Pre-processing and Indexing techniques for Constellation Queries in Big Data

Amir Khatibi[1], Fabio Porto[1], Joao Guilherme Rittmeyer[1]
Eduardo Ogasawara[2], Patrick Valduriez[3], and Dennis Shasha[4]

[1] DEXL Lab, LNCC, Petropolis, RJ, Brazil, {ahassan,joanonr,fporto}@lncc.br
[2] C.S. department, CEFET/RJ, RJ, Brazil, eogasawara@ieee.org
[3] Zenith, LIRMM, Inria, Montpellier, France, patrick.valduriez@inria.fr
[4] NYU, Courant Institute, USA, shasha@courant.nyu.edu

Abstract. Geometric patterns are defined by a spatial distribution of a set of objects. They can be found in many spatial datasets as in seismic, astronomy, and transportation. A particular interesting geometric pattern is exhibited by the Einstein cross, which is an astronomical phenomenon in which a single quasar is observed as four distinct sky objects when captured by earth telescopes. Finding such crosses, as well as other geometric patterns, collectively referred to as constellation queries, is a challenging problem as the potential number of sets of elements that compose shapes is exponentially large in the size of the dataset and the query pattern. In this paper we propose algorithms to optimize the computation of constellation queries. Our techniques involve pre-processing the query to reduce its dimensionality as well as indexing the data to fasten stars neighboring computation using a PH-tree. We have implemented our techniques in Spark and evaluated our techniques by a series of experiments. The PH-tree indexing showed very good results and guarantees query answer completeness.
Keywords: Constellation Queries, Geometric Shapes, PH-tree Indexing, Dataset Pre-Processing, Query Pre-Processing, SQL extension

1 Introduction

The availability of large datasets in science, web and mobile applications enables new interpretations about natural phenomena and human behavior. From inferring sites of touristic interest based on pictures taken in social network applications [1] to the existence of dark matter inferred from multiple occurrences of quasars [2], new knowledge emerges whenever individual observations are combined allowing for queries on patterns. This paper extends the algorithms and techniques in a type of pattern queries in spatial databases that we referred to as constellation queries (CQ) in our previous work [3]. Constellation queries are obtained from compositions of individual elements in large datasets. CQ computation entails matching geometric pattern queries against sets of individual data observations, such that each set collectively agrees in the geometric constraints expressed by the pattern query. In particular, we are interested in efficiently finding patterns like the Einstein cross (EC). From a constellation query representing

the EC, involving a set of sky objects,we should compare it's attributes with other set of sky objects in a astronomy catalog. The data scheme of an astronomy catalog such as Sloan Digital Sky Survey (SDSS)[5] is as the following relation: *SDSS (Obj_ID, RA, DEC, u, g, r, i, z, Redshift, ...)* . The attributes u, g, r, i, z refer to the magnitude of light emitted by an object measured in specific wavelength. A constellation in the SDSS scenario would be defined by a sequence of objects from the catalog whose spatial distribution forms a shape conforming to a constellation query. In this paper we focus on improving the process of executing constellation queries, as described in [3], by applying query pre-processing and indexing techniques. We propose two new steps that could be executed prior to processing a user's query: (a) query pre-processing and (b) dataset pre-processing using the PH-tree algorithm. The advantage of pre-processing is hidden in the fact that solving a constellation query in a big dataset is hard due to the numerous possible compositions from billions of observations. In general, for a big dataset D and a number k of elements in the pattern query, the number of possible candidate combinations, $\binom{|D|}{k}$ is the number of ways to choose k items from D. Dataset pre-processing aims to reduce the size of D while pre-processing the query tries to reduce the size of k in a way that without losing the quality of solutions, we process the constellation queries in a shorter time. Our main contributions in this paper are: the adoption of the PH-tree indexing algorithm; the definition of a SQL extension for the constellation queries; and a query and dataset pre-processing techniques.

The rest of the paper is organized as follows: in section 2, we review the related works. Then section 3 presents the problem statement. Section 4 discusses our contributions in order to improve the CQ processing. In section 5, we show our experimental results. Finally, section 6 concludes.

