begin, which can hamper the anytime performance (as it causes a large delay at the beginning).

3.2 Depth-First Search based Algorithm

Depth-First search algorithm for mining all Maximal valid groups (DFMax)

```
1: INPUT :: A spatio-temporal data set $\langle u_i, t_i, l_i \rangle$, $max\_dis$, $min\_dur$, $min\_wei$, solution set $G$ (initially empty), and the current set of users which form a valid group $g$ (initially empty).
2: if $g = \phi$ then
3:     for $i = 1$ to $NUM\_USERS$ do
4:         DFMax(group with single member $i$);
5:     end for
6:     Return;
7: end if
8: for $i =$ (largest $uid$ of $g$) $+ 1$ to $NUM\_USERS$ do
9:     if $g \cup i$ is a valid group then
10:        DFMax($g \cup i$);
11:    end if
12: end for
13: if no $i$ could form a valid group with $g$ then
14:     MaximalityChecking($g, G$);
15: end if
```

The DFMax routine presents how all the maximal valid groups could be mined using a depth-first mechanism on the set enumeration tree (Rymon 1992) or a lexicographic tree similar to that of (Agarwal et al. 2000). *MaximalityChecking* routine is borrowed from (Wang et al. 2008). Testing whether the addition of i to g will form a valid group can be done efficiently by storing the valid segments for each group in the data-structure of that group. Then, one can simply perform intersections among the valid segments of the group and the valid segments of 2-groups $\langle i, j \rangle \forall j \in g$ to get the valid segments of $g \cup i$. If the weight of resulting valid segments at any point drops below *min_wei* while performing intersections, then $g \cup i$ could be right away declared as not valid. Note that, *valid segments* of all groups of size 2 are handled via dynamic programming. This is essentially similar to VG-graph except that it is generated on demand and stored for later use, avoiding big lapse in time early. There are no conditional VG-graphs generated which distinguishes DFMax from VGMax.

The algorithm as such is not as efficient as VGMax or VGBK. However, it requires less amount of memory than VG-graph based algorithms because of not having to generate conditional VG-graphs. It is exactly for this reason it involves a large duplication effort in generating *valid segments* of groups of sizes > 2, which ultimately amounts to larger time for mining all maximal valid groups. In spite of this or we can say because of this, it could be very apt for use in mining groups with maximum coverage. Since it does not require to generate conditional VG-graphs, it reports the first maximal group very quickly. We use this property effectively to come up with our next algorithm for coverage.

Algorithm DFRC. Algorithm 2 presents the DFMax based method for solving the coverage problem. The idea is to traverse through those set of vertices/users first in each iteration which are not currently covered, which would ensure quick coverage of all the users. Whenever a maximal group is mined

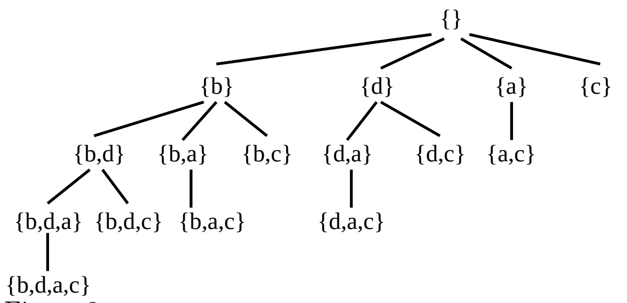

Figure 3: Working of DFRC: Set enumeration tree consisting of four users when they are reordered assuming that a and c are covered.

that increases the overall coverage, the current execution of DFMax is aborted and a new execution is begun which focuses on the new set of uncovered users first. For example, as shown in Figure 3, assuming that a and c are covered currently, set enumeration tree is traversed such that groups involving b and d are mined next. Here, we may need to keep all the maximal valid groups being mined in memory which are necessary for the MaximalityChecking routine. However, since we are only interested in K groups here, we may want to get away with this type of MaximalityChecking operation to save space and time. One way is to follow similar strategy as that of VGBK (which is inspired from the *not* set of Bron-Kerbosch algorithm (Bron & Kerbosch 1973)), but that can become costly to maintain as DFMax does not have conditional VG-graphs for efficiently handling it.

Algorithm 2 DFMax based Re-ordering algorithm for Coverage (DFRC)

```
1: INPUT :: A spatio-temporal data set $\langle u_i, t_i, l_i \rangle$,
   $max\_dis$, $min\_dur$, $min\_wei$, and K.
2: $G \leftarrow \phi$; $c \leftarrow 0$; $\forall i A(i) \leftarrow 0$; $old\_c \leftarrow -1$;
3: while $c \neq old\_c$ do
4:    $old\_c \leftarrow c$;
5:    Re-order users in the increasing order of $A(i)$;
6:    Invoke DFMax() to traverse the set enumer-
      ation tree in the above order and use Maxi-
      mizeCoverage() to update $G, c, A$ whenever
      a new maximal valid group $g$ is mined. Abort
      DFMax when $c$ is increased;
7: end while
8: Return $G, c$;
```

Another way is to perform MaximalityChecking in a different manner, namely, by checking whether any user could be added to the potential group and consider all the users in doing so. Previous knowledge of whether an user was already found to be not compatible with the group (or its parent) during DFMax (line 9) can also be used to speed up MaximalityChecking.

Interestingly, here, one may skip maximality checking methods entirely once the first K-set is obtained and directly use MaximizeCoverage to check for improvement in the coverage of K-set since the first group to improve the coverage will always be maximal (note that, supersets are always explored first in the set enumeration tree). Though this involves invoking MaximizeCoverage with a number of non-maximal groups, this strategy has been proved to be the best in our experiments and hence preferred.

