# Point Pattern Search in Big Data

Fabio Porto
LNCC, DEXL Lab
Petropolis, Rio de Janeiro, Brazil
fporto@lncc.br

João N. Rittmeyer
LNCC, Dexl Lab
Petropolis, Rio de Janeiro, Brazil
joaonr@lncc.br

Eduardo Ogasawara
CEFET-RJ
Rio de Janeiro, Brazil
eogasawara@ieee.org

Alberto Krone-Martins
University of Lisboa
Lisbon, Portugal
algol@sim.ul.pt

Patrick Valduriez
Inria, LIRMM, Computational Biology Institute and University of Montpellier, France
patrick.valduriez@inria.fr

Dennis Shasha
New York University
New York, USA
shasha@courant.nyu.edu

## ABSTRACT

Consider a set of points P in space with at least some of the pairwise distances specified. Given this set P, consider the following three kinds of queries against a database D of points : (i) pure constellation query: find all sets S in D of size |P| that exactly match the pairwise distances within P up to an additive error $\epsilon$; (ii) isotropic constellation queries: find all sets S in D of size |P| such that there exists some scale factor $f$ for which the distances between pairs in S exactly match $f$ times the distances between corresponding pairs of P up to an additive $\epsilon$; (iii) non-isotropic constellation queries: find all sets S in D of size |P| such that there exists some scale factor $f$ and for at least some pairs of points, a maximum stretch factor $m_{i,j} > 1$ such that $(f \times m_{i,j} \times dist(pi, pj)) + \epsilon >$ dist(si,sj) $> (f \times dist(pi, pj)) - \epsilon$. Finding matches to such queries has applications to spatial data in astronomical, seismic, and any domain in which (approximate, scale-independent) geometrical matching is required. Answering the isotropic and non-isotropic queries is challenging because scale factors and stretch factors may take any of an infinite number of values. This paper proposes practically efficient sequential and distributed algorithms for pure, isotropic, and non-isotropic constellation queries. As far as we know, this is the first work to address isotropic and non-isotropic queries.

## CCS CONCEPTS

• **Information systems → Information systems applications**; **Information retrieval query processing**; *Decision support systems*;

## KEYWORDS

Point set Registration; Geometrical Patterns, Spatial Patterns; Pattern Search; Isotropic; Big Data; Distance Matching


*SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy*

ACM ISBN 978-1-4503-6505-5/18/07…$15.00
https://doi.org/10.1145/3221269.3221294

**ACM Reference Format:**
Fabio Porto, João N. Rittmeyer, Eduardo Ogasawara, Alberto Krone-Martins, Patrick Valduriez, and Dennis Shasha. 2018. Point Pattern Search in Big Data. In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy,* Michael Bohlen, Johann Gamper, and Dimitris Sacharidis (Eds.). ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3221269.3221294

## 1 INTRODUCTION

Finding collections of objects having some metric relationship of interest to one another has many applications in astronomy, geology and design applications such as architecture and town planning, to name a few. The problem has different names depending on the discipline, including *Object Identification* [19], *Graph Queries* [22], *Point Set Registration* [12, 20] and, in general, *Pattern Recognition* [4]. When the goal is to find just one matching set of points to a given pattern set of points, the problem is called *point set registration* and includes problems such as finding submarines in a sonar noise cloud and aligning images of stars [12, 20]. By contrast, this paper focuses on methods that match a pattern to *all* subsets of points in a database at different scales and with different stretch factors within an additive error.

Consider the following use case: given the points describing a known pattern, say a hexagon, find sets of stars that form a hexagon at all scales allowing additive noise.

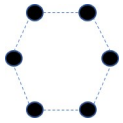


**Figure 1: Can we find all matching sets of stars having this hexagonal shape no matter what the scale and allowing an additive error?**

Extracting such *point patterns* or *constellations* from large datasets entails matching a pattern query to sets of points, such that each set obeys the geometric constraints expressed by the pattern query according to some matching criterion. We consider three matching criteria in order of increasing generality. We begin with a discussion of *pure constellation queries* in which responsive sets of points must match the pattern distances exactly up to an additive factor.

This is very similar to point set registration. The paper then shows how to extend pure constellation queries to *isotropic* queries in which responsive points match the pattern up to an arbitrary scale factor and an additive error factor. Finally, the paper discusses *non-isotropic constellation queries* in which at least a subset of distances can be stretched up to a certain point. Our running example will come from astronomy, but the algorithms are generally applicable.

Our contributions are:

(1) Constructing algorithms to answer isotropic queries even though scale factors can take on any value.
(2) Extending isotropic query algorithms to the non-isotropic setting where, in addition, stretch factors can take any value from 1 to some user-specified maximum.
(3) Exploring a space of algorithmic trade-offs under a Spark distributed implementation.

The remainder of this paper is organized as follows. Section 2 formalizes the problem of answering general constellation queries. Next, in section 3, we introduce algorithms for *Pure Constellations*. In section 4, we show how to extend pure constellation queries to isotropic and non-isotropic queries. That section presents our major algorithmic innovations. In section 5, a thorough experimental evaluation explores the problem parameter space based on a distributed implementation of all algorithms using Apache Spark. Section 6 highlights related work. Finally, section 7 concludes.

## 2 PROBLEM FORMULATION

A Dataset $D$ is defined as a set of elements having coordinates in some n-dimensional space.

A constellation pattern $Q_k = \{q_1, q_2, \ldots, q_k\}$ is (i) a set of $k$ elements at certain locations and an additive error factor $\epsilon$.

As introduced informally above, we define three kinds of constellation queries, in increasing levels of generality: (i) pure constellation; (ii) isotropic constellation, and (iii) non-isotropic constellation. Solutions to pure constellation queries consist of sets of $k$ elements whose pairwise distances exactly match those of the query up to an additive $\epsilon$. A set $s_1, ..., s_k$ of elements of length $k$ in $D$ *purely matches* query pattern $Q = p_1, ..., p_k$ if, for all i and j between 1 and k, $dist(s_i, s_j)$=$(dist(p_i, p_j))\pm\epsilon$. Because the match depends only on distances, pure constellation queries are translation- and rotation-invariant.

Isotropic constellation queries admit any solution equal (up to an additive factor) to the original query but where distances can all be multiplied by a single scale factor that can take an arbitrary value. A set $s = s_1, \ldots, s_k$ elements in $D$ *isotropically matches* query $Q$ if there exists a scale factor $f$ such that for every $i, j \leq k$, $dist(s_i, s_j)$=$f \times (dist(p_i, p_j))\pm\epsilon$. The challenge is to find every match even though $f$ can take any arbitrary value.

Finally, non-isotropic constellation queries admit controlled deformations at one or more pairwise distances. That is, a set $s = s_1, \ldots, s_k$ in $D$ *non-isotropically matches* a query pattern $Q = p_1, ..., p_k$ if there is some scale factor $f$ and some set of stretch factor modifiers $m_{i,j} \geq 1$ such that dist($s_i$, $s_j$) is within ($f \times$ dist($p_i$, $p_j$)) - $\epsilon$ and ($f \times m_{i,j} \times$ dist($p_i$, $p_j$)) + $\epsilon$. The additional challenge here is that each stretch factor for each pair $i$ and $j$ can also take an arbitrary value between one and $m_{i,j}$. Again, we want to find all such matches for every $f$ and every stretch factor up to $m_{i,j}$.