2 Related Works

The relational data model, and SQL therein, adopt set based constraints that are imposed to each individual tuple in order to appear in a query result set. There are nevertheless many practical real-world problems such as geometric pattern queries that require tuples in a set to collectively satisfy a set of constraints [4], [5]. The former is concerned with topological constraints among multi-dimensional objects. In [5], the authors present package queries that enable users to express constraints over package of tuples. The approach considers local constraints, as traditional where clause in SQL, and global constraint that refer to packages of tuples. The query expressed using additional SQL clauses is rewritten into an expression composed of a SQL query, submitted to a relational database, and an integer linear program that solves the package constraints on top of database results. Constellation queries [3] is a class of package queries, in which a geometric shape defines the global constraints. The assessment of such constraints requires tuples in candidate packages to be labelled so that they can be referred to

[5] http://skyserver.sdss.org/dr12/en/help/browser/browser.aspx

elements of the query according to the ordering imposed by the shape. Expressing CQ as package queries leads to self joins in the number of elements of the query that would be impractical for large datasets. Constellation queries combine quad-trees, matrix multiplication, and un-indexed join processing to discover sets of elements that match a geometric pattern within some additive factor on the pairwise distances.

3 Problem Formulation

We formulate the problem of solving spatial pattern queries, referred to as constellation queries as follows: We consider a Big Dataset D defined as a set of elements $D = \{e_1, e_2, \ldots, e_n\}$, in which each e_i, $1 \leq i \leq n$, is an element of a domain Dom. Furthermore, $e_i =< atr_1, atr_2, \ldots, atr_m >$, such that atr_j, $1 \leq j \leq m$, is a value describing a characteristic of e_i. Conversely, a sample query Q is defined as a set of elements $Q = \{q_1, q_2, \ldots, q_k\}$, where q_j, $1 \leq j \leq k$, are elements of the same domain Dom as D. We further adopt the following definitions: **Definition 1:** A boolean function **fe** $(e_i : Dom,\ Q_j : Q,\ \theta : \mathbb{R})$ verifies whether an element e_i from a domain Dom is at most at a similarity distance θ from any element $q_i\ in\ Q_j$. **Definition 2:** A boolean function **fs** $(C_i : Dom,\ Q_j : Q,\ \epsilon : \mathbb{R})$ verifies whether the sets (C_i, Q_j) is at most a distance of ϵ with respect to the similarity of their composition model. Moreover, an increasing value for ϵ flexibilizes the distance evaluation. Finally, the semantics of *fs* evaluation considers all permutations of C_i. **Problem statement**: given a Big dataset D and a sample query Q_j, both with elements in a domain Dom, and constants $\theta = r1$ and $\epsilon = r2$, efficiently compute the set of all compositions $C = \{C_1, C_2, \ldots, C_m\}$, $C_i \subset D$, with $|C_i| = |Q_j|$, such that $fe(e_u, Q_j, r1) = true$, for all $e_u \in C_i$, and $fs(C_i, Q_j, r2) = true$, for all $1 \leq i \leq m$.

4 CQ Processing

This section discusses our contributions in improving the performance of CQ processing. In general, query execution in large datasets is a challenge due to long execution time. However, one way to improve the user's experience of the system is pre-processing the input large dataset so that at the time of query execution, the system provides a quicker response as it has done some of the steps in advance. Likewise, another alternative to reduce the user's waiting time is by pre-processing the query itself. In the next subsections we elaborate each of these pre-processing steps.

4.1 Query Pre-processing

In constellation queries, the number of possible compositions increases exponentially with the query size k, $(N \approx |D|^k)$. One may intuitively suggest reducing the size k of a constellation query in order to save computation. As it turns out, elements in a full query Q_k may induce redundant constraints, specially those

located very close to each other. In this context, $Q_{k\prime}$ can be built from subset M of elements of Q_k that only includes elements that are candidates for defining the geometric shape induced by $Q_{k\prime}$ ($k\prime \leq k$). As an example, $Q_{k\prime}$ may only include elements that are at a certain distance apart. Once $Q_{k\prime}$ has been fixed, an anchor element q_0 is picked and pairwise distances from it to every remaining element $Q_{k\prime} \setminus q_0$ are computed.

Axiom 1: *Given constellation queries $Q_{k\prime}$ and Q_k, such that $Q_{k\prime} = Q_k - Q_v$, $Q_v \subset Q_k$, and $S_{k\prime}$ a set of all sequences $s_{k\prime}$, such that $s_{k\prime}$ matches $Q_{k\prime}$, and S_k a set of all sequences s_k,such that s_k matches Q_k, we say that $Q_{k\prime}$ is equivalent to $Q_k \leftrightarrow (1 - F-measure(S_{k\prime},S_k))^6 < \delta$ and $shape(s_{k\prime}) \cong shape(s_k)$.*

4.2 Query Transformation

The discussion in Section 4.1 raises an optimization strategy for constellation queries based on query pre-processing. The intuition is that some elements would offer little contribution for constellation specification but would add to the query elapsed-time. In this context, detecting such elements, and subtracting them from the query, could reduce query elapsed-time. Query modification, however, must be subject to equivalence guarantees between the original query and its reduced version, as stated by axiom 1. Proposition 1 suggests a verifiable condition to assess constellation query equivalence. In section 5 we experimentally evaluate the proposition 1.