Note that, a BKC like version (traversing entire set enumeration tree) using DFMax would not be effective since it would not have the advantage of conditional VG-graphs for quick traversal. Also, a DFRC like version (restarting after first improvement) using VGBK/VGMax would not be effective since it would have the unnecessary overhead of conditional VG-graphs which will be thrown away after each iteration as they will not be of any use when the traversal order changes.

3.3 Heuristic Search based Algorithms

In the following, we present the five components using which, one can use any heuristic-search method to mine a valid group covering most number of uncovered users (based on the input):

Start state. The start state is same as the root node of the set enumeration tree which we have also used in DFMax. It consists of empty set of users.

Child generation. Child generation also follows closely with the set enumeration tree and DFMax where exactly one more (and each) lexicographically superior user is added to the set of users in the current state, and the child state is generated if that group is valid.

f-value computation. This is the crucial element which guides the search. f-value is obtained by the addition of two values, namely, g (denoting the cost from start node to current node) and h (denoting the estimated cost from current node to a goal node). We obtain g-value by counting the number of uncovered users in the current set. The h-value is set as the number of uncovered users among the lexicographically eligible set that can form a valid group with the set of users in current node. Note that, this number over-estimates the ideal h-value as all those users may not simultaneously form a valid group with the current set of users (we only know that they form a valid group with the current set individually). Overestimating heuristics (in case of a maximization problem) are also called admissible heuristics since they do not undermine the significance of a node. The h-value estimation can be quickened by using the information about users which are known to be not forming valid groups with the parent (and ancestors) of current node (such information is already obtained during h-calculation of its ancestors). Finally, $f = g + h$. (f-value of start state is NUM_USERS). Figure 4 shows the set enumeration tree along with g and h values of different nodes when a and c are already covered via maximal valid group $\{a, c\}$ and assuming that any user can form a valid group with any current set of nodes other than $\{a, c\}$. Note that, only valid children are shown in the figure so that their g and h-values make sense.

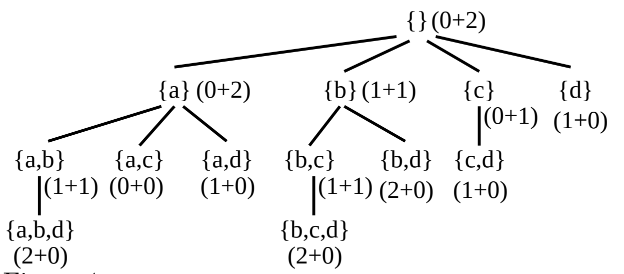

Figure 4: Search tree consisting of four users. f-values corresponding to best-first search are given for each node as $(g + h)$, assuming that a and c are covered via maximal valid group $\{a, c\}$ and any user can form a valid group with any set of users other than $\{a, c\}$.

Goal condition. A node is said to be a goal node if it can not have any valid children (the node itself and its ancestors (except for start node) must all be valid children). Note that, since the traversal is on a set enumeration tree (which has all possible sets as its leaf nodes), a goal node obtained via anytime best-first search need not translate into a maximal valid group at all times. Therefore either maximality checking like mechanisms (like the ones presented in previous subsection) can be used to determine the maximality of a group or further processing may be done to produce a maximal group containing the current set of users of goal node. The former is good for use in BKC or DFRC as a maximal group is always mined ahead of its subgroups, and therefore we may like to ignore the subgroups. Whereas here, the heuristic-search algorithm in a way promises that the goal node is good in terms of coverage (though may not be maximal), and hence we may like to use it rather than throwing away. The technique for making the set of users of goal node into a maximal valid group is similar to that of the maximality checking used in DFRC except that here we keep on adding all eligible users to the set which form a valid group (instead of reporting whether the current group is maximal or not).

Admissible pruning mechanism. This is not an essential part to use heuristic search algorithms but a very important part in order to speedup the search. Admissible pruning refers to the deletion of certain nodes (and therefore their subtrees) which does not effect the results of the search, in other words, the pruning of nodes that are guaranteed to be of no use. Here, we can prune all the nodes whose f-value is zero, as they will not be able to increase the coverage (we can guarantee this since we have used an admissible heuristic for computing the f-value).

Now, we explain how two efficient heuristic-search algorithms, namely, beam search (Bisiani 1987) and MAWA* (Vadlamudi et al. 2011) can be adapted to mine groups with maximum coverage. The choice of these two particular algorithms has been explained in Section 2.

We have managed the *valid segments* of all groups of size 2 via dynamic programming, similar to how we did it in case of DFRC.

As explained in Section 2, beam search expands *beam-width* number of nodes at each level until a goal node is found. For mining a group quickly that increases coverage, we may like to start with beam-width value 1, and increase the beam-width to explore more search space only if needed. The ItB routine presents this technique which stops as soon as coverage is improved.

Iterative Beam search (ItB)

```
1: INPUT :: A spatio-temporal data set $\langle u_i, t_i, l_i\rangle$,
   $max\_dis$, $min\_dur$, $min\_wei$, K, $MAX\_BEAM$,
   solution set $G$, its coverage $c$, and array $A$ with
   counts of number of times each user is covered in
   $G$.
2: $old\_c \leftarrow c$;
3: for $i = 1$ to $MAX\_BEAM$ do
4:    Invoke Beam($i$) with the aforementioned
      heuristic-search components such that the final
      maximized goal node is fed to MaximizeCov-
      erage() to update $G, c, A$;
5:    if $c > old\_c$ then
6:       Return;
7:    end if
8: end for
```

Algorithm ItBC. Given a set of groups G, the above mining algorithm finds a group which increases its coverage. Now, we iteratively invoke the above algorithm to come up with ItBC (similar to how we iteratively used VGBK). Algorithm 3 presents this technique. Since it may take a while to produce a starting solution due to the heuristic calculations involved, it is initialized with DFRC which can give the first K-set quickly and also blend in easily as it too uses dynamic programming strategy for managing *valid segments* of groups of size 2. Initialization with DFRC improved the anytime performance of best-first search based algorithms.

