Concurrent Execution of Data Mining Queries for Spatial Collocation Pattern Discovery[*]

Pawel Boinski and Maciej Zakrzewicz

Poznan University of Technology, Institute of Computing Science
{pawel.boinski,maciej.zakrzewicz}@cs.put.poznan.pl

Abstract. In spatial databases, Collocation Pattern Discovery is a very important data mining technique. It consists in searching for types of spatial objects that are frequently located together. Due to high requirements for CPU, memory or storage space, such data mining queries are often executed at times of low user activity. Multiple users or even the same user experimenting with different parameters can define many queries during the working hours that are executed, e.g., at off-peak night-time hours. Given a set of multiple spatial data mining queries, a data mining system may take advantage of potential overlapping of the queried datasets. In this paper we present a new method for concurrent processing of multiple spatial collocation pattern discovery queries. The aim of our new algorithm is to improve processing times by reducing the number of searches for neighboring objects, which is a crucial step for the identification of collocation patterns.

1 Introduction

Most of the spatial datasets consist of instances that are described by *spatial features* which can be interpreted as a characteristic of space in a particular location. Typical examples of spatial features include species, business types or points of interest (e.g., hospitals, airports). Shekhar and Huang introduced an important concept of *spatial collocation patterns* [11]. The definition of a spatial collocation pattern (or in short a *collocation*) assumes that it is a subset of spatial features whose instances (e.g., particular airport and custom office) are frequently located together in a spatial neighborhood. Such patterns are the product of the data mining which is one of the most important steps in the Knowledge Discovery in Databases - a non-trivial process of discovering valid, novel, potentially useful and ultimately understandable patterns in the data [7].

For end users *a data mining system* can be regarded as an advanced database with sophisticated querying methods. Users define data mining queries, i.e., a classes of interesting patterns, sets of criteria and input datasets. The task of the data mining system is to choose and execute an appropriate algorithm and finally return discovered patterns to the users. In regular databases the time required

[*] This paper was funded by the Polish National Science Center (NCN), grant No. 2011/01/B/ST6/05169.

to execute user commands is usually very short (except some administrative tasks), while in data mining systems, the time to answer a single query can be expressed in minutes or even hours. Therefore, in a real-life scenario, data mining queries are collected during the user's working hours and executed at nights, when the system activity is low. It is very likely that some of the data mining queries are related in such a way, that they share some input data. This relation can be utilized to process them all at once in such a way that the total processing time will be reduced in comparison with the straightforward serial execution (the low activity time slot is limited to 6-8 hours). Another possibility is to execute multiple queries in such order that consecutive queries can take advantage of already computed and stored results of previous queries. Finally, the spatial data mining system could execute queries that occur at random times by incorporating them into currently ongoing data mining process.

In this paper we propose a new algorithm for processing batches of spatial data mining queries for collocation patterns discovery. We introduce a concurrent collocation candidate generation method and an extended iCPI-tree structure that stores materialized neighbor relationships for multiple queries. Conducted experiments have confirmed the high efficiency of the proposed algorithm.

2 Motivation and Related Work

2.1 Motivation

Consider a database of spatial objects describing various facilities in a particular city. One can be interested in collocation patterns involving cinemas, food stores, tram stops and schools whereas another user may want to find collocations of such features as theaters, cinemas, opera houses and tram stops. The first user wants to analyze districts d_1 and d_2 of the city and is interested in patterns with at least 40% prevalence, while the second user wants to analyze districts d_2 and d_3 with minimum prevalence of 25%. The neighbor relation in both cases is Euclidean distance less than 100 m. In general, we assume that the neighbor relation is consistent across the processed queries, however it can vary depending on the location (e.g., different for cities and rural areas) in the analyzed space.

The most trivial approach is to execute each query separately. We will refer to this strategy as a *sequential processing*. The sequential processing is easy to implement although it cannot benefit from the input data shared by multiple queries. In the considered example, instances of features 'cinema' and 'tram stop' located in the district d_2 are shared among two queries. By merging the execution of these queries we can reduce the number of searches across space required to identify instances of certain candidates.

