Expert Systems With Applications 63 (2016) 310–323

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A fast space-saving algorithm for maximal co-location pattern mining

Xiaojing Yao[a], Ling Peng[a,*], Liang Yang[b], Tianhe Chi[a]

[a] Lab of Spatial Information Integration, Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, No. 20 North, Datun Road, Chaoyang District, Beijing 100101, China
[b] School of Information Engineering, Tianjin University of Commerce, No. 409, Guangrong Road, Beichen District, Tianjin 300134, China

ARTICLE INFO

Article history:
Received 3 March 2016
Revised 5 June 2016
Accepted 3 July 2016
Available online 6 July 2016

Keywords:
Spatial data mining
Maximal co-location patterns
Sparse undirected graph
Condensed tree
Hierarchical verification

ABSTRACT

Real space teems with potential feature patterns with instances that frequently appear in the same locations. As a member of the data-mining family, co-location can effectively find such feature patterns in space. However, given the constant expansion of data, efficiency and storage problems become difficult issues to address. Here, we propose a maximal-framework algorithm based on two improved strategies. First, we adopt a degeneracy-based maximal clique mining method to yield candidate maximal co-locations to achieve high-speed performance. Motivated by graph theory with parameterized complexity, we regard the prevalent size-2 co-locations as a sparse undirected graph and subsequently find all maximal cliques in this graph. Second, we introduce a hierarchical verification approach to construct a condensed instance tree for storing large instance cliques. This strategy further reduces computing and storage complexities. We use both synthetic and real facility data to compare the computational time and storage requirements of our algorithm with those of two other competitive maximal algorithms: "order-clique-based" and "MAXColoc". The results show that our algorithm is both more efficient and requires less storage space than the other two algorithms.

1. Introduction

Some types of features in real-world data inevitably have spatial neighbor relationships. Instances of these feature types are frequently located in the same locations, thereby creating a pattern that can be used to discover interesting phenomena in the natural or social world and facilitate decision-making by humans. For example, fire emergencies demonstrably occur more frequently around schools and residential buildings than around other urban structures. Using this pattern information, urban planners can locate additional fire-fighting equipment around such locations to prevent extensive damage (Fan & Luo, 2013). As a member of the data mining family, co-location represents an essential tool for finding these types of spatial patterns, and it is extensively used in many application domains, including but not limited to species distribution analysis (Shekhar & Huang, 2001; Sierra & Stephens, 2012), mobile services (Yoo, Shekhar, Kim, & Celik, 2006), public safety (Leibovici, Claramunt, Le Guyader, & Brosset, 2014), urban facility distribution analysis (Yu, 2016), and environmental management (Akbari, Samadzadegan, & Weibel, 2015).

1.1. Context

Co-location was originally proposed by Shekhar and Huang (2001) as a way to discover the rules of ecological species distributions. This approach was followed by an *Apriori*-like method called the "join-based algorithm", which created a precedent for pattern-mining technology that was later used for spatial applications. However, the efficiency and storage requirements of this method became unfavorable as the amount of data increased. Efficiency and storage are always core issues in co-location. In this section, we focus mainly on related works that address these two issues.

Some studies indicate that the join-based method becomes inefficient with increasing data size because it requires massive amounts of instance-connecting operations. Thus, to reduce the number of connections, several scholars have presented faster algorithms, some of which provide satisfactory results. For example, the partial-join approach proposed by Yoo and Shekhar (2004) build a set of disjoint cliques in spatial instances to identify neighbor relationships. However, this approach is affected by the distribution of the data, or more precisely, the number of cut neighbor relations. Arunasalam, Chawla, Sun, and Munro (2004) achieved the mining of complex relationships in a bigger dataset. Subsequently, Yoo and Shekhar (2006) also presented an improved algorithm called "join-less," which inputs the neighbor

* Corresponding author.
E-mail addresses: yaoxj@radi.ac.cn, yaoxiaojing2008@126.com (X. Yao), pengling@radi.ac.cn (L. Peng), yangliang@iie.ac.cn (L. Yang), chith@126.com (T. Chi).

http://dx.doi.org/10.1016/j.eswa.2016.07.007
0957-4174/

relationships into a compressed star model. However, both of the above methods are based on *Apriori* tactics, making it difficult to avoid large connecting processes; therefore, their efficiency improvements are limited when addressing big or dense data. Moreover, the large amount of storage space required by instance tables presents a challenging problem. Recently, some scientists have found that maximal clique discovery from graph theory can be used to address the co-location issue and have presented inventive methods to generate maximal instance cliques (Al-Naymat, 2013; Kim, Kim, & Kim, 2011). These methods significantly increase the computational speed. However, listing all maximal cliques is inherently an NP-hard problem, resulting in exponential time consumption. Although many maximal clique-mining algorithms have been explored (Agrawal & Srikant, 1994; Bron & Kerbosch, 1973; Cazals & Karande, 2008; Johnston, 1976; Makino & Uno, 2004; Tomita, Tanaka, & Takahashi, 2006), finding all maximal instance cliques in a massive spatial dataset remains a difficult problem. Another problem concerns the enormous memory requirements faced during the calculations. To mitigate the dilemma between space requirements and time consumption, algorithms such as the "density clustering" (Huang, Zhang, & Zhang, 2008; Lee, Qu, & Lee, 2012) and "grid differential" (Yao, Wang, Chen, & Zou, 2015) algorithms make approximations for spatial instance locality or connecting definitions. However, these methods can lose instance connections and miss some significant co-locations. Currently, the prevailing co-location algorithms run in parallel (Yoo & Boulware, 2013; Yoo, Boulware, & Kimmey, 2014) and exhibit good performance when applied to big data. However, these algorithms require distributed programming frameworks and are somewhat uneconomical. The recommended solution is to reduce the time and storage requirements before modifying the algorithms to run in parallel.

From the above, it is apparent that alleviating storage redundancy and improving efficiency for co-location in a stand-alone device without losing precision remains a thorny task. To address these problems, a maximal framework was developed and demonstrated to be superior (Wang, Zhou, Lu, & Yip, 2009; Yoo & Bow, 2011). This framework contains two main steps: (1) candidate maximal co-location acquisition from the size-2 instance table and (2) final maximal co-location acquisition from the candidates obtained in the first step. The basic concept was first proposed by Wang et al. (2009) using an FP-growth-like (Han, Pei, & Yin, 2000) approach called the "order-clique-based" (OCB) algorithm (Wang et al., 2009). The OCB algorithm uses a pruning strategy to return maximal co-locations of which the subsets with cardinal numbers larger than one represent all co-location patterns. Specifically, it compresses the neighbor relationships of spatial instances and prevalent size-2 co-locations into extended prefix-tree structures and then uses the order-clique-based approach to construct long-size candidate maximal co-locations and their instance cliques to obtain the final results. Wang et al. (2009) and Boinski and Zakrzewicz (2014) verified the efficiency and storage advantages of this mining method. Subsequently, other researchers used this maximal framework and obtained satisfactory results (Yoo & Bow, 2011).

1.2. Challenges

Among existing maximal methods, we are aware of two deficiencies that become prominent when datasets are large or dense. These deficiencies present opportunities for further improvements in two ways:

- Current methods use time-consuming strategies, such as FP-growth-like strategies, which contain a number of expensive tree-based operations such as branch chopping and grafting during the discovery of long candidates or instance cliques. To our knowledge, both of these operations strongly impact the execution speed of tree-based algorithms.
- Current methods usually build a duplicate initial structure, such as the prefix-tree in the OCB algorithm, which reserves instance pairs or prevalent size-2 co-location patterns for further clique acquisition. Building this structure requires extra space during execution; however, most of the nodes in this structure are eventually removed or reorganized.

1.3. Contributions

To address the above two problems, we propose a sparse-graph and condensed tree-based maximal co-location algorithm (hereafter referred to as the SGCT algorithm) to reduce the computation time and storage requirements. We mainly use two improved strategies:

First, we induce a fast, maximal-clique mining algorithm applied to a sparse undirected graph to find candidate maximal co-locations. This strategy makes our method more efficient than most other maximal algorithms under the same step and input conditions. We observed that the candidate maximal co-locations can be extracted by detecting all maximal cliques in an undirected graph when the neighbor relationships of prevalent size-2 co-locations are abstracted as edges and when the relevant features are abstracted as vertexes. The maximal clique idea is not new in co-location issues; it has been used primarily for finding complete instance cliques in an entire space. Because the time complexity of maximal clique algorithms greatly depends on the vertex number, the efficiency improvement is limited in large or dense data despite the addition of some spatial optimizations. In real-world data, feature types are generally much less common than spatial instances; thus, the maximal clique idea can be better adapted to obtain long-size candidates than to find instance cliques in an entire space. Furthermore, the prevalence threshold constricts the lengths of candidate maximal co-locations to a limited ceiling; i.e., the graph extracted from prevalent size-2 co-locations tends to be sparse. Therefore, we introduce a parameter called "degeneracy" to describe the sparseness of a graph (Eppstein, Löffler, & Strash, 2010). In that case, we settle for the exponential time complexity of the traditional Bron–Kerbosch maximal clique algorithm (Bron & Kerbosch, 1973) by applying parameterized complexity with "pivot" (Tomita et al., 2006) and "degeneracy" (Eppstein & Strash, 2011; Eppstein et al., 2010) interferences. This modification can improve the inner-and outer-level recursive procedures of the Bron–Kerbosch algorithm and reduce the time complexity of obtaining candidate maximal co-locations to a near polynomial function with small vertex size and low degeneracy parameters.

Second, we adopt a hierarchical verification approach to construct a condensed instance tree that stores instance cliques for each long-size candidate maximal co-location. The method substantially reduces the execution time and storage requirements for acquiring satisfactory long-size instance cliques. An instance clique is a group of type-specific instances in which any instance pair has neighbor relevance and the type collection matches a particular candidate maximal co-location. We abandon current common methods that construct an initial structure of neighbor instance pairs before the pruning stage. Instead, for each maximal co-location, we adopt a hierarchical verification approach to directly obtain a tree structure containing all instance cliques. This structure is called "condensed tree" because the construction process consists primarily of simple and basic tree operations such as adding or removing nodes. Compared with traditional methods, it omits most of the expensive tree-based operations such as grafting or chopping branches and also reduces the ceiling storage requirements of the algorithm. In a sense, this improved method for acquiring instance cliques is condensed not only in space but also in time.