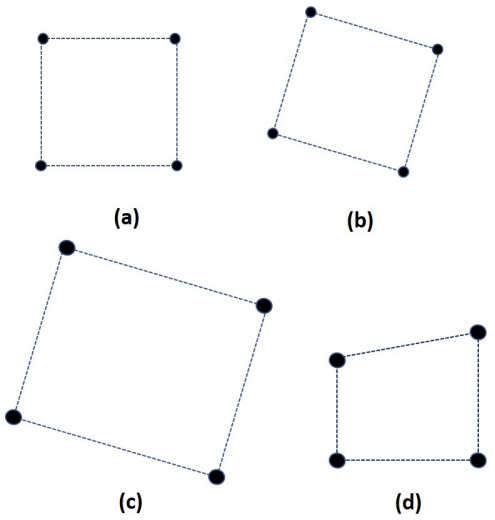



**Figure 2: Geometric pattern query a, assuming zero additive error for purposes of illustration: b is a pure constellation query solution forming a rotated square, c is an isotropic constellation query solution forming a larger square, and d is a non-isotropic constellation query solution (a square can match one of a constrained but infinite set of quadrilaterals).**

Figures 2.b, 2.c, and 2.d depict different kinds of Constellation Queries expressed by Figure 2.a. In Figure 2.b, a pure constellation solution offers the same distances among stars, with the constellation slightly rotated in its center. Solution of Figure 2.c is isotropic concerning the query, whereas the solution of Figure 2.d is a trapezoid, reflecting a stretch to three sides, forming a non-isotropic solution. When noise is present, even pure constellation queries may cause a square query, as in this example, to match non-square constellations.

This paper first shows how to process pure constellation queries, because such queries are the simplest to process and because our algorithms for processing isotropic and non-isotropic queries generalize the algorithm for pure constellation queries.

## 3 PURE CONSTELLATION QUERIES

Answering *pure constellation queries* on astronomical data requires efficient query processing techniques as the catalog may hold billions of sky objects. The SDSS data release 14, for instance, holds approximately 1.3 billion objects, which would lead to the evaluation of roughly $\frac{1.3B^4}{4!} = 1.2 \cdot 10^{35}$ candidate sets for a query of size 4. (Note that the solution set size could in fact be of that order of magnitude. For example, 1/4 of the stars could be at each of the four corners of a square.) Point set registration algorithms, such as Iterative Closest Point (ICP) can do this in $O(n * m)$ time, where n is the number of elements in the query and m is the size of the dataset. However, ICP requires the two sets of points to be roughly of the same size and near one another, which is too restrictive for point pattern search. Thus, to render the problem computationally tractable, when $\binom{|D|}{k}$ is big, our strategy to process pure constellation queries involves three main techniques. First, we use a quadtree

to retrieve stars at most at a max-distance from each star. Second, we use query element properties (like brightness and frequency in the astronomical context) to constrain sets of candidate neighbors. Third, for each star and its candidate neighbors, we solve a k-1 spatial star join combined with a bucketed k-2 spatial join. For example, to look for a triangle having side lengths L1, L2, and L3, we want to find points a, b, c such that the distance between a and b is of distance L1 ( $\pm\epsilon$), between b and c is of distance L2 ( $\pm\epsilon$), and the distance between c and a is L3 ( $\pm\epsilon$).

The next sections describe the query processing techniques in detail.

## 3.1 Data Partitioning

Answering pure constellation queries involves matching each star in the catalog against all neighboring stars at appropriate distances based on the query, a costly procedure in large catalogs. To reduce this cost and make the deployment on Spark more efficient, we partition the dataset off-line in a pre-processing step. Data partitioning applies an equi-depth histogram [14] based algorithm on one of the spatial dimensions.

The output of the partitioning procedure splits the dataset into $p$ primary partitions. In the astronomy case this will yield rings around the earth. Each primary partition is extended with a neighborhood set containing objects in the two neighboring partitions at most a *max-distance* from the borders. Max-distance represents the maximum radius from the *anchor* (intuitively, the median centroid) of any query of interest. For astronomy, max distance has been defined as 5, 10 and 15 arcsecs (0.0041 degrees), based on feedback from astronomers. The minimum width of each partition is max-distance plus the error bound, so that any star in the primary partition that potentially maps to the anchor of the query will find all relevant stars in the primary partition itself or in one or both of the two neighboring partitions.

Partitioning enables the parallelization of the search procedure as all stars of one partition can be tested as matches to the query anchor in parallel with all stars of other partitions.

## 3.2 Indexing Data Spaces

Our method parallelizes across partitions, so we explain what happens within each partition. The data space in a partition includes a primary and two neighboring partitions (for the two-dimensional case). To reduce search time, we build a quadtree index [17] covering the complete data space of a partition. (Thus, quadtrees of neighboring partitions will include some of the same stars.) The quadtree construction process builds a tree such that leaf nodes maintain the following property: any two stars covered in a leaf node quadrant do not both appear in a solution. This is achieved by specifying the tree height as a function of the shortest distance among elements of the query.

Expanding on the intuition above, the *anchor* of a query is a point in the query whose maximum distance to any other point in the query is minimal. If there are several such points, any one will do.

Now for each leaf L of the quadtree (different leaves can be treated in parallel), find all the other leaves that could possibly have relevant stars based on the size of L and the maximum distance from the anchor to any other point in the pattern query. Now consider each star in L as a potential match to the query anchor.

## 3.3 Filtering Step

The neighborhood filtering step is illustrated in Figure 3, in (a) a query Q has an anchor element $q_0$, and the max-distance corresponds to the largest distance $\rho$ to the remaining query elements $d_{0,2}$. In (b), a star from the current leaf L (which we will call an anchor representative) is picked as a potential match to the anchor, and all neighboring stars within distance $d_{0,2} + \epsilon$. These are preliminary candidates for distance matching.

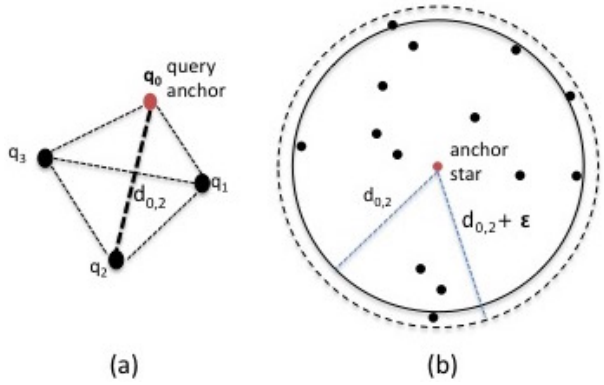



**Figure 3: (a) Pure constellation query with anchor and maximum distance (b) Neighboring elements of anchor representative**

## 3.4 K-1 Spatial Join

As noted above, the query anchor $q_0$ is the point closest to all points in the query (whose maximum distance to another point in the query is minimum). Thus, for each pattern element $q_i$,$i \neq 0$, we determine whether the distance between a neighbor star $s'$ to the anchor representative $s$ is dist($q_0$, $q_i$) $\pm$ an additive factor of $\epsilon$. If so, $s'$ is placed in bucket $b_i$, for $1 \leq i \leq k-1$. Note that $s'$ may be placed in several buckets.

## 3.5 Composition algorithms

Recall that each bucket $i$ holds the set of stars whose distance from the candidate anchor star $s$ is equal ($\pm\epsilon$) to the distance between the query anchor and star $i$ of the query.