2.2 Related Work

One can notice that collocation discovery problem is substantially similar to the frequent itemset discovery problem presented in [1], however, direct application of well-known association mining algorithms, e.g., *Apriori* [2], is very challenging. Difficulties arise from significant differences in the characteristics between classical market-basket data and spatial data. For example, instances in the market-basket analysis are precisely nested in transactions, while instances of spatial features are embedded in a continuous space and share 'hidden' neighbor relationships. As a result, new methods for collocation mining have been developed. The most interesting ones are *Co-Location Miner* [11], *Joinless* [14] and *iCPI-tree* [12] accompanied with our work on efficient processing of spatial data mining queries in a limited memory environment [3,4,5].

The problem of efficient execution of multiple queries in classical databases has been extensively studied (e.g., [8,10]) and basically consists in a single execution of expressions shared by at least two queries. This general idea remains the same in the context of spatial data mining queries, however due to the more complex processing it cannot be directly transferred. The problem of batch processing of queries for association discovery has been introduced in [13]. The authors proposed two solutions, named *Mine Merge* and *Apriori Common Counting*, to reduce the total I/O and CPU cost of executing a set of data mining queries.

To the best of our knowledge there are no current works on batch processing of collocation pattern mining queries, although there are some works on indexing collocation patterns for future reuse [6] and on incremental maintenance of collocation patterns when a set of new spatial data arrives [9]. In the first approach, there is a computationally demanding step of pre-calculating and materializing all collocation instances. To compensate the time required to perform this task, a significant number of queries (counted in dozens or even hundreds) must be executed afterwards. In the second approach results of the query are materialized and updated in the response to changing input dataset. Contrary, we do not pre-calculate or materialize collocations. In the proposed method collocation queries are executed concurrently only for the required subset of the input data.

2.3 Basic Definitions

Definition 1. *Let f be a spatial feature. An object x is an* ***instance*** *of the feature f, if x is a type of f and is described by a location and unique identifier. Let F be a set of spatial features and S be a set of their instances. Given a neighbor relation R, we say that the* ***collocation*** *C is a subset of spatial features $C \subseteq F$ whose instances $I \subseteq S$ form a clique w.r.t. the relation R.*

Definition 2. *The* ***participation ratio*** *$Pr(C, f_i)$ of a feature f_i in the collocation $C = \{f_1, f_2, \ldots, f_k\}$ is a fraction of objects representing the feature f_i in the neighborhood of instances of collocation $C - \{f_i\}$. $Pr(C, f_i)$* is equal to the number of distinct objects of $f_i$in instances of C divided by the number of all objects of f_i. *The* ***participation index (prevalence measure)*** *$Pi(C)$ of a collocation $C = \{f_1, f_2, \ldots, f_k\}$ is defined as $Pi(C) = \min_{f_i \in C} \{Pr(C, f_i)\}$.*

Lemma 1. *The participation ratio and participation index are monotonically non-increasing with increases in the collocation size.*

Definition 3. *Given a subset of spatial instances $I = \{o_l, \ldots, o_v\}$, where $l, v \in \{1, 2, \ldots, m\}$, if $o_i \leq o_j$ holds for any $l \leq i \leq j \leq v$, the I is called as an* ***ordered instance set****. If the feature of o_i is not the same as the feature of o_l and $R(o_l, o_i)$ holds for any $l < i \leq v$, the I is called as* ***ordered neighbor relationship set of the instance*** *o_l. The set of ordered neighbor relationship sets of all instances of a spatial feature x is denoted as δ_x. Given a set of spatial features $F = \{f_1, f_2, \ldots, f_n\}$ and a set of ordered instance neighbor relationship of spatial these features $\delta = \delta_{f_1} \cup \delta_{f_2} \cup \ldots \cup \delta_{f_n}$, a tree designed as follows is called as an* ***improved Collocation Pattern Instances tree (iCPI-tree)****. The iCPI-tree consists of one root labeled as "null" and a set of the spatial features sub-trees as the children root. The spatial feature f_i sub-tree consists of the root f_i and each subset of δ_{f_i} as a branch of the root. Each branch records an ordered neighbor relationship set of corresponding instance and relevant feature.*

2.4 The iCPI-Tree Based Method

The general approach to collocation mining has been proposed in [11]. It consists of three major steps: (1) generating collocation candidates, (2) identifying instances for candidates and (3) filtering candidates w.r.t. to the minimum prevalence threshold. These steps are executed iteratively. In $k-$th iteration, size-k candidates are processed. The first step can be accomplished by applying well-known *Apriori* strategy [2] due to the anti-monotonicity property of the prevalence measure. The last step is very straightforward and basically consists in computing prevalence measure for each candidate. The most time consuming part of the algorithm is the second step. The first idea presumed that spatial join should be used to find co-located objects. In [14] a concept of materialized neighborhoods has been introduced. Wang et. al [12] extended this concept by defining a tree structure (called iCPI-tree) for fast identification of neighbors.