Table 1
The important notations used in this article.

Notation	Definition
F	A set of feature types
S	A set of spatial instances
r	Distance threshold
Min-prev	Prevalence threshold
InsTable2	Size-2 instance table
E	An edge set of neighbor relationships of the prevalent size-2 co-locations
V	A vertex set of feature types existing in the prevalent size-2 co-locations
G	The size-2 co-location graph extracted from E and V
$N(v)$	A set of neighborhoods of vertexes v
CP_m	A set of candidate maximal co-locations
C_m	A candidate maximal co-location
$InsC_m$	Ordered instance clique of C_m
CInsTree	Condensed instance tree of C_m

To demonstrate the improvements obtained by our algorithm, we tested it both with synthetic datasets and real data collected in 2014 from facilities in Beijing, China. The results show that the SGCT algorithm is more efficient, saves space, and is less sensitive to the threshold strategy, data size and density than are other competitive co-location methods such as the OCB (Wang et al., 2009) and MAXColoc (Yoo & Bow, 2011) algorithms.

The remainder of this paper is organized as follows. Section 2 presents our mining algorithm using sparse graph theory and the condensed tree-based method. The time and storage complexities are also analyzed in Section 3. The performance tests are discussed in Section 4. Section 5 provides a conclusion of our work and suggests future directions.

2. Method

This section presents the details of the SGCT algorithm. Table 1 lists the important notations used in this article.

In the following, we present the formalized expression of our mining algorithm. We assume a set of feature types $F=\{f_1, f_2,..., f_m\}$ and a spatial dataset $S=\{s_1, s_2,..., s_n\}$ endowed with the familiar Euclidean distance function. Each instance s_i contains information about its type and location with X and Y coordinates. Specially, the set S_{f_i} contains instances of type f_i. In addition, two constrained parameters, namely, the distance threshold r and the prevalence threshold *Min-prev*, are given as a priori information. The task is to discover the prevalent maximal co-locations, which concisely represent all prevalent co-location patterns.

Example 1. In Fig. 1, we give simulated spatial instances of five types ($F=\{A,B,C,D,E\}$) to illustrate how the SGCT algorithm works.

2.1. Overview

This section presents a flowchart of the SGCT algorithm. As shown in Fig. 2, this algorithm has four steps in the following flow chart:

Step 1 (size-2 instance table construction): Based on the distance threshold r, we construct a two-dimensional hash table, i.e., the size-2 instance table $InsTable_2$, that stores different-type instance pairs that have neighbor relationships in space (Shekhar & Huang, 2001; Yoo & Shekhar, 2004, 2006). These instance pairs are indexed by their types, which are regarded as the candidate size-2 co-locations.

Example 2. If the distance threshold r is set to 2, we connect instance pairs whose distances are within r and acquire $InsTable_2$. As shown in Fig. 3, The candidate size-2 co-locations $\{A,B\}$,$\{A,C\}$... are marked by "$*$" in a hash table structure that contains instance pairs

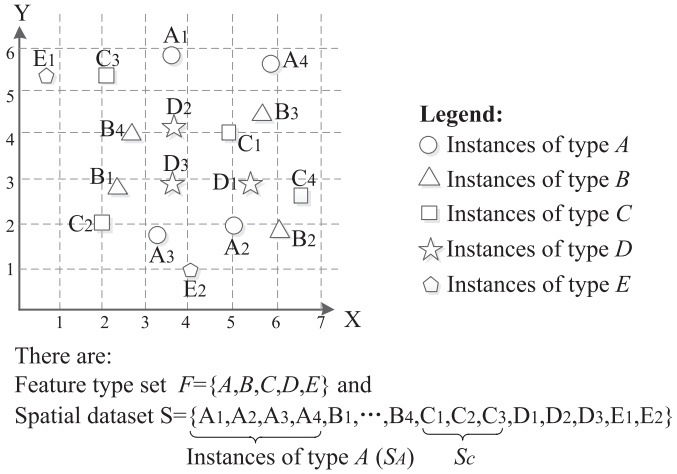

Fig. 1. The simulated instances of five types.

indexed by their corresponding types, such as $InsTable_2(A,B)$ on the right side of the figure.

Step 2 (prevalence index calculation for prevalent size-2 co-locations): We calculate the prevalence index of each candidate size-2 co-location and select those whose prevalence indices are not smaller than the pre-determined prevalence threshold *Min-prev* as the prevalent size-2 co-locations (Shekhar & Huang, 2001).

Example 3. The prevalence threshold *Min-prev* is set to 0.3. Based on Fig. 3, we calculate the prevalence indices of all candidate size-2 co-locations and find that the prevalent size-2 co-locations are $\{A,B\}$, $\{A,C\}$, $\{A,D\}$, $\{A,E\}$, $\{B,C\}$, $\{B,D\}$, $\{C,D\}$ and $\{C,E\}$.

Step 3 (candidate maximal co-location generation): We use a modified maximal clique algorithm of a sparse graph to find all candidate maximal co-locations from the prevalent size-2 co-locations.

Step 4 (pruning process for prevalent maximal co-locations): Using a pruning framework (Wang et al., 2009), we obtain the final maximal co-locations from the candidates obtained in the preceding step. Specifically, for each candidate maximal pattern, we acquire its instance cliques (Step 4.1) and calculate its prevalence index (Step 4.2). If the prevalence index is not smaller than the prevalence threshold, we reserve the candidate as a real co-location pattern (Shekhar & Huang, 2001). Otherwise, we supplant it by its subsets. In step 4.1, we adopt a hierarchical verification approach to construct a condensed tree of stored instance cliques for each long-size candidate. In this way, the instance cliques can be easily derived from this structure.

Of the above four steps, Steps 3 and 4.1 are innovative and are described in the upcoming two sections.

2.2. Generation of candidate maximal co-locations

To yield the candidate maximal co-locations, we first provide the following definitions related to this procedure.

Definition 1. (size-2 co-location graph). If the neighbor relationships of the prevalent size-2 co-locations are regarded as edges $E=\{e_1,...,e_\upsilon\}$ and the feature types appearing in the prevalent size-2 colocations are regard as vertexes $V=\{v_1,...,v_\lambda\}$, where υandλ are numbers of edges and vertexes, respectively, then the size-2 co-location graph can be modeled as an undirected graph $G=(E,V)$, which is stored in a list data structure in ascending order. The set $N(v_i)$ is the neighborhoods of vertex v_i and can be

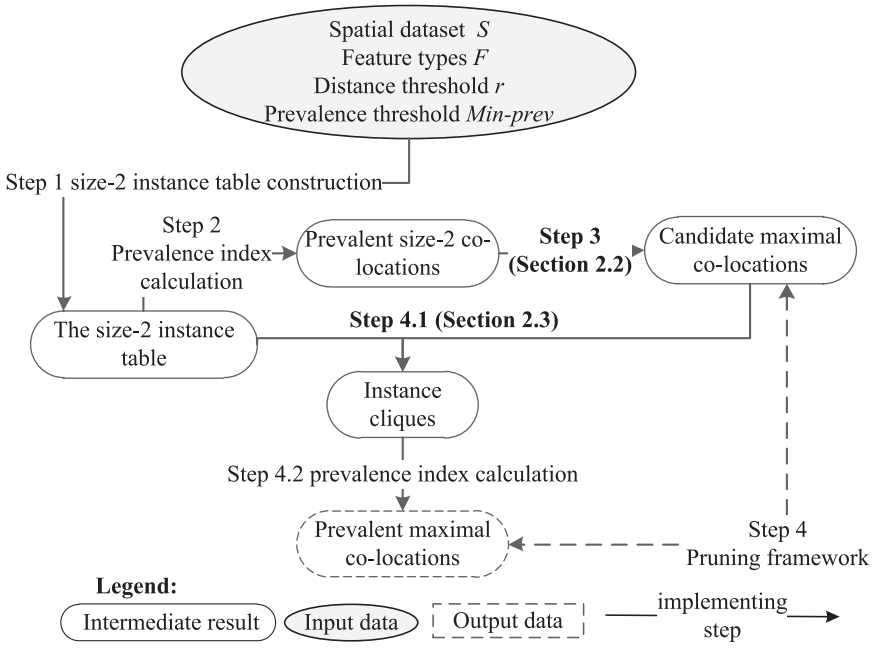

Fig. 2. The flow of the SGCT algorithm.

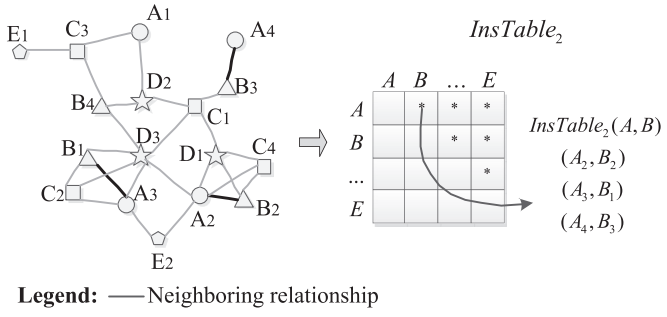

Fig. 3. The structure of the size-2 instance table based on the instances in Fig. 1.

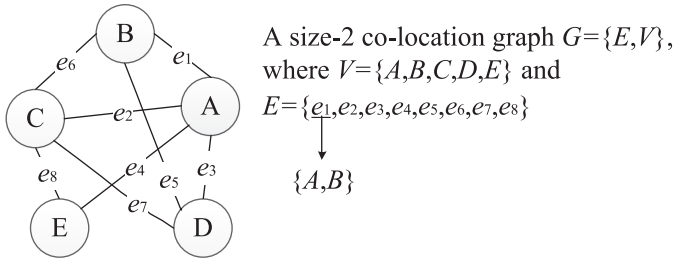

Fig. 4. The illustration of the size-2 co-location graph *G*.

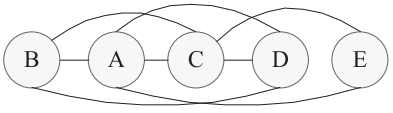

Fig. 5. The degeneracy order of *G* in Fig. 4.

defined as

$$N(v_i) = \{w | \{v_i, w\} \in E\} \quad (1)$$

Based on the *G*, the task is to find all maximal cliques, which we term "candidate maximal co-locations". These cliques are denoted by $CP_m = \{C_{m1}, \ldots, C_{m\delta}\}$, where δ is the number of CP_m. Each item C_m is defined below.