Algorithm 3 ItB for Coverage (ItBC)

```
1: INPUT :: A spatio-temporal data set
   $\langle u_i, t_i, l_i\rangle$, $max\_dis$, $min\_dur$, $min\_wei$, K, and
   $MAX\_BEAM$.
2: $G \leftarrow \phi$; $c \leftarrow 0$; $\forall i A(i) \leftarrow 0$;
3: Run DFRC() until K groups are found;
4: $old\_c \leftarrow 0$;
5: while $c \neq old\_c$ do
6:    $old\_c \leftarrow c$;
7:    ItB($G, c, A, MAX\_BEAM$);
8: end while
9: Return $G, c$;
```

Algorithm MAC. We now present the algorithm when MAWA* is adapted for mining K groups with maximum coverage. MAWA* can be used iteratively to obtain the K-set similar to how Iterative Beam Search has been used. This works fine as long as the first solution produced by MAWA* keeps on improving the K-set. However, when the first solution does not improve the K-set and other solutions need to be searched to improve the K-set, MAWA* can sometimes get into an infinite loop driving towards the same solution. This situation does not arise when it finds more than one solution to give anytime performance in case of traditional optimization problems because, there the solution paths already seen will be cutoff based on f-value as better solutions are being looked for, so the previous solutions will not be found again. However, in our case, we do not obtain any particular cutoff upon discovering the first solution or n^{th} solution to prune previously explored goal nodes.

The solution to this is as follows: one needs to inform the parent of a goal node that the child be no longer considered in future once it is processed. That is, for the rest of the search (of that iteration), that child is always completely ignored. However, upon deleting the parent node itself due to memory limit, this information may get lost and the same goal node may again be generated in future, and the algorithm can still get into an infinite loop. Upon careful observation, it can be seen that this situation is similar to the problem faced when the memory given is limited where same path is explored again and again. To handle this, we use the same remedy used by MA*, the *backup* operation (see (Chakrabarti et al. 1989, Vadlamudi et al. 2011) for details). Therefore, one also needs to perform the *backup* operation after processing the goal node and informing the parent. It involves updating the f-value of parent with maximum of f-values of all its children other than the node(s) to be ignored, and recursively *backing up* all its ancestors. This will ensure that the algorithm though may sometimes regenerate a same goal node, will not do so infinitely often before finding another solution. Detailed proof for this can be obtained on the similar lines of the termination proof of MA* (Chakrabarti et al. 1989). We call MAWA* with the above modification as **Modified-MAWA***. Algorithm 4 shows how this can be iteratively used to construct MAC. Like in ItBC, here too we use DFRC for initialization for good anytime performance.

Algorithm 4 MAWA* based algorithm for Coverage (MAC)

```
1: INPUT :: A spatio-temporal data set $\langle u_i, t_i, l_i \rangle$, $max\_dis$, $min\_dur$, $min\_wei$, K, and $MAX$ (memory limit in terms of number of nodes).
2: $G \leftarrow \phi$; $c \leftarrow 0$; $\forall i A(i) \leftarrow 0$;
3: Run DFRC() until K groups are found;
4: $old\_c \leftarrow 0$;
5: while $c \neq old\_c$ do
6:     $old\_c \leftarrow c$;
7:     Invoke Modified-MAWA*($G, c, A, MAX$) with the aforementioned heuristic-search components such that the final maximized goal node is fed to MaximizeCoverage() to update $G, c, A$. Abort Modified-MAWA* when $c$ is increased.
8: end while
9: Return $G, c$;
```

3.4 Properties

No maximal valid group exists which can replace a valid group from the K-sets produced by BKC, DFRC, and MAC at termination.

This is easy to observe as all these three algorithms (being complete in nature) sweep the entire space of maximal valid groups in their last iteration to find any maximal valid group which can replace a group from their current K-sets improving the coverage. However, note that, these K-sets may be different, because replacing one group at a time from the K-set can only mean that the K-set produced is a *local maxima* (hence the algorithms guarantee to produce *maximal* solutions only), which can be many. ItBC on the other hand does not hold this property as it is not complete for any fixed beam-width.

4 Experimental Results

We now present the experimental results obtained on various spatio-temporal datasets. These include the ones obtained from three data generators (also called as mobility simulators or simply simulators): Random Waypoint model (Broch et al. 1998) based simulator, Oporto simulator (Saglio & Moreira 2001), and Network based data generator (Brinkhoff 2002). It may be noted that, IBM city simulator used in (Wang et al. 2008) is no longer available online. Real datasets are scarcely available due to privacy concerns. We present experiments with one real dataset– the popular Geolife dataset collected and provided by Microsoft (MicrosoftResearchAsia 2012) for research purposes. All the algorithms are implemented in C++. All the experiments have been performed on a machine with Intel Core2 Duo CPU at 2.93-GHz and 2.92-GB RAM.

4.1 With Random Waypoint Simulator Data

For generating synthetic data, we developed a simple yet effective simulator inspired from random waypoint model (Broch et al. 1998), whose details are as follows:

The place of activity is square shaped integer grid whose dimensions are calculated as per the population density and number of users. For example, area of the square grid for 1000 users at a density of 25,000 users/sq.km. is 40,000 sq.m. which results in length of each side of the square to be 200 metres (units). Starting location for each user is chosen randomly on the integer grid. For each timepoint, each user stays at his/her previous location with a probability of 0.5. When the user moves, he/she moves a single step (unit) along x/y-axis in one of the four directions with equal probability (note that, they may slightly get off the grid while moving, no restriction is put).