In the *iCPI-tree* each child of the root node is a subtree that contains neighbors for instances of a specific spatial feature. Sub-trees are composed of nodes representing spatial features of neighbors and leafs corresponding to neighbor instances. For example, in Fig. 2 (section 3.2) the tree $iCPI_1$ contains two subtrees for features A and B. Given the instance $A7$ we can easily find that it has one neighbor with B feature ($B6$) and two neighbors with C feature ($C5$ and $C8$). During the execution of the algorithm, new instances of candidates are constructed from instances of collocations from previous iteration. For example, to find instances of candidate ABC, an instance $A1, B2$ can be used. The procedure searches for neighbors with feature C of $A1$ and $B2$. If there are common neighbors for both elements, a new instance is constructed. Using the $iCPI_1$ tree, one can find that there is an instance $A1, B2, C3$. For details of the iCPI-tree based algorithm please consult the paper [12].

3 Batch Processing of Spatial Data Mining Queries

In this section we introduce preliminaries, motivations and our new algorithm for batch processing of spatial data mining queries.

3.1 Preliminaries

Definition 4. *A* ***spatial data mining query*** *SDMQ is a tuple (S, F, L, R, mp), where S is a spatial framework, F is a set of spatial features, L is a subset of spatial framework S, R is a neighbor relation and mp is a minimum prevalence. The result of the SDMQ is a set of collocation patterns discovered from instances of F w.r.t. to R located in L having the prevalence not less than mp.*

Definition 5. *A* ***set of spatial data mining queries*** *$QS = \{(SDMQ_1, t_1), (SDMQ_2, t_2), \ldots, (SDMQ_n, t_n)\}$ consists of pairs $(SDMQ_i, t_i)$, $1 \leq i \leq n$ where $SDMQ_i$ is a spatial data mining query and t_i is the time of the arrival of this query to the data mining system.*

In this work we focus on the execution of batches of data mining queries, i.e., sets of n spatial data mining queries where for each $1 \leq i, j \leq n$, $t_i = t_j$.

Definition 6. *A set $A_S = \{a_1, a_2, \ldots, a_m\}$ is a set of* ***distinct areas*** *of S, i.e., set of uniquely numbered subsets of spatial framework S such that for each $1 \leq i, j \leq m$, areas a_i and a_j do not overlap and all areas from A_s constitute a framework S.*

Definition 7. *A* ***shared collocation pattern*** *is a subset of spatial features with an additional list of SDMQs that it belongs to. A* ***shared collocation instance*** *is a set of instances of collocation features located together in a spatial neighborhood with assigned set of distinct areas $SA = \{a_k, \ldots, a_l\}$ such that for each $a_i \in SA$ at least one collocation feature instance is located in a_i.*

Definition 8. *Given a set of n data mining queries QS and a set $F = \{F_1 \cup F_2 \cup \ldots \cup F_n\}$, where F_i denotes a set of spatial features of $SDMQ_i \in QS$, a* ***Common iCPI-tree*** *is an enhanced iCPI-tree such that for each spatial feature $f_i \in F$, sub-tree consists of the root f_i and each subset of a set of ordered instance neighbor relationship sets of all instances of f_i. Each instance node of f_i is extended with the identifier of the distinct area that it belongs to.*

3.2 The Common iCPI-Tree Based Method

In this section we introduce our new algorithm called *Common iCPI-tree* for the concurrent execution of multiple queries in a batch. The pseudocode for this algorithm is shown in Alg. 1. Within the following paragraphs we will refer to this pseudocode by putting the corresponding line numbers in brackets. The general idea introduced in the iCPI-tree method remains the same although there are additional algorithm steps and extensions of structures required to perform effective execution of batched queries.

To explain how our method works, we will use an example dataset shown in Fig. 1. There are 4 features A, B, C and D with the total of 21 instances. The batch is composed of two queries: $SDMQ_1 = (Input, \{A, B, C\}, 1.5 < x \leq 13, d \leq 2, 0)$ and $SDMQ_2 = (Input, \{A, B, C, D\}, 6.5 < x \leq 19.5, d \leq 2, 0)$. For simplicity and better explanation of the algorithm both minimum prevalence thresholds are set to 0 and we use only x axis to specify query area of interest. A line connecting two objects represents a neighbor relationship (distance not grater than 2 units).