Definition 2. (candidate maximal co-location). The candidate maximal co-location C_m is composed of ordered types with two properties: each pair of types in C_m is connected by an edge, and no additional types can be added to C_m while preserving its complete connectivity.

Example 4. Using the prevalent size-2 co-locations in Example 3, we give a diagram of the size-2 co-location graph *G* in Fig. 4. We can infer that {*A, B, C, D*} is a case of a candidate maximal co-location based on the above definitions.

In the previous paragraph of Section 1, we stated that the size-2 co-location graph can be treated as a sparse graph because of the prevalence filtering. Thus, based on the Bron–Kerbosch algorithm, which is a recursive backtracking process for obtaining all maximal cliques from an undirected graph, we introduce a variant algorithm mixed with a graph-sparseness consideration and pivot selection to improve the efficiency of finding all candidate maximal co-locations. The sparseness of a graph is described by "degeneracy" (Eppstein et al., 2010), defined as:

Definition 3. (degeneracy of a graph). The degeneracy of a graph *G* is the smallest value *k* such that every nonempty sub-graph of *G* contains a vertex of degree at most *k*. That is, the size of the maximal clique may not exceed $k + 1$.

Example 5. We reorder the vertexes of *V* in Fig. 4 and acquire a sequence B → A → C → D → E, from which we see that each vertex has at most three neighbors to its right (see Fig. 5). Thus, the vertex of any sub-graph has degree at most 3 and the degeneracy of *G* is 3. We term the above sequence in which each vertex has *k* or fewer neighbors coming later "degeneracy ordering" in the remaining sections.

Based on degeneracy ordering, we give the pseudocode in Algorithm 1 (CMCP) to obtain all candidate maximal co-locations.

Algorithm 1 comprises inner recursion (lines 7–15) and outer recursion (lines 1–6) that return all candidate maximal

co-locations that contain all vertices in CP_m. In contrast to the original Bron–Kerbosch algorithm, Algorithm 1 uses two improved strategies: the first is a heuristic strategy called "pivoting" selection (Tomita et al., 2006; line 10). More concretely, as to a candidate maximal co-location pattern, it must contain the vertex u or its non-neighboring vertexes. Otherwise, u should be added to enlarge this pattern (lines 12–14). Thus, the algorithm only selects u and its non-neighboring vertexes (line 11) to reduce the number of inner recursive calls. The second strategy is called the "degeneracy" strategy. The degeneracy ordering of V is denoted by $v_1^*, v_2^*, \ldots, v_\lambda^*$ (line 2). In each outer recursion (lines 3–5), the number of vertexes waiting for verification will not exceed k. Thus, the number of outer recursions can be reduced (Yao et al., 2015; Yoo & Boulware, 2013). Consequently, the time complexity of the variant algorithm is limited to $O(k\lambda 3^{k/3})$ and is quite satisfactory for graphs with small degeneracy values.

Next, we will prove the correctness of the CMCG algorithm.

Theorem 1. *The CMCG algorithm can find all candidate maximal co-locations.*

Proof. Based on Tomita et al. (2006), the Bron–Kerbosch algorithm using pivoting selection can obtain all maximal cliques containing all vertices in K, some vertices in M, and no vertices in T, without duplication. The degeneracy strategy verifies each vertex of G in degeneracy ordering. If v_i^* is an earlier vertex of this order and a member of a maximal clique C_m, C_m will be put into CP_m after finishing the inner recursion for v_i^*. When the outer recursion traverses to other latter vertexes of C_m, v_i^* will be in T. It can be concluded that C_m will not be put into CP_m again. Thus, the CMCG algorithm can find all maximal cliques from G and will not be trapped into an endless loop. Moreover, Wang et al. (2009) has proved that the candidate maximal co-locations, i.e., maximal cliques, contain all prevalent maximal co-locations, so the CMCG algorithm is correct. □

2.3. Condensed instance tree construction

In this section, we describe a new method for obtaining instance cliques of C_m. This method greatly reduces the time and storage requirements of current time-consuming strategies such as the FP-growth-like method used in the OCB algorithm. First, we provide the definitions involved this procedure.

Definition 4. (ordered instance clique of C_m). Given a candidate maximal co-location C_m, its ordered instance clique $InsC_m$ is a group of spatial instances that satisfy the following conditions.

- The size of $InsC_m$ is equal to that of C_m, and the type of each instance in $InsC_m$ is the same as that of C_m in the corresponding order.
- Instances of any instance pair in $InsC_m$ are spatially adjacent and can be found in the size-2 instance table $InsTable_2$.

Example 6. In Fig. 3, given a candidate maximal co-location $\{A,B,C,D\}$, $\{A_3,B_1,C_2,D_3\}$ is a corresponding ordered instance clique based on Definition 4.

Definition 5. (condensed instance tree of C_m). Given a candidate maximal co-location C_m, the condensed instance tree *CInsTree* is a compressed construction that compresses all ordered instance cliques of C_m.

The entire pseudocode for generating a condensed instance tree from the candidate co-location pattern C_m and the size-2 instance table $InsTable_2$ is shown in Algorithm 2.

Algorithm 2 is a finite iterative process. A cursor i initialized to 1 is used for recording the iteration number. First, we initialize the parameter *CInsTree* and create a root "CIT" for it. The root is initialized to level 0 (line 1). Then, the tree constructing process, which is divided into two steps, begins as follows: When i is equal to 1, the process implements the first step; otherwise, it executes the second step. The iteration repeats until there is no node in the current level i or until i is one less than the size of C_m.

Step 1 (lines 3–11): For each instance pair of the first two feature types of C_m, we determine whether the first element of the current instance pair exists in the first level of *CInsTree* (denoted by $CInsTree_1$). If so, we add the second element as a child node to the corresponding node at level 1; otherwise, we create a sub-tree made up of the current instance pair and graft it to the root of *CInsTree*. When that occurs, the *CInsTree* has two levels. It can be inferred that the instance types at each level are the same as the type of C_m that share the same index at the tree level.

Step 2 (lines 13–28): For each instance node in level i of *CInsTree*, we construct a list *EI* that contains the instances of type $C_m(i+1)$ by scanning $InsTable_2(C_m(i),C_m(i+1))$, where $C_m(i)$ is the ith type of C_m. Then, each instance pair grouped by the current instance node of *CInsTree* and any member of this list can be found in $InsTable_2(C_m(i),C_m(i+1))$. Next, for each current member in this list, we determine whether the instance pair composed of this member and every ancestor of the current instance node of *CInsTree* can be found in $InsTable_2$. If so, we add the current list member as a child node to the current instance node.

The above process highlights two properties of *CInsTree* as follows:

Property 1. The depth of *CInsTree* is less than or equal to the size of C_m. The root is considered level 0 with no appended instance. Except for level 0, the types of instances at each single level i are the same as $C_m(i)$.

Property 2. The number of ordered instance cliques of C_m is equal to the number of nodes at level $size(C_m)$. If the depth of *CInsTree* is less than $size(C_m)$, then there is no satisfactory ordered instance clique. A final ordered instance clique is actually the group of path nodes from level $size(C_m)$ to level 1. Thus, we can easily acquire all instance cliques by collecting all these paths for the subsequent prevalence filtering.

To make this process more intelligible, we give an illustration of constructing the condensed instance tree for the C_m $\{A,B,C,D\}$ using the simulated data in Fig. 1.

Example 7. As shown in Fig. 6, (a) corresponds to the Step 1 sub-procedure. We build a two-level tree containing instances of type A and type B according to $InsTable_2(A,B)$. Fig. 6(b, c) shows cases of the Step 2 sub-procedure. In (b), for nodes of type B in level 2, we search for their neighbor instance partners of type C from $InsTable_2(B,C)$ and store them in a list $EI = [C_4,C_2,C_1]$. Then, hierarchical verification is performed and two nodes (C_4 and C_2) are obtained from the third level. Fig. 6(c) is a bit more sophisticated for performing two-level verification than (b). By collecting path nodes from level 1 to level 4, we finally obtain two ordered instance cliques $\{A_2,B_2,C_4,D_1\}$ and $\{A_3,B_1,C_2,D_3\}$ in Fig. 6(d).

3. Complexity analyses

In this section, we analyze the time and storage complexities of two improved strategies in our algorithm: the processes of candidate maximal co-location generation (CMCG) and condensed instance tree construction (CITC).