Factors affecting performance. We observed that the performance of the algorithms is mainly affected by three things: group size distribution of all the maximal valid groups (especially the presence of large sized groups), number of groups K to be mined, and the size of the data (number of users, and number of timepoints). We show representative results on each of the related aspects (results with all possible combinations of parameter values are skipped due to space constraint).

There are several parameters which can be varied: number of users, number of timepoints, population density, max_dis, min_dur, min_wei, and K. Group size is mainly affected by population density, max_dis, min_dur, and min_wei. However, we show results with varying max_dis only (affect of variation in other parameters leading to different group sizes is observed to be similar). We also present results with variation in K and with different sizes of data varying both number of users and timepoints.

Performance measures. The performance of the algorithms can be measured on three fronts: solution quality, time, and memory. The first two are covered by studying the plots showing anytime performance and we provide details on the memory used separately.

Upper bound and Estimate for Coverage. An upper bound for the coverage obtained by K groups can be the sum of sizes of K largest groups (amongst all maximal valid groups). A rough estimate for the coverage can be K * Average group size. These values can only be known for small datasets where one can mine all groups and can then test the quality of coverage attained. In case of large datasets, the estimate can be K * Expected average group size.

Environment settings. Density is set to 25,000/sq.km. (which is a typical value of a city in today's world[2]), no. of timepoints: 1000, min_dur : 3 timepoints (30Min.; assuming 10Min. spacing), and min_wei : 1% (of total number of timepoints; equals 10 timepoints) in all the cases. We use a simpler closeness metric than Euclidean distance to quicken the algorithms, namely, two users are declared close if both x-axis and y-axis distances between them are not more than max_dis. The ItBC algorithm is run with 200 beam-width (corresponds to 200×2 nodes of memory as discussed in Section 2), and the MAC algorithm is run with 100 node memory (sufficient as its expected to be greater than maximum depth (group size)) in all our experiments.

Results on a Small dataset. No. of users: 100. For this set, the total number of valid groups mined are (using VGBK): 2828 with an average group size of 6.026. The distribution of the groups is as follows: 2: 13, 3: 179, 4: 462, 5: 628, 6: 549, 7: 343, 8: 290, 9: 209, 10: 74, 11: 49, 12: 18, 13: 11, 14: 3, where $x : y$ denotes that there are y groups of size x. The time taken is just 2 seconds.

Now, we find the top K groups for coverage using the proposed algorithms on the same dataset. Table 1 shows the coverage attained by different algorithms and the time taken for different values of K.

Note that, the time taken to produce the K-sets is less than the total time taken for mining all maximal

[2]`http://www.citymayors.com/statistics/largest-cities-density-125.html`

Table 1: Coverage attained by the proposed algorithms and the time taken in brackets (Seconds) for different values of K on Random Waypoint simulator data.

Algo.	K				
	20	24	25	26	30
BKC	95 (2)	99 (2)	98 (3)	100 (3)	99 (3)
DFRC	92 (0)	97 (0)	98 (0)	99 (0)	100 (0)
ItBC	93 (5)	99 (5)	98 (0)	99 (0)	100 (0)
MAC	94 (0)	99 (0)	98 (0)	99 (0)	100 (0)

valid groups in most cases. Also, different algorithms terminate with different local maxima solutions as described in Section 3.4. This also results in an interesting scenario where the local maxima corresponding at a lower value of K may be higher than that of the one corresponding to higher value of K, which explains BKC terminating with 99 coverage with K= 30 whereas it terminated with 100 coverage with K= 26, and similar cases.

Results on Larger datasets. The plots show the coverage values of different algorithms from the instant when the first set of K groups are reported by them. All algorithms are given a fixed time of one hour. Size of each node is around 80KB for 5,000 users (though the size of each node is proportional to the number of users, it will not be problematic as the number of nodes needed is very low as described previously in this subsection).

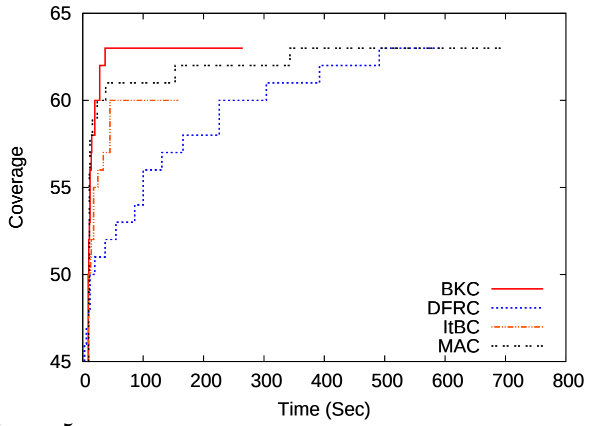

Figure 5: $NUM\_USERS : 1,000$, max_dis : 10m, and $K : 5$ on Random Waypoint simulator data.

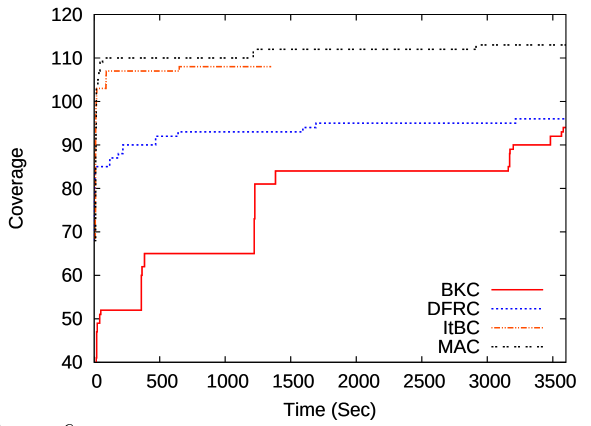

Figure 6: $NUM\_USERS : 1,000$, max_dis : 20m, and $K : 5$ on Random Waypoint simulator data.