Proposition 1: Given constellation queries $Q_{k\prime}$ and Q_k, such that $Q_{k\prime} = Q_k - Q_v$, $Q_v \subset Q_k$, If for each query element $q_j \in Q_v \ \exists \ q_i \in (Q_k - Q_v)$, such that: $distance(q_j,q_i) \leq \epsilon$ then $Q_{k\prime}$ is equivalent to Q_k.

4.3 Dataset Pre-processing

The second opportunity to reduce the complexity of constellation queries, approximately $N \approx |D|^k$, is to reduce the size of D. Additionally, we want to look for elements to build compositions that are candidates for producing shapes geometrically close to that of Q_k. The element q_0 from Q_k becomes a key to such reduction. It enables to fix an anchor for building compositions both with respect to attribute values and to shape constraints. Regarding the former, q_0 reduces the size of D to $|\sigma_{f(e_i)}(D)|$. In other words, we only test for compositions that hold a similar anchor element as q_0 in Q_k. Secondly, as we scan D, looking for anchor elements, we store each element in a PH-tree [6]. The latter is used in the sequel to search for candidate elements in the neighborhood of selected e_i within a radius ρ, corresponding to the distance of the furthest query element q_i to q_0 plus ϵ as depicted in Figure 1. The possible constellations having e as anchor are within this set.

The constellations based on an anchor e includes the neighbors within radius ρ whose distances to e match the distance d_i of some query element in Q_k. For a constellation query with k query elements, we produce $k-1$ buckets holding

[6] F-measure = $precision/recall$.

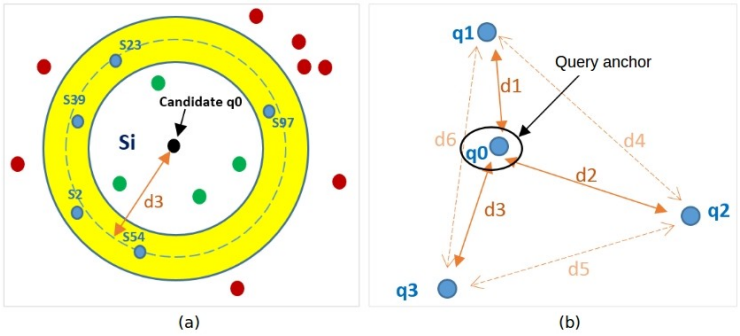

Fig. 1. (a) candidate anchor and neighboring ring elements and (b) geometric query

neighbors of anchor e. The matching constellations are the sets of k-1 neighbors of e with one element from each bucket and having pairwise distances matching the corresponding pairwise distances of elements in Q_k. The pre-processing of D produces an intermediary relation $D\prime$, substantially smaller than D, with schema $D\prime = (e : Dom, list\ of\ neighbors < n_k, d_k >)$, where e refers to an anchor element in D, and n_k is a neighbor of e in D with distance less than the largest distance (q_i, q_0), for all q_i in Q_k. An interesting side-effect of computing $D\prime$ is that it fosters the parallelization of the constellation query processing by enabling a balanced distribution of $D\prime$ tuples over a cluster of machines to be evaluated by a Big Data processing framework, such as Spark.

5 Experiments

This section presents the experiments conducted to evaluate the pre-processing and indexing techniques in Constellation queries. We first evaluate the query pre-processing technique. Next, we evaluate the performance of the PH-tree applied to the Constellation query problem.

5.1 Query Pre-processing

In our first experiment, we randomly generate 10 queries with 7 elements each. Next, we reduce the query size, by randomly selecting one element to be deleted from each query. Figure 2(a) shows the results in elapsed-time, in seconds, for each query, sorted in ascending order by the elapsed time of the full query, k=7. Two main observations can be drawn from this plot. Firstly, indeed, as expected, the query size impacts in its response-time. The elapsed-time of the full query dominates the ones from its reduced versions. Moreover, the decrease in computation costs follows the modification of the query once one of its elements is, randomly, subtracted. However, as it can be more clearly verified in query 8, the choice of the element to be subtracted contributes differently to the query outcome, as it can be observed by a drastic reduction on the query elapsed-time.