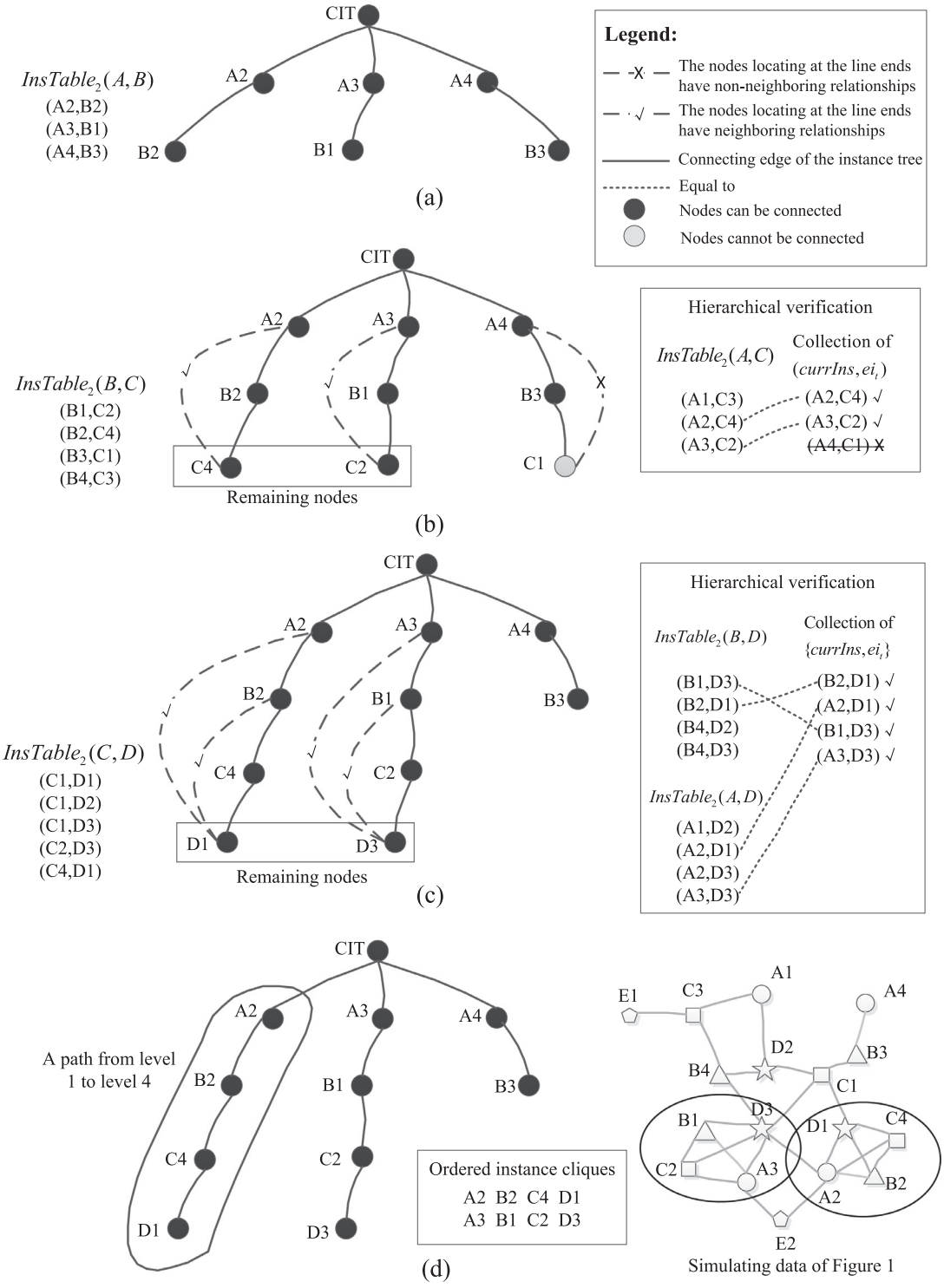

Fig. 6. Constructing the condensed instance tree for candidate {*A*, *B*, *C*, *D*}.

3.1. Time complexity

In Algorithm 1, we stated that the time complexity of CMCG is $O(k\lambda 3^{k/3})$, where k is the degeneracy of the size-2 co-location graph G and λ is the vertex number of G. In real-world applications, λ is generally smaller than the type number m of the dataset; thus, the worst time complexity of CMCG is $O(km3^{k/3})$.

Previous studies have indicated that calculating long instance clique consumes the bulk of the execution time required by the type-first maximal co-location algorithm (Wang et al., 2009). For simplicity, we take only the longest candidate maximal co-location as an example for analyzing the time complexity of CITC. Assuming that the longest candidate maximal co-location is C_L and that $N_L = \{InsTable_2(C_m(x), C_m(y))|1 \le x < y \le |C_L|\}$ consists of instance pairs with types in C_L, then the worst time complexity of CITC is

$O(f(N_L))$, where $f(N_L)$ is calculated as follows:

$$f(N_L) = \sum_{i=1}^{|C_L|-1} \left(\left| \pi_{C_L(i)} InsTable_2(C_L(i), C_L(i+1)) \right| * \sum_{j=i+1}^{|C_L|} \log_2 |InsTable_2(C_L(i), C_L(j))| \right) \quad (2)$$

In Eq. (2), $\pi_{C_L(i)} InsTable_2(C_L(i), C_L(i+1))$ denotes the instance projection of type $C_L(i)$ from instance table $InsTable_2(C_L(i),C_L(i+1))$. Generally, the cardinal number of $\pi_{C_L(i)} InsTable_2(C_L(i), C_L(i+1))$ is much smaller than $|S_{C_L(i)}|$, where $S_{C_L(i)}$ is the instance set of type $C_L(i)$. Therefore, it can be concluded that $f(N_L) << |S_{C_L}^{Max}| * \log_2 |N_L|$, where $|S_{C_L}^{Max}|$ is the maximum number of instances with a single type in C_L.

From the above conclusion, we can see that the worst time complexity of CITC satisfies $O(f(N_L)) << O(|S_{C_L}^{Max}| * log_2 |N_L|)$.

3.2. Storage complexity

The storage requirements of instance clique construction are the most costly in the maximal co-location algorithms. Our improved algorithm CITC adopts a hierarchical verification approach; thus, the peak space cost is greatly reduced. The worst space complexity of the longest candidate maximal co-location C_L is $O(g(N_L))$, where $g(N_L)$ is calculated as follows:

$$g(N_L) = \sum_{i=1}^{|C_L|-1} (|\pi_{C_L(i)} InsTable_2(C_L(i), C_L(i+1))| * |InsTable_2(C_L(i), C_L(i+1))|) \quad (3)$$

From the foregoing clarification, it can be concluded that $\pi_{C_L(i)} InsTable_2(C_L(i), C_L(i+1))$ is much smaller than $|S_{C_L(i)}|$. Therefore, we have $g(N_L) = O(|C_L| * |S_{C_L}^{Max}| * |Max(InsTable_2^{C_L})|)$.

We compute instance cliques for only single candidate iterations. Memory is released after the preceding iteration completes. Thus, the storage cost of instance clique construction never exceeds $|C_L| * |S_{C_L}^{Max}| * |Max(InsTable_2^{C_L})|$.

Summarizing, the entire algorithm is affected by the number of instances, the number of feature types, the degeneracy of size-2 co-locations, the number of instance pairs, and the number and length of candidate maximal co-locations. Specifically, $k \ll m \ll n$, meaning that the time and space complexity mainly depend on the last three items.

4. Experiment evaluations

In this section, we demonstrate the performance of our proposed algorithm by comparing it with two recent maximal algorithms, "OCB" and "MAXColoc", on a 3.3-GHz Centrino PC machine with 4G main memory. All the programs were coded and compiled using MATLAB. The Tree package utilized in our experiments was provided by Tinevez J. Y. and can be down-loaded from http://tinevez.github.io/matlab-tree/index.html.

4.1. Experimental data

We used both synthetic and real datasets for experiments. The two types of datasets are shown in Fig. 7 and detailed below.

Synthetic dataset: Figs. 7(a) and (b) are two synthetic datasets produced by a synthetic data generator (Huang et al., 2004). The two datasets have different data sizes and the same distribution area of 200∗280, so their densities are different. Specifically, Fig. 7(a) shows a sparse dataset with 500 points and 8 feature types, and Fig. 7(b) shows a dense dataset that has 5000 points with 8 feature types.

Real dataset: The real dataset used in our experiments is the recent (2014) facility point data from Beijing, China, that contains no sensitive location information. It was produced by the Beijing Administration for Industry and Commerce, and contains 288,486 items with 85 feature types such as bank, food store, and restaurant. Each item has four essential fields (instance ID, type ID, and X and Y coordinates). Fig. 7(c) and (d) shows the distribution of the facility points and the point densities of sixteen datasets based on administrative boundaries, respectively. It can be seen that districts 1–6 are located near the city center and have facility densities much higher than those in districts 7–16. Consequently, we separated cases 1–6 from the others, treating them as members of a dense group; the remaining cases fall naturally into a sparse group. Our experiments were organized mainly based on this division. Table 2 lists the primary statistical information of these datasets.

Specifically, we pre-process each dataset using the "Point Distance" tool of ArcGIS 10.0 to find neighbor instance pairs before implementing the three algorithms. Here, we focus primarily on the performance of the procedures that relate to our improved strategies. Thus, all three algorithms require equal computation times to find neighbor pair instances on the same dataset.

4.2. Computing speed comparisons

In this section, we measure the computational speed of our algorithm using an equidifferent series of prevalence thresholds and distance thresholds on different datasets and compare the results with those obtained using the other two contrasting algorithms.

4.2.1. Performance on synthetic datasets with changes of distance and prevalence thresholds

The execution times of the SGCT, OCB and MAXColoc algorithms over the sparse and dense datasets are graphed in Fig. 8 when the distance threshold was set to 9 as the prevalence threshold increased from 0.1 to 0.3; the results of the three algorithms over the two synthetic datasets, when the prevalence threshold was set to 0.1, as the distance thresholds increases from 5 to 9, are graphed in Fig. 9.

It can be seen that when the dataset prevalence threshold is high or the distance threshold is low, the execution times of the three algorithms on the same dataset are similar. However, with increasing distance threshold or decreasing prevalence threshold the gap between the three algorithms becomes large. The SGCT algorithm is much less sensitive to the prevalence and distance threshold changes than the other two algorithms. The MAXColoc algorithm is somewhat less sensitive, and the OCB algorithm is the most sensitive. The reason is that a lower distance threshold or a higher prevalence threshold causes more instance connections or candidate patterns. Both the OCB and MAXColoc algorithms use FP-growth-like approaches to construct candidate co-locations or instance cliques. Those approaches require many advanced tree-based operations such as grafting or chopping branches, and cause computing redundancies. On the contrary, the SGCT algorithm can reduce most of these redundancies using two proposed methods.

4.2.2. Performance on real datasets with changes of the prevalence threshold

In experiments with real datasets, the distance threshold was set to 300 m and 500 m for the dense group (cases 1–6) and sparse group (cases 7–16), respectively. The total execution times for the sixteen datasets are presented in the linear graphs in Fig. 10. Each graph presents the relationship between the prevalence threshold and entire execution time.