*3.5.1 Spatial Bucket Join.* The most straightforward way to find star sets that match the pattern is by directly joining the buckets of candidate elements based on the corresponding pairwise distances between query elements. Fortunately, this works well. In order to form a candidate solution, each bucket is viewed as a relation, having as a schema each candidate star's spatial coordinates and id, $B_i(starid, ra, dec)$. A solution is obtained whenever a tuple is produced having one neighbor element from each bucket, such that the distances between each element in the solution *distance-match* those among respective query elements, $\pm\epsilon$. *Bucket_SJ* performs

a spatial nested loop algorithm to traverse the buckets of candidate elements and checks for distance predicates. Specifically, the *distance-match* constraint corresponds to applying a cyclic join among all buckets in the bucket set followed by a filter among non-neighbors in the cycle. For example, *Bucket_SJ* would find pairs ($t_1$, $t_2$) where $t_1$ is from $B_i$ with and $t_2$ from $B_{i+1}$ if dist($t_1$, $t_2$) is within dist($p_i$, $p_{i+1}$) $\pm\epsilon$. Then given these pairs for buckets 1 and 2, buckets 2 and 3, buckets 3 and 4, etc, *Bucket_SJ* will join these cyclically. Then for any k-tuple of stars $s_1$, $s_2$, ..., $s_k$ that survive the join, *Bucket_SJ* will also check the distances of non-neighboring stars (e.g., check that dist($s_2$,$s_5$) = dist($p_2$, $p_5$) $\pm\epsilon$).

A filter join algorithm using matrix multiplication has been studied elsewhere [16] and sometimes helps. We leave it out here because this paper focusses on general constellation queries which come up in the next section.

## 3.6 Algorithm for Pure Constellation Queries

*3.6.1 Main Algorithm.* The *Pure Constellation Query* algorithm consists of five steps. It receives as input a pattern query $q$, a data partition $p$, element predicate $f_e$, similarity threshold $\theta$, and error-bound $\epsilon$.

i) Build a quadtree $qt$ on stars at each partition $p_i \in p$, according to 3.1. (This could be done once if there are to be many queries to process.)

ii) For each leaf $L$ of quadtree $qt$, find the relevant neighboring leaves based on the maximum length from the anchor in the query pattern to any other point of the query pattern.

iii) Each star $s$ in $L$ is a candidate to match the anchor node of the pattern.

iv) For each neighboring star $s'$ to $s$, determine whether its distance to $s$, $dist(s, s')$, matches within an additive $\epsilon$ the distance from the anchor point in the query to some other pattern element $p_i$. In that case, put $s'$ into bucket $i$. As noted above, a given star $s'$ could be put into several buckets.

v) Apply spatial bucket join to evaluate match stars in different buckets.

Once steps i) through v) have been executed, a set of collections of stars conforming to the pattern query have been detected.

# 4 GENERAL CONSTELLATION QUERIES

In this section, we show how to extend pure constellation queries to isotropic, and non-isotropic queries. Pure constellation queries specify both the properties and the distances of a pattern and require any matching sequence of stars to match the distances ± an error $\epsilon$. Essentially then, pure constellation queries find star collections that match a pattern, allowing rotation and translation.

*Isotropic constellation queries* find star collections that match a pattern, allowing rotation, translation, and linear (isotropic) scaling. Ignoring the error for a moment, a square pattern will scale to a large square in the sky. Formally, isotropic constellation queries allow $s_1, ..., s_k$ to match a pattern $p_1, ..., p_k$ if there is some scale factor $f$ such that, for all i and j between 1 and k, dist($s_i$, $s_j$) is within (f × dist($p_i$, $p_j$)) $\pm\epsilon$.

*Non-Isotropic constellation queries* find star collections that match a pattern, allowing rotation, translation, and skewed scaling. That is, some distances may be scaled differently than others. For example, again ignoring the error, a square pattern in which one side could expand or contract could match a trapezoid.

## 4.1 Isotropic Constellation Queries

Because the scale $f$ can take on any real value in isotropic constellation queries, the challenge is to find a sufficient discrete set of scale factors that will mimic trying the uncountable infinity of possible scale values of $f$ and to do so efficiently.

*Running Example (Isotropic Case):* An example will illustrate the issues and point towards a solution. Suppose the search is for an equilateral triangle consisting of pattern points $p_1, p_2, p_3$. Since this is an equilateral triangle, all intra-pattern distances are the same. So, we can set the maximum intra-pattern element distance to 1 without loss of generality. Suppose the additive error term $\epsilon$ has the value 2. If stars $s_1$, $s_2$, $s_3$ have the following pairwise distances dist($s_1$, $s_2$) = 8, dist($s_1$, $s_3$) = 12, dist($s_2$,$s_3$) = 12, then our algorithm might match $p_1$ and $p_2$ to $s_1$ and $s_2$. As a consequence, a naive algorithm might try to set the scale factor to dist($s_1$,$s_2$)/dist($p_1$,$p_2$), which is 8. In fact, however, a scale factor of 8 will not work because of the length 12 sides. Instead, we would want a scale factor $f$ of 10, even though no two stars are at a distance 10 of one another. So our algorithm has to discover $f$. We discuss how this happens below.

Our method starts by choosing two stars $s_1$ and $s_2$ and setting a candidate scale factor called *scalebasic* to dist($s_1$, $s_2$)/dist($p_1$, $p_2$). The following conditions express constraints on any final scale factor $f$ that maps $s_1$ and $s_2$ to $p_1$ and $p_2$. (As the pattern distances are all relative, in the sequel, we set dist($p_1$, $p_2$) to be 1, without loss of generality. This simplifies the notation.)

**isotropic f condition**: If dist($p_1$,$p_2$) = 1 and scalebasic = dist($s_1$, $s_2$)/dist($p_1$, $p_2$), let $f$ range from a minimum value $f_{min} = scalebasic - \epsilon$ to a maximum value $f_{max} = scalebasic + \epsilon$. The scale factor $f$ will have to lie between these two values.

**isotropic acceptable lengths**: For pattern points $p_i$, $p_j$, the acceptable length for a corresponding pair of stars $s_i$ and $s_j$ must fall between $(f_{min} \times dist(p_i, p_j)) - \epsilon$ and $(f_{max} \times dist(p_i, p_j)) + \epsilon$.

It will turn out that these two conditions will allow us to find all collections of stars that match the pattern collection up to any scale factor $f$ and additive error $\epsilon$ and will allow us to specify $f$.

*4.1.1 Isotropic Constellation Query Algorithm.* Our algorithm consists of six steps, starting from a pattern of k locations and an additive error bound $\epsilon$. The query may have other constraints too, such as a minimum or maximum possible scale factor. Other constraints may have to do with non-spatial attributes. Collectively, we call these constraints $C$.

i) find the most distant pair of pattern points in the query, which we will denote as $p_1$ and $p_2$, and set their distance, without loss of generality, to be 1.

ii) For every pair of stars that satisfy constraints $C$, call them $s_1$ and $s_2$, set a scale factor *scalebasic* to be sb = dist($s_1$,$s_2$)/dist($p_1$,$p_2$). Thus $s_1$ and $s_2$ will be candidate stars to match $p_1$ and $p_2$. Compute $f_{min}$ and $f_{max}$ according to the **isotropic f condition**.

iii) include a candidate star $s$ as a possible match to pattern point $p_j$ (for $j \neq 1$ or 2), if dist($s_1$, $s$) and dist($s_2$, $s$) conform to the **isotropic acceptable lengths** condition with respect to dist($p_1$, $p_j$) and dist($p_2$, $p_j$). All such stars $s$ would go into bucket $B_j$.

iv) Bucket $B_1$ consists of just $s_1$ and bucket $B_2$ consists of just $s_2$. All other buckets are calculated as in step iii.

v) Perform a k-1 Spatial join as in the pure constellation query algorithm.

vi) Post-processing: Any sequence of matching stars has to be validated concerning an error bound of $\epsilon$ as explained below.