Algorithm 1. Common iCPI-tree based collocation mining algorithm

```
Input: $QS$ - a set of $n$ spatial data mining queries with the same $r$ neighbor relation
Output: a set of collocation patterns
Variables: $F_i$ - a set of $SDMQ_i$ spatial features, $A$ - a set of distinct areas for $QS$, $CiCPI_k$ - a Common iCPI-tree, $SC_k$ - a set of size-k shared collocation candidates, $SP_k$ − a set of size-k prevalent shared collocations, $SPI_k$ − a set of size-k shared clique instances
 1: procedure COMMON_ICPI($QS$)
 2:     $A$ = genDistinctAreas($QS$)
 3:     $CiCPI$ = genCommonTree ($F_1 \cup F_2 \ldots \cup F_n, A, r$); $k = 1$
 4:     $SP_k$ = genOneElementSharedCollocations($F_1 \cup F_2 \ldots \cup F_n, A$)
 5:     while ($SP_k \neq \emptyset$) do
 6:         $SC_{k+1}$ = AprioriGenSharedCandidates ($SP_k$)
 7:         for $sc \in SC_{k+1}$ do /* for each shared candidate */
 8:             for $sp_{inst} \in SPI_k$ with features equal to $sc$ prefix do
 9:                 if $sp_{inst}$ belongs only to areas in $sc$ then
10:                     $CN$ = searchCommonNeighboors ($sp_{inst}, sc, CiCPI$)
11:                     for $ne \in CN$ do
12:                         $sc_{newInst} = sp_{inst} \cup \{ne\}$, add $ne$ area to $sc_{newInst}$ areas
13:                         $SPI_{k+1} = SPI_{k+1} \cup \{sc_{newInst}\}$
14:                     end for
15:                 end if
16:             end for
17:         end for
18:         $SP_{k+1}$ = getPrevalent ($SC_{k+1}, SPI_{k+1}, QS$)
19:         $k = k + 1$
20:     end while
21:     return $\bigcup(SP_2, \ldots, SP_{k-1})$
22: end procedure
```

In the original iCPI-tree method the most computationally demanding part is the step of searching a tree structure to construct new collocation instances having the clique property. The sequential processing strategy requires to construct and process a separate iCPI-tree for each query from the batch set (two iCPI-trees for sample data are shown in Fig. 2). We propose to build only one tree that contains instances for all queries (line 3). We refer to this structure as *Common iCPI-tree* (CiCPI-tree). To distinguish instances among different queries an additional identifier has to be stored with each node representing an object instance in the tree. In our opinion, the best solution is to use properly constructed bitmaps for that purpose. A bitmap (also known as a bitset, bit array or bit vector) is a compact structure that stores an array of bits. It is extremely fast due to the hardware, low-level parallelism in processing whole words or bytes. To determine appropriate bitmaps for tree elements, first of all, space has to be divided into a set of distinct areas (line 2).