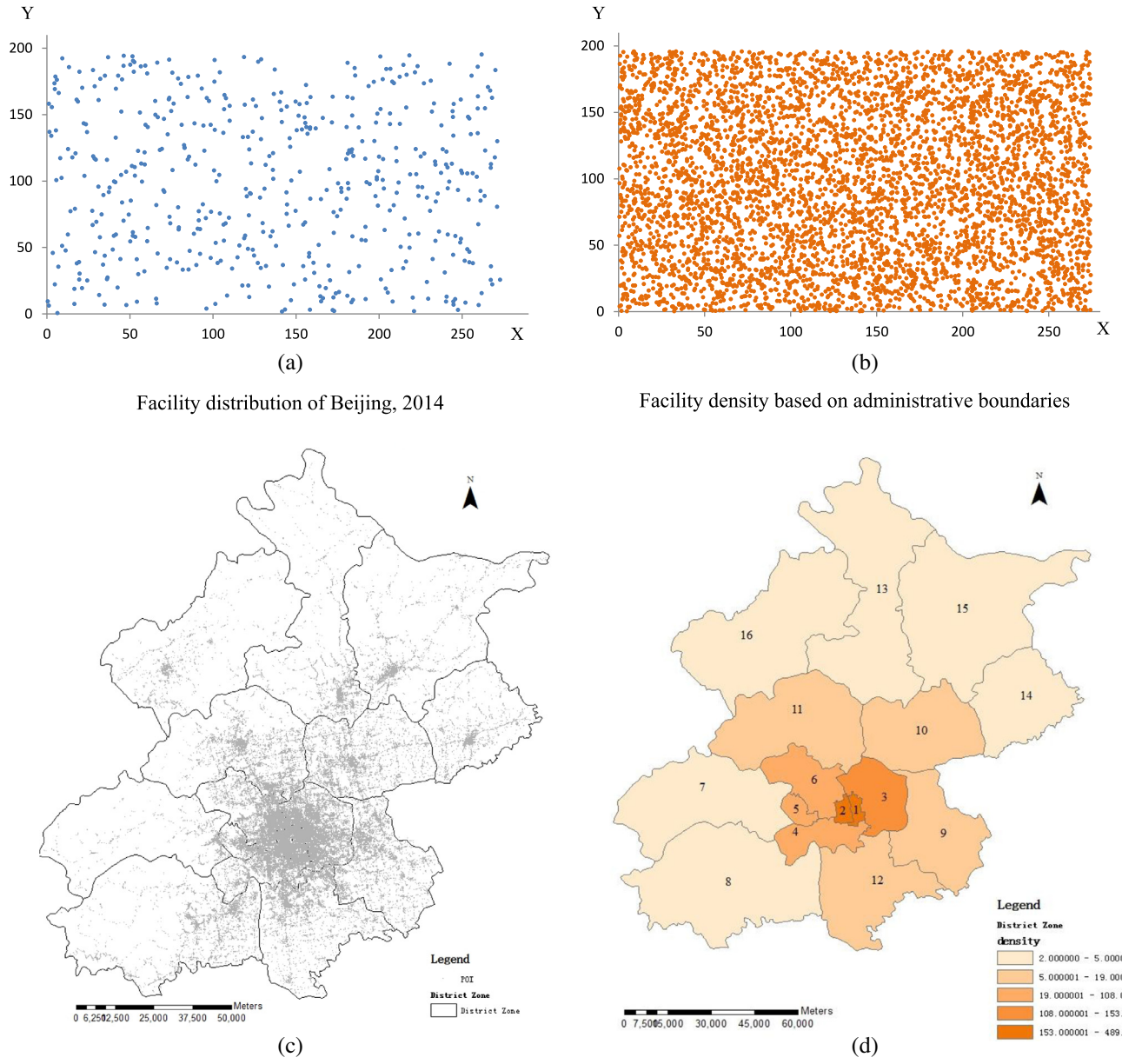

Fig. 7. Experimental data: (a) and (b) are sparse and dense synthetic datasets, (c) is the facility distribution of Beijing, 2014, and (d) is the facility density based on administrative boundaries.

Table 2
Statistical information of the real datasets.

Data group	Case ID	Name of district zone	Point number	Point density (number/km^2)	Type number
Dense group	1	Dong Cheng and Chong Wen	20813	489	75
	2	Xi Cheng and Xuan Wu	23176	483	79
	3	Chao Yang	65794	153	85
	4	Feng Tai	27130	91	83
	5	Shi Jing Shan	6135	68	80
	6	Hai Dian	44644	108	84
Sparse group	7	Men Tou Gou	2960	2	76
	8	Fang Shan	10083	5	80
	9	Tong Zhou	16329	19	83
	10	Shun Yi	13926	14	83
	11	Chang Ping	17847	14	85
	12	Da Xin	16188	16	83
	13	Huai Rou	6733	3	81
	14	Ping Gu	4857	5	76
	15	Mi Yun	7506	4	79
	16	Yan Qing	4365	2	79

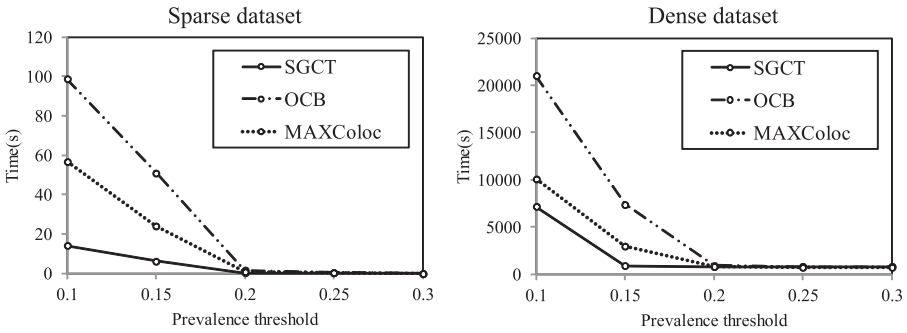

Fig. 8. The execution times over synthetic datasets with changes of the prevalence threshold.

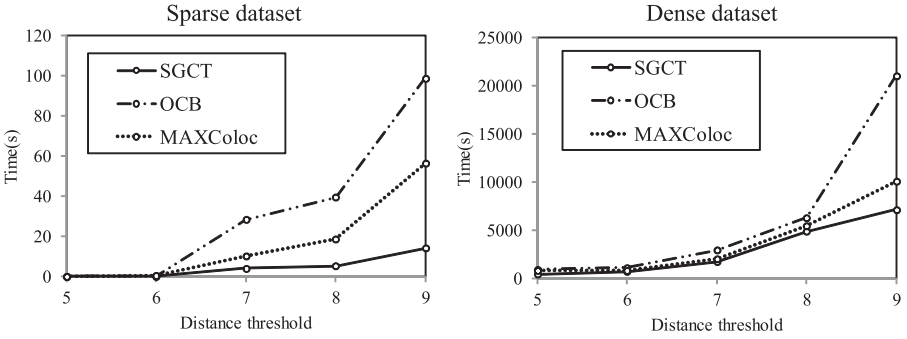

Fig. 9. The execution times over synthetic datasets with changes of the distance threshold.

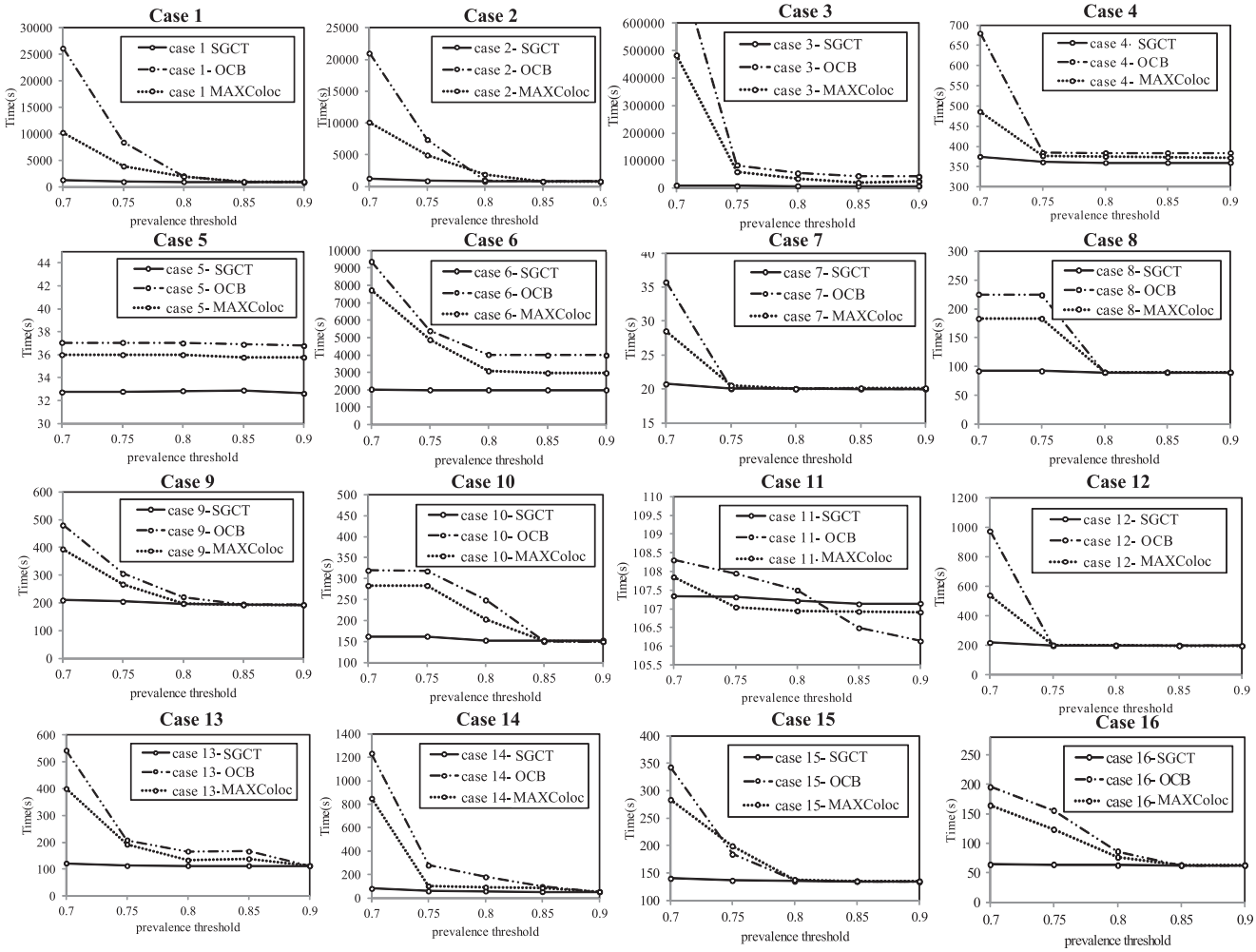

Fig. 10. The execution times with changes of the prevalence thresholds over real datasets. Cases 1–6 belong to the dense group (distance threshold of 300 m), and cases 7–16 belong to the sparse group (distance threshold of 500 m).

Fig. 10 shows that the SGCT algorithm has the lowest execution time in most cases. The results are similar to those over the synthetic datasets.

To measure the performances of the two improved strategies of the SGCT algorithm, we separately selected two representatives with different orders of magnitude from both the dense and sparse groups. These representatives are cases 1, 5, 12, and 14, representing large data with high density, small data with high density, large data with low density and small data with low density, respectively. The execution times and results of the two improved steps are charted in Fig. 11. The histograms on the left side show the execution times to obtain candidates from the size-2 instance table, and the histograms on the right-hand side show the execution time for the prevalent maximal co-locations from candidates. Moreover, the series on the right-hand side also includes line charts on the secondary axis related to the candidate and prevalent maximal co-location number changes.