*4.1.2 Theory and Explanation.* Given an additive error bound of $\epsilon$, our initial search effectively uses an error bound of $2\epsilon$. Our running example to find an equilateral triangle shows why this might be useful.

*Application of first steps to our Running Example (Isotropic Case)* In our example, the additive error term $\epsilon = 2$ and stars $s_1$, $s_2$, $s_3$ have the following pairwise distances dist($s_1$, $s_2$) = 8, dist($s_1$, $s_3$) = 12, dist($s_2$,$s_3$) = 12. Our algorithm might match $p_1$ and $p_2$ to $s_1$ and $s_2$ and set the variable *scalebasic* to $sb$ = dist($s_1$,$s_2$)/dist($p_1$,$p_2$), which is 8. Because $f_{min}$= 6 and $f_{max}$= 10 according to the **isotropic f condition**, the minimum and maximum side lengths can range from 4 to 12 based on the **isotropic acceptable lengths** condition. So, $s_3$ would also be considered a match up to step vi. The post-processing (to be described below) step shows in fact that $s_1$, $s_2$, and $s_3$ are a good match with an error bound of $\epsilon$ using a scale factor of 10. So steps i through v avoid false negatives.

Steps i through v may however find star combinations that are false positives. Step vi (post-processing) discards those. For true positives, step vi finds a possible scale factor $f$. Here is how. For each i, j such that $1 \le i, j \le k$, determine the minimum and maximum scale factor $minscale_{i,j}$ and $maxscale_{i,j}$ such that:

$$\text{dist}(s_i, s_j) = (minscale_{i,j} \times \text{dist}(p_i, p_j)) + \epsilon,$$
$$\text{dist}(s_i, s_j) = (maxscale_{i,j} \times dist(pi, pj)) - \epsilon.$$

Let the maximum of the minimum scales be denoted MaxMin and the minimum of the maximum scales be denoted MinMax. If MaxMin ≤ MinMax, then any value $v$ in the range between MaxMin and MinMax will be a satisfying scale factor provided it satisfies the **isotropic f condition**, i.e., that value $v$ is within $\epsilon$ of *scalebasic*. Otherwise there is no satisfying scale factor and the candidate solution is a false positive, so is discarded.

*Running Example Shows How Step vi Generates a Scale Factor, if At Least One Exists:* We are looking for an equilateral triangle (so dist($p_1$,$p_2$) = dist($p_2$,$p_3$) = dist($p_3$, $p_1$) = 1) and $\epsilon = 2$. If $s_1$, $s_2$, $s_3$ have the following pairwise distances dist($s_1$, $s_2$) = 8, dist($s_1$, $s_3$) = 12, dist($s_2$,$s_3$) = 12, then $minscale_{1,2} = 6$, $maxscale_{1,2} = 10$, $minscale_{1,3} = 10$, $maxscale_{1,3} = 14$, $minscale_{2,3} = 10$, and $maxscale_{2,3} = 14$. So MaxMin = 10 and MinMax = 10, so $f = 10$ would work as a scale factor and, in this example, it is the only such possibility.

*Variation of Running Example Showing Elimination of False Positives:* By contrast, if we are still looking for the same equilateral triangle and $\epsilon = 2$ and we have dist($s_1$,$s_2$) = 6, dist($s_2$,$s_3$) = 10, dist($s_3$, $s_1$) = 14, then in steps i through v, $s_1$, $s_2$, $s_3$ would correspond to $p_1$, $p_2$, $p_3$ when using a scalebasic value of 10. But step vi would yield $minscale_{1,2} = 4$, $maxscale_{1,2} = 8$, $minscale_{2,3} = 8$, $maxscale_{2,3} = 12$, $minscale_{1,3} = 12$, and $maxscale_{1,3} = 16$. So MaxMin = 12 and MinMax = 8, thus $MaxMin \le MinMax$ fails to hold. Therefore, the post-processing step would determine that s1, s2, s3 do not match the pattern.

We are now ready to formalize our guarantees. The first lemma shows that any matching solution will be found in the first five steps of the algorithm. Lemma 2 will show that the post-processing step keeps only the valid solutions.

Lemma 4.1 (no false negatives). *: Suppose that $p_1$ and $p_2$ are the most distant of the pattern points and dist($p_1$,$p_2$) = 1. Suppose further there is some matching sequence of stars $s_1$, $s_2$, $s_3$, ..., $s_k$ corresponding to a pattern $p_1$, $p_2$, ..., $p_k$ based on some scale factor $f$ and additive error tolerance $\epsilon$. Then $s_1$, $s_2$, $s_3$, ..., $s_k$ will be found to correspond to $p_1$, $p_2$, ..., $p_k$ using a scale factor of scalebasic = dist($s_1$,$s_2$)/dist($p_1$,$p_2$) using steps i to v above.*

Proof. (a) The scale factor $f$ must allow $s_1$ and $s_2$ to match $p_1$ and $p_2$. Because dist($s_1$,$s_2$)/dist($p_1$,$p_2$) = scalebasic, the construction of $f$ requires that (f × dist($p_1$,$p_2$))−$\epsilon$ ≤ dist($s_1$,$s_2$) = (scalebasic × dist($p_1$,$p_2$)) ≤ (f × dist($p_1$,$p_2$)) + $\epsilon$. Because dist($p_1$,$p_2$) = 1 by construction, this implies that $f$ must be constrained in step iii of the algorithm as follows: $scalebasic - \epsilon \le f \le scalebasic+\epsilon$. The first term is $f_{min}$ and the third term is $f_{max}$ of the **isotropic f condition**. Therefore $f$ lies between the $f_{min}$ and $f_{max}$ computed in step ii.

(b) Consider any distance dist($s_i$, $s_j$). Because $s_i$ and $s_j$ match $p_i$ and $p_j$ by assumption for scale factor $f$, (f × dist($p_i$, $p_j$) - $\epsilon$ ≤ dist($s_i$, $s_j$) ≤ (f × dist($p_i$, $p_j$)) + $\epsilon$. Because $f$ falls between $f_{min}$ and $f_{max}$ by (a), dist($s_i$,$s_j$) will fall within the **isotropic acceptable length** computed in step (iii). Because this holds for all i, j, the stars $s_1$, $s_2$, ... $s_k$ will be found to match $p_1$, ..., $p_k$ in steps i through v. QED □

Lemma 4.2 (No false positives). *Post-processing step vi as described in the paragraph above correctly determines whether S corresponds to P based on a $\epsilon$ tolerance.*

Proof. By construction, the minscale and maxscale values are the minimum and maximum possible scale factors for each pair i, j. Any single scale factor $f$ that lies between the minimum and maximum possible scale factors for all i,j will work provided it conforms to the **isotropic f condition**. If there is no such scale factor, then none can work. QED. □

Theorem 4.3 (Isotropic Constellation Query Theorem). *The isotropic constellation algorithm finds all matches of any arbitrary pattern for a given error tolerance $\epsilon$.*

Proof. Lemma 1 tells us that any sequence of stars that matches the pattern based on some scale factor $f$ will be found by steps (i) through (v). Lemma 2 tells us that any sequence of stars found by (i) through (v) and verified by step (vi) will be correct for the specified error tolerance $\epsilon$. QED. □