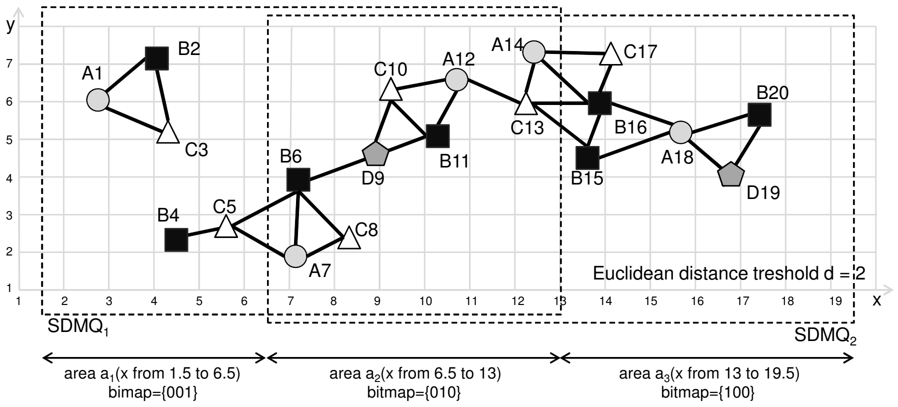


Fig. 1. Input datasets

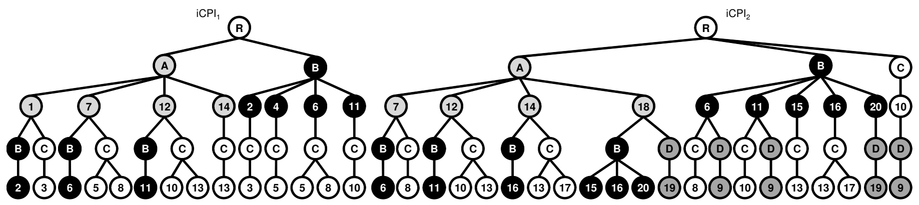


Fig. 2. Two iCPI-trees generated in sequential processing approach

In the sample dataset, three distinct areas can be distinguished: $a_1(1.5 < x \leq 6.5)$, $a_2(6.5 < x \leq 13)$ and $a_3(13 < x \leq 19.5)$. For each area, a unique bitmap is generated by setting the i-th element for the i-th area. Therefore we have identifiers $\{001\}$, $\{010\}$, $\{100\}$ for areas a_1, a_2 and a_3 respectively (notice: although three areas can be encoded on two bits, the mentioned solution is more efficient in the further processing). Given the set of distinct areas, the next step is to create the Common iCPI-tree. For each object o_i analyzed by at least one SDMQ all neighbors with features greater than o_i feature have to be found. To perform this task a plane sweep method or a spatial index can be utilized. Neighbors are ordered by their feature (e.g., using the lexical order) and their identifier. For each object a new bitmap is created. It must correspond to the bitmap of the area in which this object is located in. For example, given the object $A14\{010\}$ the final list of neighbors contains $B16\{100\}$, $C13\{010\}$ and $C17\{100\}$. The discovered neighborhoods (and their bitmaps) are inserted into the CiCPI-tree using the procedure described in [12]. Figure 3 presents final

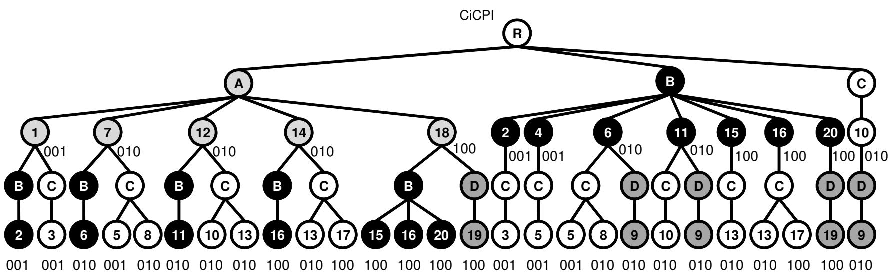
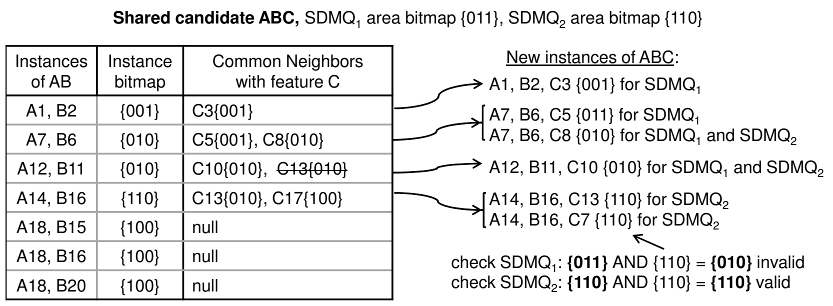


Fig. 3. Common iCPI-tree generated in concurrent processing approach

CiCPI-tree structure that will be used to scan for neighbor instances in the consecutive iterations.