Figures 5 & 6 display the results on a dataset consisting of 1000 users when K = 5. max_dis is set to 10 in one set of experiments and 20 in another, resulting in different group size distributions, one with low average group size and the other with high average group size respectively. Note that, when the group sizes are small (Figure 5) BKC is performing better, and when the group sizes are large (Figure 6) MAC is performing better.

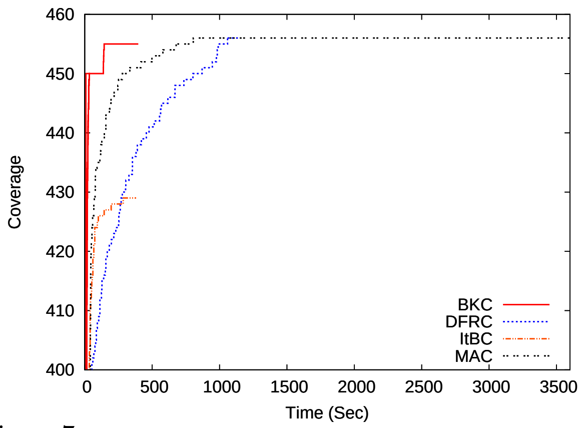

Figure 7: $NUM\_USERS : 1,000$, max_dis : 10m, and $K : 50$ on Random Waypoint simulator data.

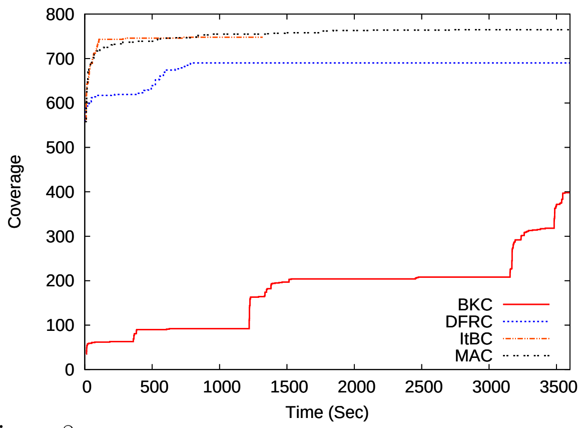

Figure 8: $NUM\_USERS : 1,000$, max_dis : 20m, and $K : 50$ on Random Waypoint simulator data.

Figures 7 & 8 display the results on the same dataset consisting of 1000 users when K = 50. Once again, max_dis is set to 10 in one set of experiments and 20 in another, resulting in different group size distributions, on the lower side and on the higher side respectively. Note that, when the group sizes are small (Figure 7) BKC is performing better, and when the group sizes are large (Figure 8) MAC is performing better, similar to what is noted previously.

We now repeat similar set of experiments with larger number of users, 5000. Figures 9 and 10 display the results obtained for different values of max_dis. Note that, the relative performance of various algorithms is similar to what has been observed with the 1000 user dataset, depending only on the group size distribution.

We would like to mention that, when higher values of K are used such that all the users are covered, the algorithms terminated within very few minutes (details not given here), showing their efficacy. The hardness of the problem effects their performance most when the K value is such that a strict subset of users can only be covered which needs to be maximized, as one may rightly expect.

For even larger number of users, the algorithms slow down a bit but we have not noticed memory problems on our machine up-to 15000 users which is remarkable. However, for best performance, its advisable to devise efficient parallel or distributed methods for $NUM\_USERS > 5,000$.

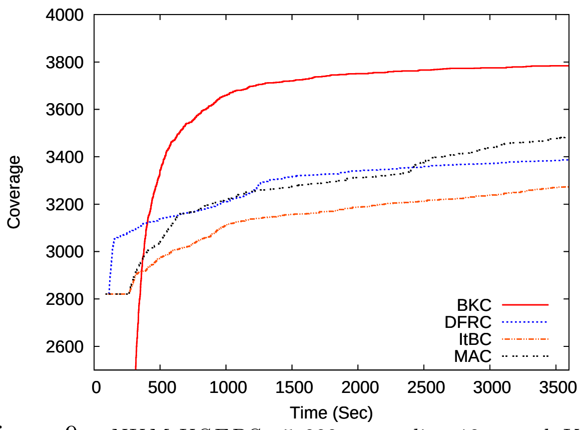

Figure 9: $NUM\_USERS : 5,000$, max_dis : 10m, and K : 500 on Random Waypoint simulator data.

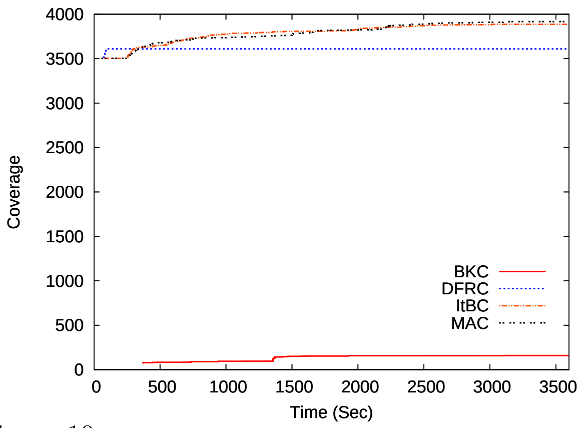

Figure 10: $NUM\_USERS : 5,000$, max_dis : 20m, and K : 300 on Random Waypoint simulator data.