*4.1.3 Scale-dependent Error Tolerance for Isotropic Queries.* For some applications, it would be more natural for the error tolerance $\epsilon$ to increase monotonically with the scale factor. For example, the error bound $\epsilon$ might increase linearly with the scale factor (e.g., $e \times scale$ for $e < 1$). In such a case, steps (i) through (v) would set the $\epsilon$ to be $e \times f$ for the maximum value of $f$ which is $scalebasic+\epsilon$. So, steps (i) through (v) would set the $\epsilon$ to be $e\times(scalebasic+\epsilon)= (e\times scalebasic)+(e\times\epsilon)$. Rearranging terms, we get $\epsilon = e \times scalebasic/(1-e)$. For example, if $\epsilon = 0.1 \times scale$ and scalebasic = 10, then for purposes of steps (i) through (vi) set $\epsilon$ to

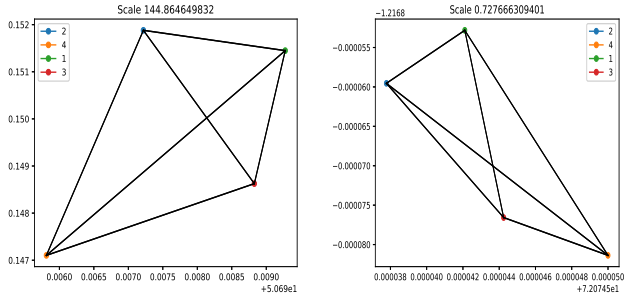
be $0.1 \times 10/0.9 = 1.11111...$, so twice $\epsilon$ is 2.2222.... The goal again is to avoid false negatives, even when we don't know what $f$ could be exactly.

For an example in which $\epsilon$ is not constant: suppose e = 0.2 and we are looking for an equilateral triangle and dist($s_1$, $s_2$) = 8, dist($s_1$, $s_3$) = 12, dist($s_2$,$s_3$) = 12. Then $minscale_{1,2} = 8/1.2 = 6.7$, $maxscale_{1,2} = 10$, $minscale_{1,3} = 10$, $maxscale_{1,3} = 15$, $minscale_{2,3} = 10$, and $maxscale_{2,3} = 15$. (To see how these calculations work, consider the computation of $minscale_{1,2}$. We know that $minscale_{1,2} \times (1+e) = 8$, so $minscale_{1,2} = 6\ 2/3$. Similarly, $maxscale_{2,3} \times (1-e) = 12$, so $maxscale_{2,3} = 12/0.8 = 15$.) Thus, MaxMin = 10 and MinMax = 10 and $s_1$, $s_2$, $s_3$ would match.

## 4.2 Non-isotropic Constellation Queries

Recall again that pure constellation queries find collections of stars that match a pattern, allowing rotation and translation. Isotropic constellation queries find collections of stars that match a pattern, allowing rotation, translation, and linear (isotropic) scaling. Non-isotropic queries find collection of stars that match a pattern, allowing rotation, translation, and non-linear (skewed) scaling. Formally, general non-isotropic constellation queries allow $s_1, \ldots, s_k$ to match a pattern $p_1, ..., p_k$ if there is some scale factor $f$ and some set of scale factor "stretching" modifiers $m_{i,j} > 1$ such that dist($s_i$, $s_j$) is within ($f \times$ dist($p_i$, $p_j$)) - $\epsilon$ and ($f \times m_{i,j} \times$ dist($p_i$, $p_j$)) + $\epsilon$. Thus $m_{i,j}$ is the maximum proportional "stretch" to the scale factor for the connection between i and j. For example, the distance between A and B could be $1.1 \times f$.

In contrast to the isotropic case, the scale factor for different pairs of pattern points may differ. That is, the isotropic case is a special case in which all $m_{i,j}$ equal 1. The challenge, again, is to find a sufficient discrete set that will mimic trying the infinity of possible scale factors and values of stretch (e.g., between 1 and $m_{i,j}$), and to do so efficiently.




(a) (b)

**Figure 4: Four points matching an Einstein cross query based on an isotropic formulation at scale 144.86 (a) and a non-isotropic solution with deformation (b)**

*Running Example (non-Isotropic Case):* Suppose the additive error $\epsilon = 0.2$. Suppose that the pattern to search for forms a square but the sides and diagonals can individually extend to a 10% more than the scale factor (i.e. for all i, j, $m_{i,j} = 1.1$). Then stars at locations (0,0), (0,11.2), (10,10) and (11.01,0) would meet the conditions.

For a second example, let's say the pattern forms a rectangle with the sides in a two-to-one ratio so that the longer sides can extend 10% ($m_{i,j} = 1.1$) but the shorter sides and diagonals can extend 30% ($m_{i,j} = 1.3$). Again, the additive error is $\epsilon = 0.2$. If we take a base side of (0,0) to (0,20), then the other base side could be from (10,0) to (10,21.5). The result is not even a rectangle. In this case, the longest side of the pattern is set to 1, so scalebasic is 20. However there could be different values of $f$, which could be less than or greater than scalebasic. For example, $f$ could be as large as 20.2 or $f$ could be as small as 18 (because $(1 \times 18 \times 1.1)+0.2= 20$. If $f = 20.2$, then the short sides of the pattern could correspond to star distances as great as length $(0.5 \times 20.2 \times 1.3)+0.2= 13.33$ or as short as $(0.5 \times 18)-0.2 = 8.8$.

Here are the general formulas. These directly generalize the isotropic case (for which $m_{i,j} = 1$). Again, start by choosing two stars s1 and s2 and set scalebasic to dist(s1, s2)/dist(p1, p2). Assume again without loss of generality that dist(p1,p2) = 1.

**non-isotropic f condition**: $f$ lies between a minimum value $f_{min} = (scalebasic - \epsilon)/m_{1,2}$ and a maximum value $f_{max} = scalebasic+\epsilon$.

**non-isotropic acceptable lengths**: The acceptable length for some other side i, j lies between $(f_{min} \times dist(pi, pj)) - \epsilon$ and $(f_{max} \times m_{i,j} \times dist(pi, pj))+\epsilon$.

*4.2.1 Non-Isotropic Constellation Query Algorithm.* Again, we use a six-steps algorithm, given an additive error bound $\epsilon$ along with stretch factors $m_{i,j}$.

i) Take as the base pair $p_i$ and $p_j$ if $dist(p_i, p_j))$ is the maximum over all i, j pairs. Without loss of generality, set i to 1 and j to 2 for consistency with the isotropic approach above. Set dist($p_1$, $p_2$) to 1. Compute $f_{min}$ and $f_{max}$ based on the **non-isotropic f condition**.

ii) For every pair of stars that satisfy some constraints $C$, call them $s_1$ and $s_2$. Set a scale factor *scalebasic* to be dist($s_1$,$s_2$)/dist($p_1$,$p_2$). This is exactly as in the isotropic case.

iii) For a candidate star s to correspond to pattern point $p_j$ (for $j \neq$ from 1 or 2), dist($s_1$, s) and dist($s_2$, s) should conform to **non-isotropic acceptable lengths** concerning dist($p_1$, $p_j$) and dist($p_2$, $p_j$). Such stars make up bucket $B_j$.

iv) (as for isotropic case) Bucket $B_1$ consists of just $s_1$ and bucket $B_2$ consists of just $s_2$. All other buckets may consist of zero or more stars based on each pattern point $p_j$ as calculated in step iii.

v) (as for isotropic case) Perform a Bucket Spatial join as in the pure constellation query algorithm.

vi) Post-processing: Any sequence of matching stars has to be validated concerning an error bound of $\epsilon$ and the stretch factors $m_{i,j}$. That means there has to be some $f$ such that for all sides i, j, $(f \times$ dist($p_i$,$p_j$)) $-\epsilon \leq$ dist($s_i$, $s_j$) $\leq (f \times m_{i,j} \times$ dist($p_i$,$p_j$)) $+\epsilon$, as we discuss in the next subsection. So the intention is the same as for the isotropic case, but the calculation changes because of the stretch factors.