In the k-th iteration (lines 5-20), the algorithm discovers prevalent size-k+1 collocations for each SDMQ by generating (shared) collocation candidates (line 6), identifying their instances (lines 8-16) and counting their prevalences (line 18). Due to the participation index definition, all size-1 collocations are prevalent (with prevalence = 100%). In the sequential processing approach, *AprioriGen* [2] method would be applied to each SDMQ separately. We propose to generate candidates simultaneously for all queries using *AprioriGenSharedCandidates* method (line 6). At the beginning, for each candidate collocation a bitmap is assigned. Each bitmap has length equal to the size of the batch and indicates queries which share this particular candidate. If such a candidate (or a collocation) is shared by the i-th SDMQ, the i-th bit is set. In our example there are 4 size-1 collocations: A, B, C and D. All except D are shared between $SDMQ_1$ and $SDMQ_2$ hence the following set of bitmaps is assigned: $\{11\}$, $\{11\}$, $\{11\}$, $\{10\}$. The general idea of AprioriGenSharedCandidates is similar to the original method. All pairs of size-k-1 collocations sharing at least one query are joined to get size-k candidates. Each generated candidate has a bitmap resulting from bitwise AND operation on all bitmaps from its size-k-1 subsets. Finally, a pruning step is applied to remove candidates that cannot be prevalent. In the introduced example, size-2 candidates are: $AB\{11\}$, $AC\{11\}$, $AD\{10\}$, $BC\{11\}$, $BD\{10\}$ and $CD\{10\}$, size-3: $ABC\{11\}$, $ABD\{10\}$, $ACD\{10\}$ and $BCD\{10\}$, size-4: $ABCD\{10\}$.

Starting with $k = 2$, instances for each size-k candidate are constructed by expanding instances of size-k-1 collocation discovered in the previous iteration (lines 8-16), however a sharing property of candidates and collocations must be taken into consideration. For $k = 2$ the process traverses the CiCPI-tree and for each instance of the first candidate feature, the neighbors with the second feature are retrieved from the tree. For example, given the candidate $AB\{11\}$, for the instance $A14$ there is one neighbor $B16$. Because the instance $A14, B16$ can be shared between two queries (the candidate $AB\{11\}$ is shared by $SDMQ_1$ and $SDMQ_2$), there is a necessity to store such information in the form of bitmap. It is a result of bitwise OR operation performed on bitmaps for individual objects.



Shared candidate ABC, SDMQ$_1$ area bitmap {011}, SDMQ$_2$ area bitmap {110}

Instances of AB	Instance bitmap	Common Neighbors with feature C
A1, B2	{001}	C3{001}
A7, B6	{010}	C5{001}, C8{010}
A12, B11	{010}	C10{010}, ~~C13{010}~~
A14, B16	{110}	C13{010}, C17{100}
A18, B15	{100}	null
A18, B16	{100}	null
A18, B20	{100}	null

New instances of ABC:

A1, B2, C3 {001} for SDMQ$_1$

A7, B6, C5 {011} for SDMQ$_1$
A7, B6, C8 {010} for SDMQ$_1$ and SDMQ$_2$

A12, B11, C10 {010} for SDMQ$_1$ and SDMQ$_2$

A14, B16, C13 {110} for SDMQ$_2$
A14, B16, C7 {110} for SDMQ$_2$

check SDMQ$_1$: **{011}** AND {110} = **{010}** invalid
check SDMQ$_2$: **{110}** AND {110} = **{110}** valid

Fig. 4. Search procedure for instances of shared candidates

For the considered instance it is $\{010\} \cup \{100\} = \{110\}$. The same procedure is applied to the remaining instances of A ($A1$, $A7$, $A12$ and $A18$).

Let us now assume that there is a candidate $ABC\{11\}$. To generate its instances, we try to expand already known instances of $AB\{11\}$ by searching the tree for instances of C (line 10). For the aforementioned $A14, B16$ instance, neighbors $C13$ and $C17$ are retrieved from the CiCPI-tree. Both of them are common neighbors of $A14$ and $B16$, therefore instances $A14, B16, C13$ and $A14, B16, C17$ are created (lines 12-13). Once again bitmaps for such instances have to be computed using bitwise OR operation on the bitmap for $A14, B16$ and bitmaps for neighbors. The final bitmap is $\{110\}$ for both mentioned instances. Full example illustrating the search procedure for candidate $ABC\{11\}$ is shown in Fig. 4.

The prevalence for each query is computed by browsing through discovered instances. Because one instance can belong to multiple queries, it is necessary to increment prevalence counters only for applicable queries. To identify queries that a particular instance belongs to, bitmaps representing a sum of distinct query areas identifiers and a bitmap for candidate instance can be used. If for a given query the result of bitwise AND operation is equal to the instance bitmap, it means that such an instance belongs to this query. For example, the instance $A12, B11, C10\{010\}$ belongs to $SDMQ_1$ as well as to $SDMQ_2$ because its bitmap is contained in bitmaps representing areas for both queries (Fig. 4).