Fig. 11 shows that first, for the SGCT and OCB algorithms, the execution time required for the first step is much less than that required for the second step, given the same distance and prevalence thresholds. In contrast, the MAXColoc algorithm exhibits the opposite characteristics. The reason is that the former two algorithms adopt a type-first strategy and therefore acquire candidate maximal co-locations directly from prevalent size-2 co-locations, whereas the latter uses an instance-first strategy by first constructing star candidate sets from type neighborhood transactions that map to the neighbor relationships of all co-location instances, and then using a candidate pruning method to obtain the candidate maximal co-locations. Generally, the instance number is extremely large compared to the type number, so the time complexity of the instance-first strategy will be less satisfactory than that of the type-first strategy in obtaining candidate maximal co-locations. However, searching for instance cliques that match a particular candidate maximal co-location is much simpler in the instance-first strategy because after the global instance-connecting operation, the instance-first strategy needs to find only satisfactory items from a lookup table.

Second, Fig. 11 shows that the time cost required for the second step increases as the gap between the candidate number and final result number increases. However, the execution time of the SGCT algorithm exhibits a lower sensitivity to this gap change than the other two methods because a larger gap signifies more long-size candidates are obtained. Due to the condensed tree-based tactic of obtaining instance cliques of the candidate maximal co-locations in the SGCT algorithm, we can directly obtain long-size instance cliques of these candidates within the instance pairs of the candidate co-location types to avoid more instance connections.

4.2.3. Performance with the change of the pattern number on real datasets

To determine the effects of the pattern number on performance, we extracted the total time expense of the three algorithms and the number of maximal patterns found at the prevalence threshold set in experiments (2), and we then constructed Fig. 12.

As shown in Fig. 12, with the increase of the number of patterns, the time costs of the three algorithms increased. However, the performance of the SGCT algorithm was still more impressive than the other two algorithms in all cases. That was because the larger number of patterns meant more iterations were required for long-size candidate co-location. In this case, the SGCT algorithm can cut down more redundant operations than the other two algorithms due to the sparse graph and condensed tree-based tactics.

4.2.4. Performance with different data sizes and densities on real datasets

To more deeply investigate the effects of data size and density on the total execution time of the three algorithms, we produced separate charts for the dense and sparse datasets using a prevalence threshold of 0.7, as shown in Fig. 13. The upper two graphs show the relationship between data size and entire execution time for dense and sparse groups separately, whereas the lower two graphs show the relationship between data density and entire execution time for these two groups. The time values on the vertical axis are section results with the same prevalence threshold of 0.7.

Fig. 13 demonstrates that the total execution time required by the three algorithms had a weak positive relationship with data size and density. However, the SGCT algorithm continued to have a lower sensitivity to these two factors than the other two algorithms. A dense or large dataset may easily cause a large number of instance connections. According to the previous analysis, it can be concluded that the SGCT algorithm tends to perform well under extreme conditions.

4.3. Storage comparisons

The storage costs of the maximal co-location algorithms depend on the long-size instance cliques and size-2 instance table. In our experiments, all tested algorithms were pre-processed; therefore, their size-2 instance tables on the same dataset cost the same storage space. Naturally, the difference of the space cost between the tested algorithms is mainly reflected by the instance cliques. The maximum number of instance connections directly affects the storage need of obtaining instance cliques. This value is actually the instance connection number of the neighborhood transaction in the MAXColoc algorithm, the neighbor relationship tree in the OCB algorithm, and the largest condensed instance tree in the SGCT algorithm.

In this section, except for two synthetic datasets, we selected two real datasets, i.e., cases 5 and 12, to evaluate the improvements in storage requirements for our algorithm. The synthetic datasets were implemented using a prevalence threshold of 0.1, and a distance threshold from 5 to 9. The real datasets used the prevalence from 0.7 to 0.9, and the distance threshold of 500 m. Figs. 14 and 15 show the maximum number of instance connections in different thresholds on synthetic datasets and real datasets, respectively.

Figs. 14 and 15 clearly show two phenomena. First, the maximum numbers of instance connections of the SGCT algorithm are considerably lower than those of the other two algorithms in most cases. The OCB algorithm has the second-fewer number, and the MAXColoc algorithm has the most instance connections. The reason is that the MAXColoc algorithm constructs neighborhood transactions and takes instance connections in the global instance space. The OCB algorithm constructs a neighbor relationship tree among the instances with types of the candidate co-location patterns. Even the latter method shrinks the scope of connecting instances compared with the former; the neighbor relationship tree is still a redundant initial structure because most tree nodes will be removed during the instance clique acquirement. On the contrary, the SGCT algorithm gains instance cliques for only an isolated candidate pattern during each iteration, releasing the space when the iteration completes. Furthermore, the hierarchical verification method avoids the initial duplicate construction process, so the storage need is the most satisfactory.

The second phenomenon in Figs. 14 and 15 is that as the distance threshold increases or the prevalence threshold decreases, the maximum numbers of instance connections of all three algorithms increases. In addition, this value tends to become larger

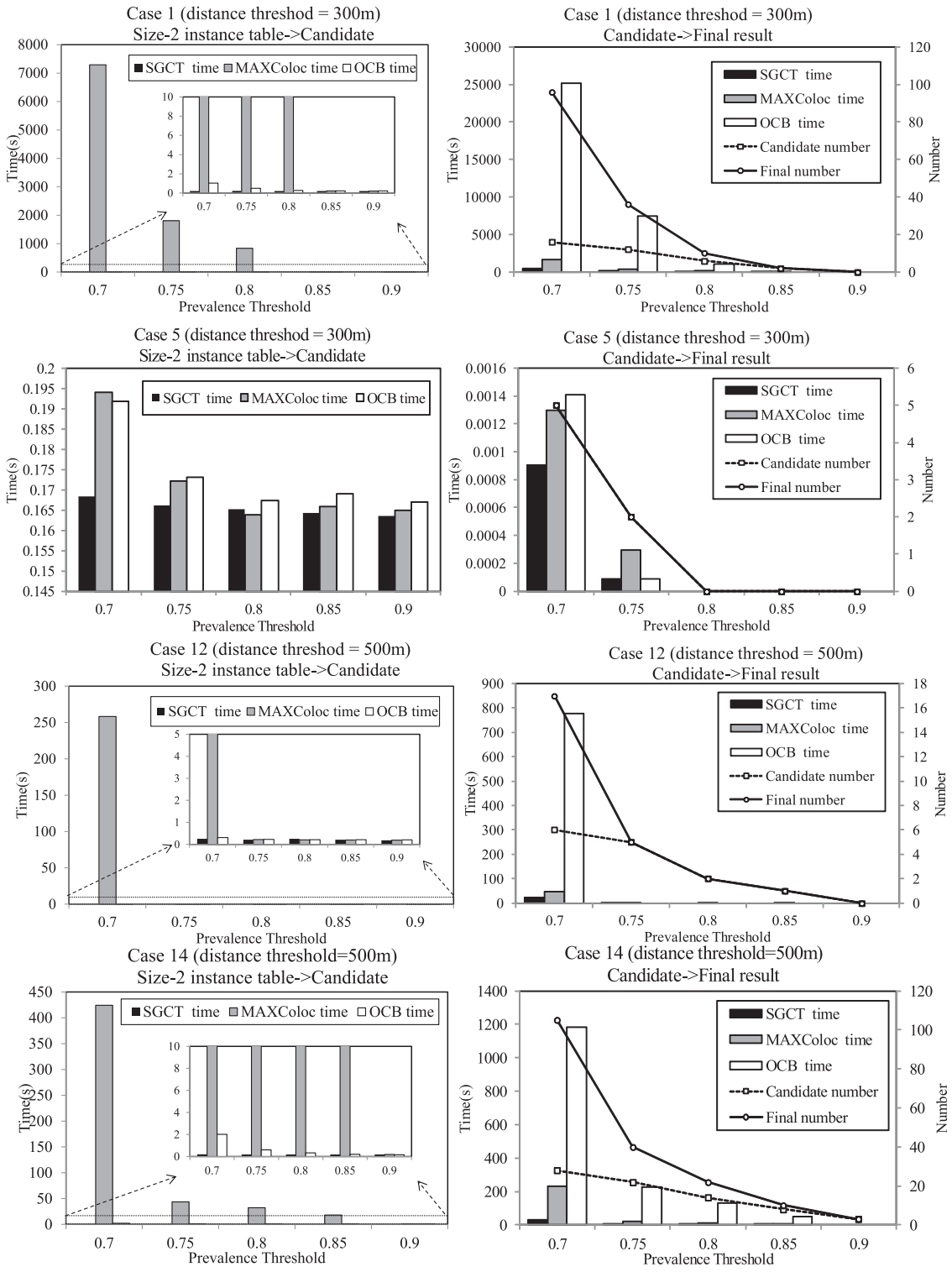

Fig. 11. Execution time and resulting size changes of two steps on four representative datasets.

for larger and dense datasets than for smaller and sparse datasets. In Section 4.2, we explained why the higher distance threshold, lower prevalence threshold, and larger and dense dataset may easily cause more instance connections, which means more redundancy operations will be implemented in the OCB or MAXColoc algorithms. Thus, in these extreme situations, the advantage of the SGCT algorithm can impressively stand out.

4.4. Result and analysis

This section provides some interesting co-location patterns mined by our algorithm. We chose two real datasets, i.e., cases 2 and 6, which are two concentrated industrial regions in Beijing. The prevalent maximal patterns of the two cases using different thresholds are listed in Table 3.