*4.2.2 Theory.*

Lemma 4.4 (No false negatives). *: Suppose that $p_1$ and $p_2$ are the most distant of the pattern points and dist($p_1$,$p_2$) = 1. Suppose further there is some matching sequence of stars $s_1, s_2, s_3, ..., s_k$ corresponding to a pattern $p_1, p_2, ..., p_k$ based on some scale factor $f$, stretch factors $m_{i,j}$ for i, j between 1 and k inclusive, and additive error tolerance*

*$\epsilon$. Then $s_1$, $s_2$, $s_3$, ..., $s_k$ will be found to correspond to $p_1$, $p_2$, ..., $p_k$ with a scale factor scalebasic = dist($s_1$,$s_2$)/dist($p_1$,$p_2$) using steps i to v above.*

Proof. (a) The scale factor $f$ must allow $s_1$ and $s_2$ to match $p_1$ and $p_2$. Because dist($s_1$,$s_2$)/dist($p_1$,$p_2$) = scalebasic , this constrains $f$ to satisfy ($f\times$ dist($p_1$,$p_2$)) - $\epsilon \le$ dist($s_1$,$s_2$)= ($scalebasic\times$ dist($p_1$,$p_2$)) $\le$ ($f\times$ dist($p_1$,$p_2$) $\times m_{1,2}$) + $\epsilon$. Because dist($p_1$,$p_2$) = 1 by construction, this implies that $(scalebasic-\epsilon)/m_{1,2} \le$ f $\le scalebasic+\epsilon$. The first term is $f_{min}$ and the third term is $f_{max}$ of the **non-isotropic f condition**. Therefore $f$ lies within the limits of the $f_{min}$ and $f_{max}$ computed in step i.
(b) Consider any distance dist($s_i$, $s_j$). Because $s_i$ and $s_j$ match $p_i$ and $p_j$ by assumption for scale factor $f$, ($f\times$ dist($p_i$, $p_j$)) $-\epsilon \le$ dist($s_i$, $s_j$) $\le$ ($f \times m_{i,j} \times$ dist($p_i$, $p_j$)) $+\epsilon$. As $f$ falls between $f_{min}$ and $f_{max}$ by (a), dist($s_i$,$s_j$) will fall within the **non-isotropic acceptable length** computed in step (iii). Because this holds for all i, j, the stars $s_1$, $s_2$, $s_3$, ..., $s_k$ will be found. QED □

*4.2.3 Post-processing in the non-isotropic case.* Inspired from the post-processing step vi in the isotropic case, for each side i, j, we find the minimum possible scale factor for each i,j and then the maximum possible scale factor. Here is the minimum possible scale factor for pair i, j, $g_{i,j}(min) = min_g\{(g\times m_{i,j}\times$ dist($p_i$, $p_j$)) $+\epsilon \ge$ dist($s_i$,$s_j$) $\}$. So, $g_{i,j}(min)$ = (dist($s_i$,$s_j$) $-\epsilon$)/ ($m_{i,j}\times$ dist($p_i$, $p_j$)). Intuitively, the minimum possible scale factor is the one that stretches $p_i$ to $p_j$ as much as possible and uses the additive $\epsilon$ error bound to make dist($p_i$,$p_j$) reach dist($s_i$,$s_j$). Conversely (but without the stretch factor), $g_{i,j}(max) = max_g\{(g\times$ dist($p_i$, $p_j$)) $-\epsilon \le dist(s_i, s_j)\}$. So $g_{i,j}(max)$ = ($dist(s_i, s_j) + \epsilon$) / dist($p_i$, $p_j$).

Proceeding as for the isotropic case, let the maximum for all i, j of the $g_{i,j}(min)$ be denoted MaxMin and the minimum of the $g_{i,j}(max)$ be denoted MinMax. If MaxMin $\le$ MinMax, then any value $v$ in the range between MaxMin and MinMax will be a satisfying scale factor provided it satisfies the **non-isotropic f condition**, i.e., provided it lies between $(scalebasic - \epsilon)/m_{1,2}$ and $scalebasic+\epsilon$. Otherwise there is no satisfying scale factor.

*Applying Non-Isotropic Algorithm to Running Example:* The pattern forms a rectangle with the sides in a two-to-one ratio. The longer sides can extend 10% ($m_{i,j}$ = 1.1) but the shorter sides and diagonals can extend 30% ($m_{i,j}$ = 1.3). Again, the additive error is $\epsilon$ = 0.2. Now consider stars $s_1$, $s_2$, $s_3$, and $s_4$ with these positions: $s_1$: (0,0), $s_2$: (0,20), $s_3$: (10,0), $s_4$: (10,21.5). Let *scalebasic* be 20. By the **non-isotropic f condition**, $f_{min}$ = (20 − 0.2)/1.1 to $f_{max}$ =20+0.2. Ignoring the diagonals for simplicity, $m_{1,2}$ = 1.1, $g_{1,2}(max)$= 20.2/1= 20.2, $g_{1,2}(min)$= 19.8/(1 × 1.1)= 18.
$m_{1,3}$ = 1.3, so $g_{1,3}(max)$= 20.2. $g_{1,3}(min)$= 19.8/(1 × 1.3)= 15.23.
$m_{2,4}$ = 1.3, so $g_{2,4}(max)$= 20.2. $g_{2,4}(min)$= 15.23
$m_{3,4}$ = 1.1, so $g_{3,4}(max)$= 20.2. $g_{3,4}(min)$= 18.
So MinMax is 20.2 and MaxMin is 18. Because $f_{max}$ = 20.2 and $f_{min}$ = 18, the scale factor could be any value between 18 and 20.2 inclusive. For example, if $f$ is 18, then for dist(p3,p4) to correspond to dist(s3,s4), we must use the stretch factor: dist(s3, s4) = 11.5. dist(p3,p4) = 0.5. Using the stretch factor of 1.3, (0.5×18×1.3)+0.2= 11.9, so the stretch factor is more than enough to compensate for the small scale factor $f$ = 18.

Lemma 4.5 (No false positives). *Post-processing step vi as described in this subsection above correctly determines whether S corresponds to P based on an $\epsilon$ additive error and stretch factors.*

Proof. Similar to the isotropic case □

Theorem 4.6 (Non-Isotropic Constellation Query Theorem). *The non-isotropic constellation algorithm finds all matches of any arbitrary pattern for a given error tolerance $\epsilon$ and stretch factors $m_{i,j}$ for all i, j in the pattern.*

Proof. Lemma 3 tells us that for all scale factors $f$ any sequence of stars that matches the pattern based on $f$ will be found by steps (i) through (v). Lemma 4 tells us that any sequence of stars found by (i) through (v) and verified by step (vi) will be correct for the specified error tolerance $\epsilon$ and stretch factors $m_{i,j}$. QED. □

*4.2.4 Scale-dependent Error Tolerance.* As observed for the isotropic case, it is sometimes more natural for an application that the error tolerance $\epsilon$ increases monotonically with the scale factor. As before, suppose that the error bound $\epsilon$ increases linearly with the scale factor (e.g., $e \times scale$ for $e < 1$). Because the maximum scale factor based on the **non-isotropic f condition** is $f_{max}$, steps (i) through (v) would set the $\epsilon$ to be $e \times f_{max}$=$e\times(scalebasic+\epsilon)$
= ($e\times scalebasic$)+($e\times\epsilon$). So $\epsilon = e\times scalebasic/(1-e)$, just as in the isotropic case.