4 Experiments

In order to evaluate the performance of the Common iCPI-tree method we performed several experiments. For better control over the experiments, we used synthetic datasets that were generated using a method similar to the approach described in [14]. We have prepared 20 datasets with the following parameters. The number of spatial objects: 50.000-600.000, the number of spatial features: 20-60, the maximal collocation pattern length: 4-8, the number of noise instances: 20%-80%. To simulate dense and sparse datasets we used two spatial frameworks with sizes 10000x10000 and 1000x1000 units. In all tests the neighbor distance threshold was set to 5 units. The experiments were conducted on a Linux PC with AMD Athlon64 4200+ processor and 4 GB of main memory. All programs were written in Java.

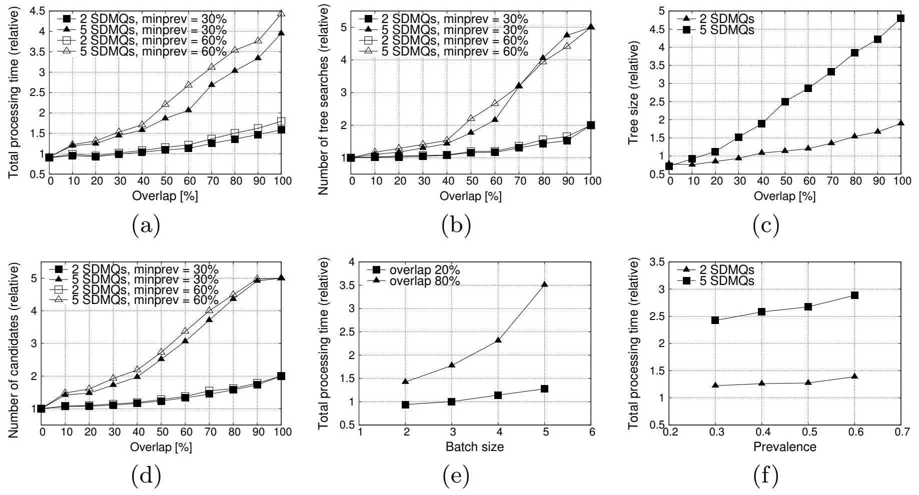


Fig. 5. Performance gain for CiCPI-tree in comparison with sequential processing

We have also prepared a set of 200 sample batches for experiments. We varied the number of queries in batch (2-5), the minimum prevalence threshold (0.3-0.6) and the level of overlapping between queries (0%-100%). The level of overlapping was equal to the average ratio of shared objects between each pair of batched queries. The CiCPI-tree method has been compared with sequential processing. For clarity, all presented results are *relative* to the results obtained from sequential execution. A particular value on the chart should be interpreted as the *acceleration* (or *improvement*) in comparison with sequential execution. For example, for charts presenting the time performance, values below 1, e.g. 0.8, mean that sequential processing took 80% of time required by CiCPI-tree solution, while value 4 means that CiCPI-tree method is 4 times faster than sequential processing.

Figure 5(a) presents how the average processing times change with the increasing overlap threshold. The series include batches of 2 and 5 queries with minimum prevalence set to 30% and 60%. As we expected, the performance gain increases with the increasing overlap of datasets. For bathes of 2 queries, the performance gain can be observed after exceeding 20% overlap, while for 5 queries even for 10% overlap the new algorithm results in faster execution times. When there is no overlap, the CiCPI method is about 10% slower than sequential processing. For 100% overlap batches of 2 and 5 queries are executed with

CiCPI-tree up to 1.75 and 4.5 times faster respectively. For example, in one of our tests the required processing time dropped from 26 minutes to less than 6 minutes.

Figure 5(b) presents the relative number of required tree searches for neighbors. Similarly to the previous experiment, the reduction of searches is increasing with the increasing overlap. With 100% overlap, our new algorithm performs only one tree search for all queries in batch, therefore there are 2 and 5 times less searches for batches of 2 and 5 queries.

Figure 5(c) presents the comparison of CiCPI-tree size and cumulative size of iCPI-trees generated in sequential processing. While the overlap is low, CiCPI-trees can reach bigger sizes than corresponding sets of iCPI-trees due to the overhead resulting from necessity to store additional bitmaps. With increasing overlap, such overhead is compensated by the elimination of redundant branches that can be found in iCPI-trees. In this chart there is no distinction between prevalence thresholds because the size of the CiCPI-tree (and corresponding iCPI-trees) does not depend on the prevalence value.