Experience with Clustering based Initialization. We have also experimented with clustering methods for obtaining a good quality K-set quickly which could be used for initialization of proposed algorithms. We have used a variation of K-means algorithm where an user is clustered with a node (set of users) with whom (any member of the set) its total length of *valid segments* is highest. Other nearness criteria, such as, cluster with most number of valid edges (weight $\geq min\_wei$), highest sum of weights, highest sum of weights ratio (with cluster size), and number of valid edges ratio, were also tried. After obtaining clusters, we ran DFMax with ordering based on clusters, each time, nodes of a different cluster being put at the front. Highest weight (shared with any user of the cluster) criterion turned out to be the best one among clustering based methods. However, the basic DFRC produced a better starting point in lesser time. This is due to valid group definition being not so close to clustering mechanism which does not help produce a similar group as that of a cluster, making clustering ineffective.

4.2 With Oporto Simulator Data

Now, we present the results obtained using the Oporto simulator (Saglio & Moreira 2001). It tries to model the movement of fishing ships where the ships go in the direction of the most attractive shoals of fish while trying to avoid storm areas. Shoals are themselves attracted by plankton areas. Ships are moving points; plankton or storm areas are regions with fixed center but moving shape; and shoals are moving regions. We used the default parametric settings which come with the simulator provided by them online to generate datasets consisting of 1000 shoals, and mined for K-sets with different max_dis values.

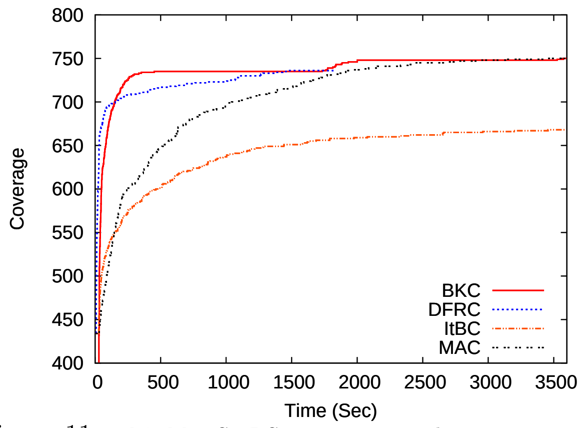

Figure 11: $NUM\_USERS : 1,000$, max_dis : 30000, and K : 100 on Oporto Simulator Data.

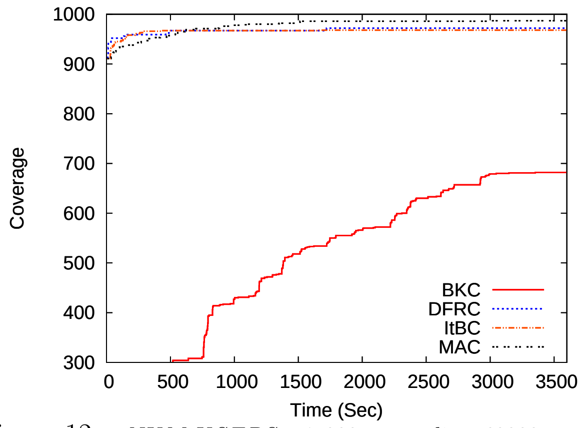

Figure 12: $NUM\_USERS : 1,000$, max_dis : 60000, and K : 100 on Oporto Simulator Data.

Figures 11 and 12 show the results obtained on a 1000 user dataset when max_dis is 30000 and 60000 respectively ($NUM\_TIMEPOINTS : 3,001$, min_dur : 3, min_wei : 1%). Note that, these two scenarios correspond to the group size being smaller in the first case and the group size being larger in the second case. And as expected, BKC performs better in the first case and MAC performs better in the second case.

4.3 With Brinkhoff Simulator Data

Here, we present the results obtained on the data generated using the network based mobility simulator of objects proposed by Brinkhoff (Brinkhoff 2002). Salient aspects of the simulator include, the maximum speed and the maximum capacity of connections, the influence of other moving objects on the speed and the route of an object, the influence of external events, time-scheduled traffic, etc. We have run their data generator which is provided online with maximum value for the number of object classes and got the data for 105 points (objects) moving inside a single time interval of 21 timepoints on the Oldenburg city map.

For our experiments, min_dur and min_wei are set to 3 timepoints. With max_dis =3000, a total of 204 groups are mined by VGBK whose average group size is 4.166667 in 0 seconds. Their size distribution is given by: 1: 10, 2: 14, 3: 38, 4: 60, 5: 52, 6: 17, 7: 8, 8: 5, where $x : y$ denotes that there are y groups of size x. Table 2 shows the results obtained with this *max_dis* value for different values of K. On such small datasets, all the algorithms are fairly competitive.

Table 2: Coverage attained by the proposed algorithms and the time taken in brackets (Seconds) for different values of K, when *max_dis* is 3000 on Brinkhoff Simulator Data.

Algo.	K				
	25	30	35	40	45
BKC	89 (0)	95 (0)	97 (0)	101 (1)	105 (0)
DFRC	87 (0)	93 (0)	98 (0)	103 (0)	105 (0)
ItBC	88 (0)	93 (0)	98 (0)	103 (0)	105 (0)
MAC	87 (0)	93 (0)	98 (0)	103 (0)	105 (0)

With *max_dis* =6000, a total of 469 groups are mined by VGBK whose average group size is 10.650320 in 67 seconds. Their size distribution is given by: 1: 10, 3: 4, 4: 9, 5: 20, 6: 20, 7: 36, 8: 31, 9: 43, 10: 39, 11: 63, 12: 59, 13: 47, 14: 20, 15: 12, 16: 11, 17: 22, 18: 19, 19: 4, where $x : y$ denotes that there are y groups of size x. Table 3 shows the results obtained with this *max_dis* value for different values of K. Note that, the time taken by BKC is significantly larger than the other algorithms owing to the large sized groups present in the search space, which is consistent with our previous observation that the performance of BKC degrades with the presence of large sized groups (which it actually inherits from VGBK (Wang et al. 2008)).