*4.2.5 Time Complexity of Our Algorithms.* Assuming we have all pairwise stars indexed by their distance, then given a starting star, we can form the buckets in time proportional to the number $k$ of points in the pattern query times the number of points at those distances +/- the additive error. So if there are a fraction $g$ of the $S$ stars that are at that distance within the error bound, then the time is $k \times g \times S$. And the time to do the join, while in the worst case can be $(g \times S)^{(k-1)}$ is in practice roughly proportional to $g \times S$. Thus, the time is proportional to $O(S \times (g \times S)^{(k-1)})$ in the worst case and $O(S \times k \times g \times S)$ in our experiments.

For isotropic queries, for each star, we must choose all possible neighbors and then the query takes about the same amount of time as in the pure case. So, $O(S^2 \times (g \times S)^k)$ in the worst case and $O(S^2 \times k \times g \times S)$ practically.

For the non-isotropic queries, the joins will encounter many more pairs, so, the time complexity will be closest to the worst case, i.e. $O(S^2 \times (g \times S)^k)$.

## 5 EXPERIMENTAL EVALUATION

This section first presents the experimental setup. Next, it assesses the different components of our implementation for point pattern search, to evaluate the scalability and feasibility of our algorithms.

### 5.1 Set Up

*5.1.1 Dataset Configuration.* The experiments focus on the Einstein cross point pattern and are based on an astronomy catalog dataset obtained from the Sloan Digital Sky Survey (SDSS), as well as synthetic datasets. The SDSS catalog, published as part of the data release DR14, was downloaded from the project website link ($http://skyserver.sdss.org/CasJobs/$).

We downloaded a projection of the dataset including attributes $(objID, ra, dec, u, g, r, i, z, err_u, err_g, err_r, err_i, err_z)$. The extracted dataset has a size of 189 GB containing around 1.3 billion sky objects. The query to obtain this dataset was this one:

```
Select objID, ra, dec, u, g, r, i, z,err_u, err_g,err_r,err_i,
err_z
From PhotoObjAll into MyTable
```

From the downloaded dataset, some subsets were extracted to produce datasets of different sizes. Additionally, to simulate very dense regions of the sky, we built synthetic datasets with 1000, 5000, 10000, 15000, and 20000 stars. The synthetic dataset includes millions of scaled solutions in a very dense region. Each solution matches a base pattern using a scale factor chosen uniformly within an interval $s = [1.00000001, 1.0000009]$.

*5.1.2 Calibration.* We calibrated point pattern search techniques using the SDSS dataset described above. The procedure aimed at finding the *Einstein Cross* in the astronomy catalog, using the proposed techniques for point pattern search. The techniques succeeded in spotting the structures we had seeded among billions of candidates.

*5.1.3 Computing Environment.* The point pattern search processing is implemented as an Apache Spark dataflow running on a shared nothing cluster. The Petrus.lncc.br cluster is composed of 10 DELL PE R530 servers running CENTOS v. 7.2, kernel version 3.10.0327.13.1.$el7.x$86_64. Each cluster node includes two Intel Xeon E5-2630 V3 2.4GHz processors, with twelve cores each, 96 GB of RAM, 20MB cache and two terabytes of hard disk. We are running Hadoop/HDFS v2.7.3 [18], Spark v2.2, and Python v2.6.6. Spark was configured with number of executors varying from 16 to 128 executors each running with 5GB of RAM and one core. The driver module was configured with 80GB of RAM.

The first experiment considers a subset of the SDSS catalog with 20 million sky objects, obtained randomly from Query I results. The second dataset considers dense synthetic data, as described above. The datasets were ingested in HDFS sharded into partitions as described in 3.1. In order to evaluate the effect of maximum scale values, We loaded three replicas of the 20 million sky objects dataset. The replicas include the same primary partitions extended with neighbors at most: 5, 10 and 15 arcsecs from the primary partition borders, respectively.

The point pattern search dataflow is depicted in Figure 5. The first transformation *BT* applies a *MapPartition* to the input catalog dataset to build the quadtree. The result is consumed by a *Map* function running PC. It implements all query steps, up to, and including, the k-1 spatial join. The output dataframe contains for each star the set of filled buckets. Next, the tuples with filled buckets are input to another *Map* function BJ that implements a query execution plan comprising the *Spatial Bucket join* and the final post-processing filter.

## 5.2 General Algorithm Evaluation

In this first experiment, we use the 20 million sky objects dataset and evaluate the three search algorithms: pure, isotropic and non-isotropic. Figure 6 depicts the number of retrieved solutions when we vary the parameters: size of neighboring partition $\{5, 10, 15\}$ and additive error $\{0.1, 0.2, 0.3, 0.4\}$ and stretch factors for non-isotropic between 1 and 1.3. As expected, the flexibility introduced by isotropic and non-isotropic queries considerably increases the number of matches.

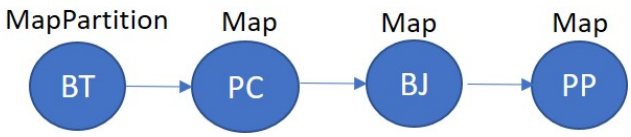



**Figure 5: Constellation Query Dataflow**

## 5.3 Increasing the Query Size

The next experiment tests the effect of increasing the query size from 4 to 7 elements, as depicted in Figure 7. We compare the number of solutions obtained by the three algorithms when running with a dataset of 20 million stars. The angular span is 15 arcsecs and the additive error is 0.4. All measurements are the average of 10 runs.

The increase in the query size is obtained by randomly adding stars to the query along the circumference of a circle corresponding to an Einstein cross. As we can observe, the addition of stars increases the constraints to be applied to form shapes. For 6 elements, pure finds only 1 solution while isotropic matches with 87 shapes. Finally, at size 7, distance constraints lead to zero pure solutions and 13 isotropic. Conversely, non-isotropic algorithm presents a different behavior. As we can see, the number of solutions increases with query size, growing up to two million solutions for query size 7. At higher query sizes the time difference among the three algorithms grows too.

Finally, we investigated the behavior of constellation queries with query sizes 15 and 200, respectively. We generated two new point queries by randomly selecting points within the circumference of a circle corresponding to an Einstein cross. For 15 and 200 query elements, Spark runs out of memory, when executing with $\epsilon > 0.2$, due to the increasing number of solution candidates. In order to maintain the number of Spark executors and the amount of allocated memory, as the ones used for running queries from 4 to 7 elements, we reduced the additive factor to $\epsilon = 0.1$. Under this additive factor, no solution is found for either pure and isotropic, but the queries run to their end. The elapsed time for both executions, considering an average of 10 runs each, were: for 15 query elements - pure: 157.59 s, isotropic: 549.69 s, and for 200 query elements - pure: 164.85 s, isotropic: 935.26 s. We observe that the behavior is very similar to smaller query sizes. The predominant cost appears when testing each new star object for matching with anchors. In the case of isotropic this test is applied twice, once for each anchor sky object.