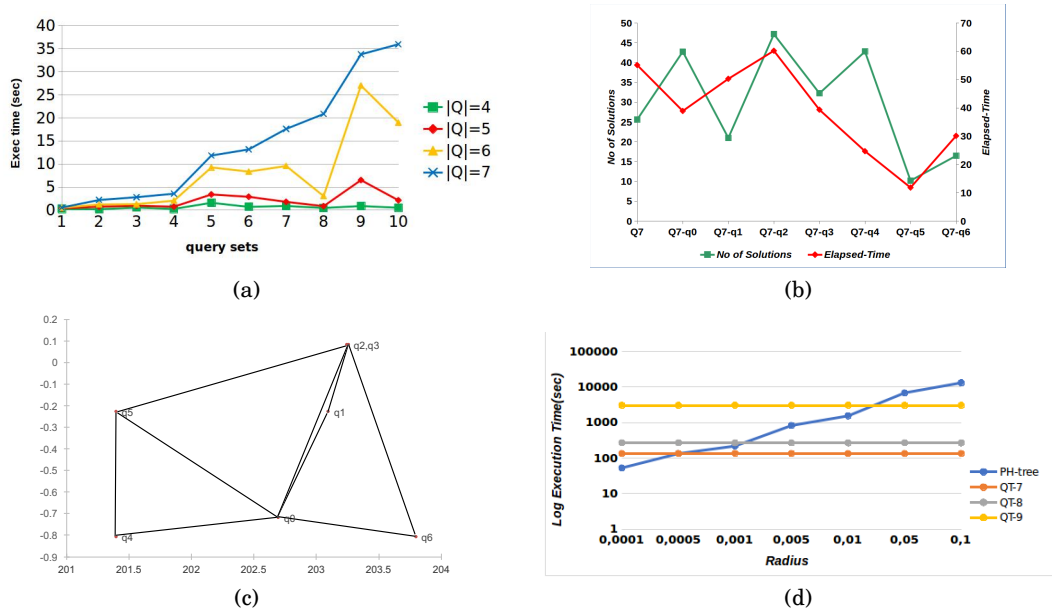

Fig. 2. (a) Query Size Reduction Impact (b)Effect on the Selected Query Element, (c) Assessing proposition 1 (d) PH-tree versus Quad-tree - Log Execution Time

In order to explore this last observation, we measured the effect of selecting different query elements to be subtracted from the query. Figure 2(b) plots, in the horizontal axis, the query *id* against the query elapsed-time, in the vertical axis. Curves show: (i) the number of solutions (divided by 10^4), in blue, and (ii) the query elapsed-time, in red. Query Q_7 corresponds to the full query. As it can be seen, the choice significantly impacts on both the elapsed-time and the answer set. The difference in the answer set reflects in variation of the $F-measure$. Moreover, as observed when subtracting q_2 from Q_7, the elapsed-time increases with respect to that of the full query, contrarily to the intuition. As a matter of fact, in Section 4.1, Proposition 1 states that modification of constellation query should be limited to query elements in close distance to one of its neighbors. Under this constraint, the results of the modified query and the full query would be equivalent. The query depicted in Figure 2(c) was specially constructed such that $dist(q_2,q_3) \leq epsilon$. According to Proposition 1, excluding either q_2 or q_3 would produce an equivalent query answer set, compared to the full query.

Table 1 depicts the results of running the query shown in Figure 2(c). Query id=1 corresponds to the full query. The remaining queries are obtained from query 1 by deleting an element corresponding to its id. The *Fs elapsed-time* column highlights the query time corresponding to running the composition function, which is the dominant constellation query cost. Unfortunately, Table 1 contradicts the premise exposed in Proposition 1. As it can be observed, the deletion of q_2 or q_3 produces many false positives. As a result, Q_2 and Q_3 are not equivalent to Q_1,

Table 1. Equivalence of Queries

Query id	F-measure	Recall	No. Solutions	Fs elapsed-time
1	1	1	38	0.3416
2	0.0112	1	6690	2.7499
3	0.01234	1	6117	2.6596
4	0.7102	1	69	0.0500
5	0.7446	0.9219	56	0.0512

according to axiom 1, despite being very close to each other. Moreover, the effect in performance of running Q_2 and Q_3 is worse than running the full query Q_1 . These results can be easily explained and, in fact, are co-related. Firstly, as the query elements are very close in space, they impose a severe restriction in the result set. Only candidate solutions that include pairs of elements, associated to $bucket_2$ and $bucket_3$, with $dist(e_2,e_3) \leq \epsilon$ are selected, expressing a very selective predicate. Conversely, modified queries caused by the deletion of either q_4 or q_5 exhibit higher precision and lower elapsed-time, with $dist(q_4,q_5) > \epsilon$. The later indicates that the original shape, as specified by the full query, would have been sensibly modified by the query modification. These experiments show that query modification must be exercised with extreme caution. An automatic decision, in line with proposition 1, is only possible if knowledge about spatial data distribution is available such that the precision of the modified query Q_v, with respect to the full query Q_k, is above a threshold. Moreover, the user may indicate her preferences regarding:elapsed-time; shape equivalence and F-measure. Selecting a faster execution with flexibility on the remaining parameters may open opportunity to query pre-processing under proposition 1.