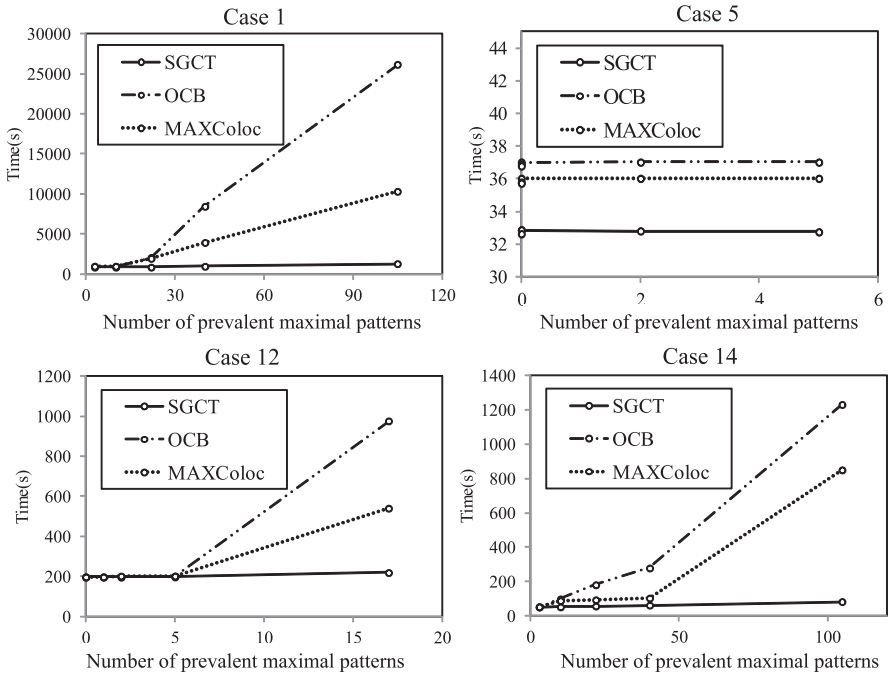

Fig. 12. Execution times with the changes of the prevalent pattern number on four representative datasets.

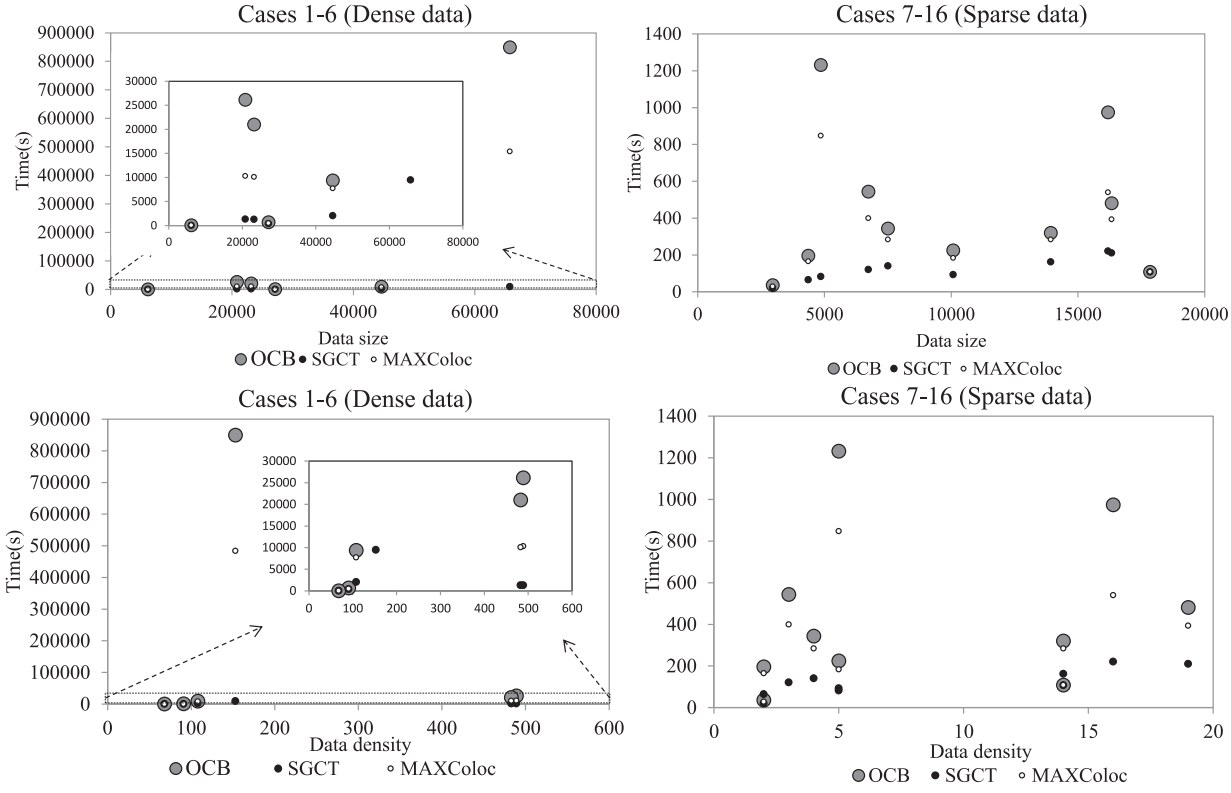

Fig. 13. The effects of data size and data density on execution time for the three tested algorithms.

From Table 3, we can see that in the results of the two cases over two threshold settings, the same patterns, as well as some different patterns, exist. For example, {restaurant, food store} and {restaurant, snack bar} are both prevalent patterns in the four situations; {hotel, leisure club} is prevalent only when the distance threshold is set to 100 m and the prevalence threshold is set to 0.3. In addition, cases 2 and 3 have more prevalent patterns relating to banks and private enterprises, respectively. That is because the two cases are the largest commercial and new high-technology centers of Beijing. This phenomenon implies that the prevalent patterns are much affected by the regional characters of datasets.

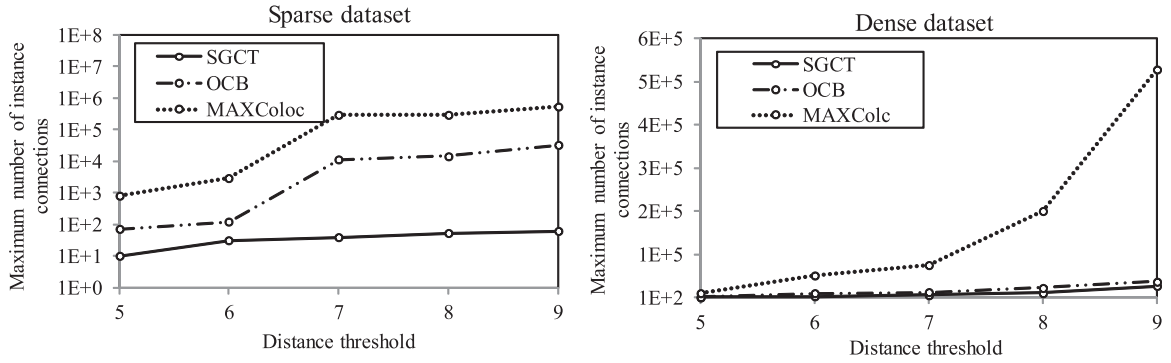

Fig. 14. The maximum number of instance connections with changes of the distance threshold for two synthetic datasets.

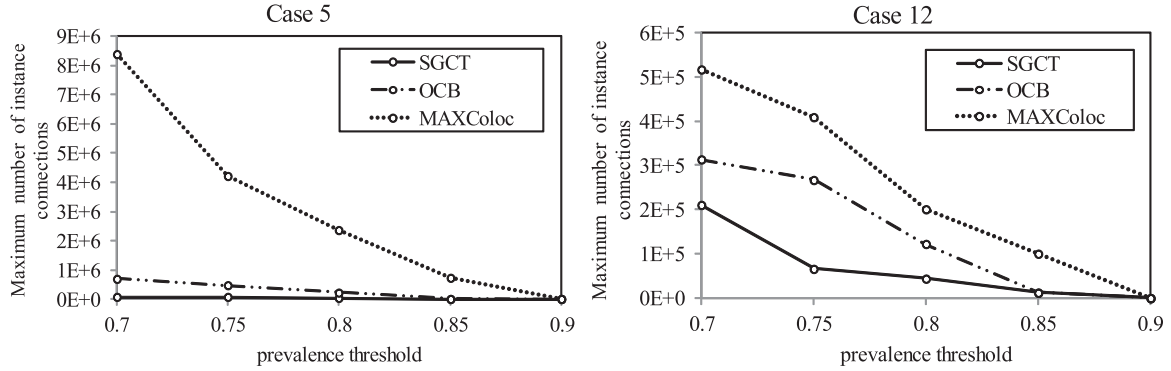

Fig. 15. The maximum number of instance connections with changes of the prevalence threshold for two real datasets.

Table 3
The prevalent maximal patterns of cases 2 and 6.

Threshold setting	Case 2	Case 6
Distance threshold = 200 m, prevalence threshold = 0.5	{Restaurant, food store}, {food store, bank}, {parking lot, bank}, {private enterprise, parking lot}, {bank, private enterprise, restaurant}, {commercial building, bank}, {private enterprise, commercial building}, {barber shop, food store}, {restaurant, snack bar}, {food store, snack bar}	{Restaurant, food store}, {restaurant, bank, private enterprise}, {restaurant, snack bar}, {parking lot, private enterprise, bank}, {private enterprise, commercial building}, {barber shop, food store}
Distance threshold = 100 m, prevalence threshold = 0.3	{Restaurant, food store}, {restaurant, snack bar}, {food store, snack bar}, {restaurant, bank}, {barber shop, food store}, {barber shop, restaurant}, {hotel, leisure club}, {bank, private enterprise}, {private enterprise, government sectors}, {optician shop, clothing shop}, {optician shop, cosmetics store}, {photo shop, clothing shop}, {cosmetics store, clothing shop}	{Restaurant, food store}, {restaurant, snack bar}, {restaurant, bank}, {parking lot, private enterprise}, {private enterprise, commercial building}, {snack bar, beverage outlets}, {leisure club, hotel}, {barber shop, food store}, {barber shop, snack bar}

Algorithm 1. Candidate maximal co-location generation (CMCG).