Table 3: Coverage attained by the proposed algorithms and the time taken in brackets (Seconds) for different values of K, when *max_dis* is 6000 on Brinkhoff Simulator Data.

Algo.	K				
	10	15	20	25	30
BKC	86 (78)	94 (122)	98 (73)	103 (82)	105 (74)
DFRC	85 (5)	94 (0)	97 (0)	102 (0)	105 (0)
ItBC	82 (3)	93 (3)	97 (0)	102 (0)	105 (0)
MAC	85 (2)	94 (0)	97 (0)	102 (0)	105 (0)

4.4 With Geolife Trajectories Data

Finally, we present the results on a real dataset, albeit small. The GPS trajectory dataset was collected in (Microsoft Research Asia) Geolife project (Microsoft-ResearchAsia 2012) of 182 users in a period of over five years (from April 2007 to August 2012). For our experiments, we have processed this data to identify a period of 30 days in which most users were active. We found that in the peak month, the number of users active were 41. We processed the data of these 41 users for that month at a sampling rate of 100 timepoints per day (resulting in a total of 3000 timepoints).

Table 4: Coverage attained by the proposed algorithms and the time taken in brackets (Seconds) for different values of K on Geolife Trajectories data.

Algo.	K			
	10	15	20	25
BKC	31 (0)	37 (0)	40 (0)	41 (0)
DFRC	31 (0)	39 (0)	41 (0)	41 (0)
ItBC	32 (1)	39 (0)	41 (0)	41 (0)
MAC	32 (0)	39 (0)	41 (0)	41 (0)

We set the values of *min_dur* and *min_wei* to 2 timepoints (corresponds to approximately 30 Minutes time; using larger values such as *min_wei* $= 1\%$ resulted in just a single valid group of size 2) and *max_dis* to 500 (using smaller values reduced the number of groups drastically). VGBK reported a total of 154 maximal valid groups with average group size of 3.097403 in 0 seconds. The size distribution is as follows: 1: 1, 2: 44, 3: 61, 4: 37, 5: 9, 6: 2, where $x : y$ indicates that there are y groups of size x. Table 4 presents the results obtained upon running the proposed algorithms on this dataset with the above parameters. We note that all the algorithms are fairly competitive on this small real dataset. Also, the coverage estimate given in Section 4.1 can be observed to be working well in this case. For example, for K = 10, K * average group size = 31 and so is the coverage attained.

Overall Observations: We observe the following points from the above sets of experiments (and based on related experiments carried out by us which are not shown due to space constraint):

1. BKC gives the best performance when the expected group size is small ($\leq$ 10). For example, in cases where one is set out to mine groups of close friends, using a small *max_dis* value or large *min_wei* value. However, its performance degrades severely when the expected group size is large.

2. MAC and ItBC give the best performance when the expected group size is large ($>$ 10). For example, in cases where one is set out to mine groups of people living/working in a nearby region, using a large *max_dis* value; or in densely populated regions.

We also measured the time taken for mining all maximal valid groups using VGBK (Wang et al. 2008) which ranged from 2 Minutes to 30 Minutes on a 1000 user dataset depending on the other input parameters, the time taken increasing further with the group size, no. of users (up-to hours and days). We have got 219057 maximal valid groups for the 1000 user Oporto dataset where the algorithms successfully found top K groups for coverage in quick time (showing almost no mining overhead before producing the first solution or in improving it).

5 Conclusion

In this paper, we introduced the problem of finding K groups with maximum coverage in the context of spatio-temporal data mining. We have proposed several efficient methods to solve this hard problem, based on existing mining techniques, depth-first search and heuristic-search techniques. The methods can work within the given amount of memory, produce solutions in anytime manner and guarantee terminating with maximal solutions. This paper also demonstrates a new way of applying best-first search methods to solve newer and more difficult problems. Experimental results show that different methods are effective under different conditions which are clearly categorized for selection as per user application. The methods can also be used when mining groups with other group definitions involving other types of data. The problem also opens up several interesting future directions, including better algorithms, parallel adaptations, and extending to other domains such as frequent itemsets.

Acknowledgment

This work was carried out as part of the Xerox India Innovation university research projects at Indian Institute of Technology Kharagpur. We thank Dr. Nathan Gnanasambandam of Xerox for his valuable comments and suggestions.

References

Afrati, F., Gionis, A. & Mannila, H. (2004), Approximating a collection of frequent sets, *in* 'Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining', KDD '04, ACM, New York, NY, USA, pp. 12–19.

Agarwal, R. C., Aggarwal, C. C. & Prasad, V. V. V. (2000), Depth first generation of long patterns, *in* 'Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining', KDD '00, ACM, New York, NY, USA, pp. 108–118.

Agrawal, R. & Srikant, R. (1994), Fast algorithms for mining association rules in large databases, *in* 'Proceedings of the 20th International Conference on Very Large Data Bases', VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 487–499.

Aine, S., Chakrabarti, P. P. & Kumar, R. (2007), AWA* - A window constrained anytime heuristic search algorithm, *in* 'IJCAI', pp. 2250–2255.

Bisiani, R. (1987), 'Beam search', *Encyclopedia of Artificial Intelligence* pp. 56–58.

Brinkhoff, T. (2002), 'A framework for generating network-based moving objects', *Geoinformatica* **6**(2), 153–180.

Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y.-C. & Jetcheva, J. (1998), A performance comparison of multi-hop wireless ad hoc network routing protocols, *in* 'Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking', MobiCom '98, ACM, New York, NY, USA, pp. 85–97.