5.2 PH-tree versus Quad-tree

In this section, we evaluate the efficiency of the PH-tree as the basic data structure used to aid retrieving the set of candidate matching neighboring stars for each given star in the dataset.In [3], a quad-tree data structure reduces the number of matching operations by adopting a representative of set of stars covered by the Quad-tree leaf nodes. In this context, node centroids are compared and, in case of successful matching, the process is carried over to stars covered by the matching nodes. Conversely, PH-tree indexes all stars, and neighboring computation must be exercised through all stars in the dataset. Thus, this experiments compares both approaches, considering the amount of memory used to build the data structure and the elapsed-time taken when retrieving the neighbors for stars in the SDSS R-12 dataset. The latter includes approximately 7 million stars and its file size is 1 GB. The Einstein cross is the query used with maximum distance between elements in order of 10^{-5}. The PH-tree implementation showed, in average, 1.5 times higher memory footprint than the quad-tree. The former consumed 4.53 GB for 3.02 GB for the latter memory space consumption.

Figure 2(d) depicts our results on searching for candidates elapsed-time (logarithmic scale). Values for the PH-tree and the Quad-tree considered the average of 10 runs, but the ones for QT-9, which only report the average of 3 runs, due to the long elapsed-time for processing the query and, at the same time, low variance among runs. Results for QT-7, QT-8 and QT-9 report on the impact on the elapsed-time for the neighbors search when the Quad-tree is built with different heights. As it can be observed, the quad-tree with heights 7 and 8 showed better results than that of the PH-tree. Height 9, however, is the inflexion point. From height 9 on, the quad-tree implementation becomes extremely costly and PH-tree is a clear winner. It is also important to note that on higher heights, the quad-tree solutions may hide candidate solutions, as they may be covered within the same node, avoiding the procedure to detect them as candidates. Thus, despite presenting better performance, high height scenarios may lead to incomplete answers.

6 Conclusion

In this paper, we proposed pre-computing and indexing techniques for processing constellation queries. The query pre-processing technique involves selecting a query element to be excluded from the pattern in order to reduce the query complexity. As our experiments have shown, this process requires further investigation. Our proposition of excluding query elements within a pre-defined threshold distance eliminates a higher selective constraint, considerably increasing the answer set and reducing its *F-measure*. Thus, we consider that further investigation is needed to determine query elements that can be deleted without compromising query answer quality. Finally, the PH-tree presented very good results in computing neighbors of stars. The technique requires larger memory space but produces very efficient neighbors computation elapsed-time with complete solution set. We intend to explore techniques that would enable the distribution of the index structure in order to cope with even larger input datasets.

References

1. I. R. Brilhante, J. A. F. de Macêdo, F. M. Nardini, R. Perego, and C. Renso, "On planning sightseeing tours with tripbuilder," *Inf. Process. Manage.*, vol. 51, no. 2, pp. 1–15, 2015.
2. D. Overbye, "Astronomers observe supernova and find they are watching reruns," New York Times, USA, 2015.
3. F. Porto, A. Khatibi, J. R. Nobre, E. Ogasawara, P. Valduriez, and D. Shasha, "Constellation queries over big data," *eprint arXiv:1703.02638 - Bibliographic Code: 2017arXiv170302638P*, 03/2017.
4. N. M. D. Papadias and V. Delis, "Algorithms for querying by spatial structure," in *Proc. of the* $24^{th} VLDB Conference, pp.$ 546 – –557, 1998.
5. A. M.Brucato, J. F.Beltran and A.Meliou, "Scalable package queries in relational database systems," *Proceedings of the VLDB Endowment*, vol. 9, pp. 576–597, 2016.
6. T. Zäschke, C. Zimmerli, and M. C. Norrie, "The ph-tree: a space-efficient storage structure and multi-dimensional index," in *Proceedings of the ACM SIGMOD international conference on Management of data*, pp. 397–408, 2014.