Input: $G=(E,V)$
Output: CP_m
1. $CP_m \leftarrow \emptyset$; $X \leftarrow \emptyset$; $P \leftarrow \emptyset$;
2. **for each** v_i^* in a degeneracy ordering $v_1^*, v_2^*, \ldots v_\lambda^*$;
3. $\{P \leftarrow N(v_i^*) \cap \{v_{i+1}^*, \ldots, v_\lambda^*\}$;
4. $X \leftarrow N(v_i^*) \cap \{v_1^*, \ldots, v_{i-1}^*\}$;
5. BK_Pivot(P, $\{v_i^*\}$,X);
6. }

7.**Procedure** BK_Pivot(M,K,T)
8.**if** $M \cup T = \emptyset$ **then**
9. $\{CP_m \leftarrow CP_m \cup K;\}$
10. Choose a pivot $u \in M \cup T$; % to maximize $|M \cap N(u)|$
11. **for each** $v_i \in M \backslash N(u)$
12. {BK_Pivot($M \cap N(v_i), K \cup \{v_i\}, T \cap N(v_i)$);
13. $M \leftarrow M \backslash \{v_i\}$;
14. $T \leftarrow T \cup \{v_i\}$;
15. }

5. Conclusions

This paper proposes a fast and space-saving algorithm (SGCT) for mining maximal co-locations. The prevalent size-2 co-locations are abstracted as a sparse undirected graph, and the degeneracy and pivot strategies in graph theory are used to improve the computing speed of obtaining candidate maximal co-locations. Furthermore, we devise a condensed-tree structure to store instance cliques of long candidate locations. This constructing process greatly diminishes the computational complexity and storage needs of the algorithm. The experimental results show that the SGCT algorithm is more efficient and requires less space than the current competitive maximal co-location algorithms. Moreover, because the iteration processes for candidate co-locations are unrelated to each other, our algorithm can be easily parallel transformed, which can further improve the computing speed.

However, the SGCT algorithm could be improved. For example, the processes for obtaining instance cliques for each candidate

Algorithm 2. Condensed instance tree construction (CITC).

Input: C_m, $InsTable_2$
Output: $CInsTree$ is the condensed instance tree of C_m

1. $i \leftarrow 1$; $CInsTree \leftarrow \emptyset$; Create a root′ CIT′ for $CInsTree$;
2. **while** $i < size(C_m)$
3. {**if** $i = 1$
4. {**for each** instance pair$InsPair_k \in InsTable_2(C_m(1), C_m(2))$
5. {**if** $InsPair(1) \in CInsTree_0.children$ **then**
6. {Add a child − node $InsPair_k(2)$ to $CInsTree_1(InsPair_k(1))$; }
7. **else**
8. {Create a subtree with $InsPair_k(1)$ as the root and $InsPair_k(2)$ as the child of its root;
9. Graft this subtree to the root of $CInsTree$; }
10. }
11. }
12. **else**
13. {**for each** instance node $ins_k \in CInsTree_i$
14. {Find the indices of items that are equal to ins_k from the first column of $InsTable_2(C_m(i), C_m(i+1))$;
15. Store the second items of the corresponding instance pair in list EI;
16. **for each** $ei_t \in EI$
17. {$flag \leftarrow i - 1$;
18. $currIns \leftarrow ins_k.parent$;
19. **while** $flag \geq 1$
20. {**if**$(currIns, ei_t) \in InsTable_2(C_m(flag), C_m(i+1))$ **then**
21. {$currIns \leftarrow currIns.parent$; }
22. **else break**;
23. $flag \leftarrow flag - 1$;
24. }
25. **if** $flag = 0$ **then** {Add a child − node ei_t to $CInsTree_i(ins_k)$; }
26. }
27. }
28. }
29. $i \leftarrow i + 1$;
30. }

co-locations are independent. When the candidates have an excessive number of communal types, the instance-connecting operations of these communal types will cause redundant computations. In addition, although the degeneracies of the size-2 co-location graphs related to our tested spatial datasets are low, we cannot confirm that this is a universal phenomenon. If the degeneracy is high, the sparse graph strategy provides little improvement over the entire algorithm. Our future studies will focus on the above two points.

Acknowledgements

We are grateful to the editor and the anonymous referees for their valuable comments and suggestions

Funding: This work was supported by the Special Foundation of the Chief of the Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences (Y6SJ2800CX) and the National Science-technology Support Plan Projects of China (2015BAJ02B00).

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), *Proceedings of the 20th international conference on very large databases* (pp. 487–499). Burlington, MA: Morgan Kaufmann Publishers.

Arunasalam, B., Chawla, S., Sun, P., & Munro, R. (2004). Mining complex relationships in the SDSS SkyServer spatial database. In *Proceedings of the 28th annual international computer software and applications conference (COMPSAC 2004): 2* (pp. 142–145). Washington, D.C.: IEEE.

Akbari, M., Samadzadegan, F., & Weibel, R. (2015). A generic regional spatio-temporal co-occurrence pattern mining model: A case study for air pollution. *Journal of Geographical Systems, 17*, 249–274. doi:10.1007/s10109-015-0216-4.

Al-Naymat, G. (2013). GCG: Mining maximal complete graph patterns from large spatial data. In *2013 ACS international conference on computer systems and applications* (pp. 1–8). Washington, D.C.: IEEE.

Bron, C., & Kerbosch, J. (1973). Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM, 16*, 575–577. doi:10.1145/362342.362367.

Boinski, P., & Zakrzewicz, M. (2014). Algorithms for spatial collocation pattern mining in a limited memory environment: A summary of results. *Journal of Intelligent Information Systems, 43*, 147–182.

Cazals, F., & Karande, C. (2008). A note on the problem of reporting maximal cliques. *Theoretical Computer Science, 407*, 564–568. doi:10.1016/j.tcs.2008.05.010.

Eppstein, D., Löffler, M., & Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. In O. Cheong, K. Y. Chwa, & K. Park (Eds.), *21st international symposium on algorithms and computation* (pp. 403–414). Berlin, Germany: Springer-Verlag.

Eppstein, D., & Strash, D. (2011). Listing all maximal cliques in large sparse real-world graphs. In P. Pardalos, & S. Rebennack (Eds.). In *Lecture notes in computer science: 6630* (pp. 364–375). Berlin, Germany: Springer-Verlag.

Fan, B., & Luo, J. (2013). Spatially enabled emergency event analysis using a multi-level association rule mining method. *Natural Hazards, 67*, 239–260. doi:10.1007/s11069-013-0556-7.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In M. Dunham, J. F. Naughton, W. Chen, & N. Koudas (Eds.), *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 1–12). New York, NY: ACM.

Huang, Y., Shekhar, S., & Xiong, H. (2004). Discovering colocation patterns from spatial datasets: a general approach. *IEEE Transactions on Knowledge and Data Engineering, 12*, 1472–1485.

Huang, Y., Zhang, P., & Zhang, C. (2008). On the relationships between clustering and spatial co-location pattern mining. *International Journal on Artificial Intelligence Tools, 17*, 55–70. doi:10.1142/S0218213008003777.

Johnston, H. C. (1976). Cliques of a graph—Variations on the Bron–Kerbosch algorithm. *International Journal of Parallel Programming, 5*, 209–238.

Kim, S. K., Kim, Y., & Kim, U. (2011). Maximal cliques generating algorithm for spatial co-location pattern mining. In C. Lee, J. M. Seigneur, J. J. J. H. Park, & R. R. Wagner (Eds.), *Secure and trust computing, data management, and applications* (pp. 241–250). Berlin, Germany: Springer-Verlag.

Lee, I., Qu, Y., & Lee, K. (2012). Mining qualitative patterns in spatial cluster analysis. *Expert Systems with Applications, 39*, 1753–1762. doi:10.1016/j.eswa.2011.08.079.

Leibovici, D. G., Claramunt, C., Le Guyader, D., & Brosset, D. (2014). Local and global spatio-temporal entropy indices based on distance-ratios and co-occurrences distributions. *International Journal of Geographical Information Science, 28*, 1061–1084. doi:10.1080/13658816.2013.871284.

Makino, K., & Uno, T. (2004). New algorithms for enumerating all maximal cliques. In T. Hagerup, & J. Katajainen (Eds.), *Algorithm theory-SWAT 2004* (pp. 260–272). Berlin, Germany: Springer-Verlag.

Shekhar, S., & Huang, Y. (2001). Discovering spatial co-location patterns: A summary of results. In C. S. Jensen, M. Schneider, B. Seeger, & V. J. Tsotras (Eds.), *Proceedings of the 7th international symposium on advances in spatial and temporal databases* (pp. 236–256). Berlin, Germany: Springer-Verlag.

Sierra, R., & Stephens, C. R. (2012). Exploratory analysis of the interrelations between co-located Boolean spatial features using network graphs. *International Journal of Geographical Information Science, 26*, 441–468. doi:10.1080/13658816.2011.594799.

Tomita, E., Tanaka, A., & Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science, 363*, 28–42. doi:10.1016/j.tcs.2006.06.015.

Wang, L., Zhou, L., Lu, J., & Yip, J. (2009). An order-clique-based approach for mining maximal co-locations. *Information Sciences, 179*, 3370–3382. doi:10.1016/j.ins.2009.05.023.

Yao, H. C., Wang, L., Chen, H., & Zou, M. (2015). Spatial co-location patterns mining algorithm over massive spatial data sets. *Journal of Frontiers of Computer Science and Technology, 9*, 24–35.

Yoo, J. S., & Boulware, D. (2013). A framework of spatial co-location mining on MapReduce. In *IEEE international conference on big data* (p. 44). Washington, D.C.: IEEE. doi:10.1109/BigData.2013.6691797.

Yoo, J. S., Boulware, D., & Kimmey, D. (2014). A parallel spatial co-location mining algorithm based on MapReduce. In *IEEE international congress on big data* (pp. 25–31). Washington, D.C.: IEEE. doi:10.1109/BigData.Congress.2014.14.

Yoo, J. S., & Bow, M. (2011). Mining maximal co-located event sets. In J. Z. Huang, L. Cao, & J. Srivastava (Eds.), *Advances in knowledge discovery and data mining* (pp. 351–362). Berlin, Germany: Springer-Verlag.

Yoo, J. S., & Shekhar, S. (2004). A partial join approach for mining co-location patterns. In I. F. Cruz, & D. Pfoser (Eds.), *Proceedings of the 12th ACM international symposium on advances in geographic information systems* (pp. 241–249). New York, NY: ACM.

Yoo, J. S., & Shekhar, S. (2006). A joinless approach for mining spatial colocation patterns. *IEEE Transactions on Knowledge and Data Engineering, 18*, 1323–1337. doi:10.1109/TKDE.2006.150.

Yoo, J. S., Shekhar, S., Kim, S., & Celik, M. (2006). Discovery of co-evolving spatial event sets. In J. Ghosh, D. Lambert, D. Skillcorn, & J. Srivastava (Eds.), *Proceedings of the of the 6th SIAM international conference on data mining* (pp. 306–315). Philadelphia, PA: SIAM.

Yu, W. (2016). Spatial co-location pattern mining for location-based services in road networks. *Expert Systems with Applications, 46*, 324–335. doi:10.1016/j.eswa.2015.10.010.