Bron, C. & Kerbosch, J. (1973), 'Algorithm 457: Finding all cliques of an undirected graph', *Commun. ACM* **16**, 575–577.

Calders, T. & Goethals, B. (2002), Mining all non-derivable frequent itemsets, *in* 'Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery', PKDD '02, Springer-Verlag, London, UK, UK, pp. 74–85.

Chakrabarti, P. P., Ghose, S., Acharya, A. & Sarkar, S. C. D. (1989), 'Heuristic search in restricted memory.', *Artificial Intelligence* **41**(2), 197–221.

Davis, R., Hagey, W. & Horning, M. (2004), 'Monitoring the behavior and multi-dimensional movements of weddell seals using an animal-borne video and data recorder', *Memoirs of the National Institute of Polar Research (Japan)* **Special Issue 58**, 148–154.

Dean, T. & Boddy, M. S. (1988), An analysis of time-dependent planning, *in* 'AAAI', pp. 49–54.

Forsyth, D. R. (2006), *Group dynamics*, 4 edn, Thomson/Wadsworth, Belmont, CA.

Gouda, K. & Zaki, M. J. (2005), 'Genmax: An efficient algorithm for mining maximal frequent itemsets', *Data Min. Knowl. Discov.* **11**, 223–242.

Han, J., Wang, J., Lu, Y. & Tzvetkov, P. (2002), Mining top-k frequent closed patterns without minimum support, *in* 'Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on', pp. 211–218.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics* **4**(2), 100–107.

Hochbaum, D. S., ed. (1997), *Approximation algorithms for NP-hard problems*, PWS Publishing Co., Boston, MA, USA.

Kalnis, P., Mamoulis, N. & Bakiras, S. (2005), On discovering moving clusters in spatio-temporal data, *in* 'In SSTD', Springer, pp. 364–381.

Lauw, H. W., Lim, E.-P., Pang, H. & Tan, T.-T. (2005), 'Social network discovery by mining spatio-temporal events', *Comput. Math. Organ. Theory* **11**, 97–118.

Lee, A. J. T., Chen, Y.-A. & Ip, W.-C. (2009), 'Mining frequent trajectory patterns in spatial-temporal databases', *Inf. Sci.* **179**, 2218–2231.

MicrosoftResearchAsia (2012), 'Geolife gps trajectories', *http://research.microsoft.com/en-us/projects/geolife/.*

Ng, R. T. & Han, J. (1994), Efficient and effective clustering methods for spatial data mining, *in* 'Proceedings of the 20th International Conference on Very Large Data Bases', VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 144–155.

Pei, J., Dong, G., Zou, W. & Han, J. (2002), On computing condensed frequent pattern bases, *in* 'Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on', pp. 378–385.

Pei, J., Han, J. & Mao, R. (2000), Closet: An efficient algorithm for mining frequent closed itemsets, *in* 'ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery', pp. 21–30.

Rymon, R. (1992), *Search through Systematic Set Enumeration*, Morgan Kaufmann, pp. 539–550.

Saglio, J.-M. & Moreira, J. (2001), 'Oporto: A realistic scenario generator for moving objects', *Geoinformatica* **5**(1), 71–93.

Schafer, J. B., Konstan, J. A. & Riedl, J. (2001), 'E-commerce recommendation applications', *Data Min. Knowl. Discov.* **5**, 115–153.

Srikant, R., Vu, Q. & Agrawal, R. (1997), Mining association rules with item constraints, *in* 'KDD', pp. 67–73.

Thayer, J. T. & Ruml, W. (2010), Anytime heuristic search: Frameworks and algorithms, *in* 'SOCS'.

Tseng, V. S., Wu, C.-W., Shie, B.-E. & Yu, P. S. (2010), Up-growth: an efficient algorithm for high utility itemset mining, *in* 'Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining', KDD '10, ACM, New York, NY, USA, pp. 253–262.

Vadlamudi, S. G., Aine, S. & Chakrabarti, P. P. (2011), 'MAWA*—A memory-bounded anytime heuristic-search algorithm', *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **41**(3), 725–735.

Vadlamudi, S. G., Gaurav, P., Aine, S. & Chakrabarti, P. P. (2012), Anytime column search, *in* 'Australasian Conference on Artificial Intelligence', pp. 254–265.

van den Berg, J., Shah, R., Huang, A. & Goldberg, K. Y. (2011), Anytime nonparametric A*, *in* 'AAAI'.

Verhein, F. & Chawla, S. (2006), Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases, *in* 'of Lecture Notes in Computer Science', Springer, pp. 187–201.

Wang, K., He, Y. & Han, J. (2000), Mining frequent itemsets using support constraints, *in* 'Proceedings of the 26th International Conference on Very Large Data Bases', VLDB '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 43–52.

Wang, Y., Lim, E.-P. & Hwang, S.-Y. (2008), 'Efficient algorithms for mining maximal valid groups', *The VLDB Journal* **17**, 515–535.

Wang, Y., peng Lim, E. & yih Hwang, S. (2003), On mining group patterns of mobile users, *in* 'Proceedings of the 14th International Conference on Database and Expert Systems Applications—-DEXA 2003', pp. 287–296.

Xin, D., Cheng, H., Yan, X. & Han, J. (2006), Extracting redundancy-aware top-k patterns, *in* 'Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining', KDD '06, ACM, New York, NY, USA, pp. 444–453.

Xu, J. J. & Chen, H. (2005), 'Crimenet explorer: a framework for criminal network knowledge discovery', *ACM Trans. Inf. Syst.* **23**, 201–226.

Zhou, R. & Hansen, E. A. (2005), Beam-stack search: Integrating backtracking with beam search., *in* 'Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)', Monterey, CA, pp. 90–98.