Figure 5(d) presents how the sharing property of collocations affects the total number of candidates. In comparison with sequential processing the number of candidates is greatly reduced even for low values of overlap threshold, especially for batches of 5 queries.

Finally, in the last series of experiments we analyzed how the number of queries in batch and minimum prevalence affect total processing time. As we expected the bigger the batch is, the bigger performance gain is achieved, notably for higher overlap thresholds (Fig. 5(e)). On the contrary, the prevalence measure does not have such essential impact on the performance gain (Fig. 5(f)). However, when the minimum prevalence threshold is low more multi-feature candidates are being generated. The possibility of sharing such candidates is limited and therefore the acceleration is reduced.

5 Summary and Future Work

In this paper we have defined the problem of efficient execution of batched spatial data mining queries for collocation patterns discovery. We have proposed a new algorithm, called CiCPI-tree, that significantly outperforms the straightforward serial execution of multiple queries. Processing times are reduced by eliminating redundant searches for neighbors and introducing shared representation of collocation instances with combined candidates generation.

In the future work we will focus on memory constraints that can be crucial when a batch of queries is being processed. In the ideal circumstances, the CiCPI-tree structure should fit in memory, however in real life applications this can be impossible. We believe that our previous researches on collocation pattern mining in limited memory environments can be adopted to concurrent processing of spatial data mining queries. Another interesting subject involve different strategies for processing spatial data mining queries, e.g., sets of queries with random times of arrival.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. SIGMOD Rec. 22(2), 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994)
3. Boinski, P., Zakrzewicz, M.: Hash Join Based Spatial Collocation Pattern Mining. Foundations of Computing and Decision Sciences 36(1), 3–15 (2011)
4. Boinski, P., Zakrzewicz, M.: Collocation Pattern Mining in a Limited Memory Environment Using Materialized iCPI-Tree. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 279–290. Springer, Heidelberg (2012)
5. Boinski, P., Zakrzewicz, M.: Partitioning Approach to Collocation Pattern Mining in Limited Memory Environment Using Materialized iCPI-Trees. In: Morzy, T., Härder, T., Wrembel, R. (eds.) Advances in Databases and Information Systems. AISC, vol. 186, pp. 19–30. Springer, Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-32741-4_3`
6. Celik, M., Kang, J.M., Shekhar, S.: Zonal Co-location Pattern Discovery with Dynamic Parameters. In: ICDM, pp. 433–438. IEEE Computer Society (2007)
7. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery in Databases. AI Magazine 17, 37–54 (1996)
8. Giannikis, G., Alonso, G., Kossmann, D.: SharedDB: Killing One Thousand Queries With One Stone. Proc. VLDB Endow. 5(6), 526–537 (2012), `http://dl.acm.org/citation.cfm?id=2168651.2168654`
9. He, J., He, Q., Qian, F., Chen, Q.: Incremental Maintenance of Discovered Spatial Colocation Patterns. In: Proceedings of the 2008 IEEE International Conference on Data Mining Workshops, ICDMW 2008, pp. 399–407. IEEE Computer Society, Washington, DC (2008), `http://dx.doi.org/10.1109/ICDMW.2008.60`
10. Sellis, T.K.: Multiple-query optimization. ACM Trans. Database Syst. 13(1), 23–52 (1988), `http://doi.acm.org/10.1145/42201.42203`
11. Shekhar, S., Huang, Y.: Discovering Spatial Co-location Patterns: A Summary of Results. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 236–256. Springer, Heidelberg (2001)
12. Wang, L., Bao, Y., Lu, J.: Efficient Discovery of Spatial Co-Location Patterns Using the iCPI-tree. The Open Information Systems Journal 3(2), 69–80 (2009)
13. Wojciechowski, M., Zakrzewicz, M.: Methods for Batch Processing of Data Mining Queries. In: Haav, H.M., Kalja, A. (eds.) Proceedings of the Fifth International Baltic Conference on Databases and Information Systems (DB&IS 2002), pp. 225–236. Institute of Cybernetics at Tallin Technical University (June 2002)
14. Yoo, J.S., Shekhar, S., Celik, M.: A Join-Less Approach for Co-Location Pattern Mining: A Summary of Results. In: Proceedings of the IEEE International Conference on Data Mining, pp. 813–816. IEEE Computer Society, Washington (2005)