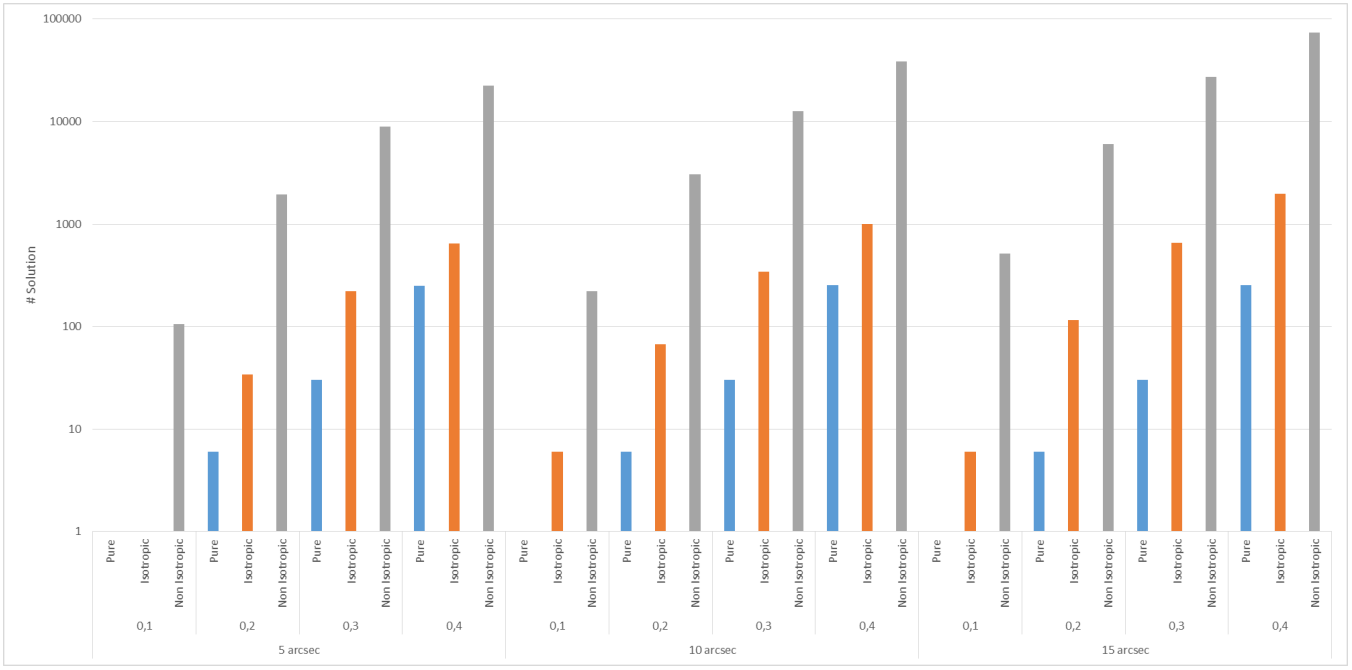



**Figure 6: Number of matches for different query types, different error bounds, and different angular spans across the sky. The number of matching solutions increases exponentially with the error bound, increases with the span of the sky (but how much depends on the query type) and is of course consistently larger for non-isotropic queries than isotropic queries and for isotropic queries than for pure ones. Stretch factors for non-isotropic are allowed to range from 1 to 1.3.**

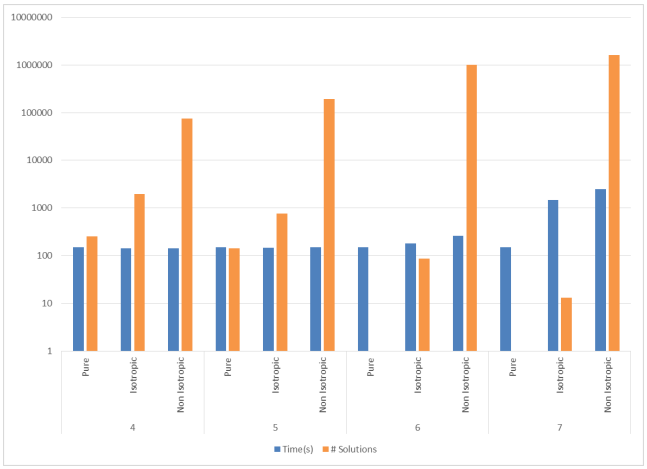



**Figure 7: Number of solutions and elapsed time (in seconds) on 128 executers for pure, isotropic, and non-isotropic queries for different numbers of query points. Non-isotropic stretch factors may range from 1 to 1.3.**

## 5.4 Scalability of Pure Constellation as a function of dataset density

We investigated Pure scale-up adopting the set of dense datasets (see section 5.1.1) with error = $4.4x10^{-6}$. The configuration produced solutions of size: 0, 730, 8570, 9880, and 43520. The run with the 1000 star dataset produced zero solutions, which shows the effect of different error bounds especially for Pure. Apart from the runs with the 15,000 stars dataset, the time increased monotonically with the number of solutions. That is, the filtering away of non-solutions cost little time. Figure 8 shows the results, where time corresponds to the elapsed-time in seconds of the parallel execution.

## 5.5 Scalability as a function of Additive error

Figure 9 shows the effect of increasing the additive error in the number of solutions produced by the isotropic algorithm. We consider the additive error as a fraction of the scale, $\epsilon = \frac{e*scalebasic}{1-e}$ and run the experiments with neighborhood extensions of 15 arcsecs. Each point reflects the average of 10 runs.

Large additive errors vastly increase the number of solutions. Indeed, when the lower bound for isotropic acceptable length becomes negative, all distances not greater than the upper bound are accepted, even tiny distances become candidate matches. The number of solutions remains relatively modest however because few candidate solutions pass the post-processing validation step (vi).

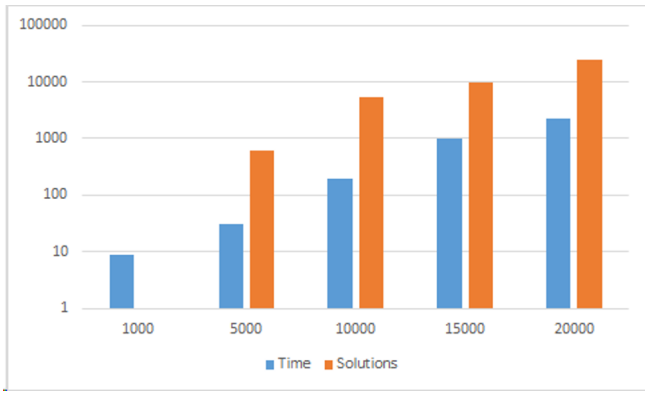



**Figure 8: Pure Constellation Scale-up as a function of the density of solutions: time (in seconds) is closely related to the number of solutions. This implies that filtering away non-solutions costs little for pure constellation queries.**

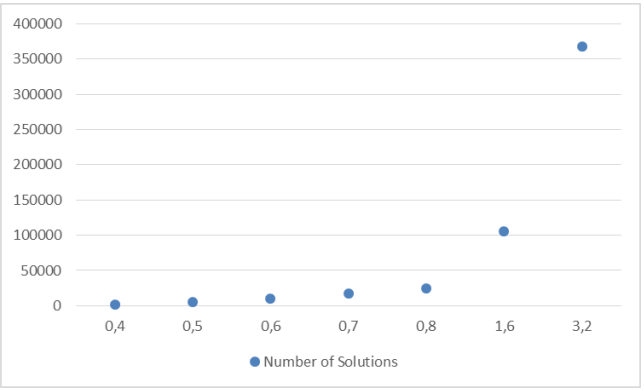



**Figure 9: Number of Solutions vs Additive Error for isotropic constellation queries. Again, we see an exponential relationship.**

The following results, depicted in Figure10, considers additive errors of 0.4, 0.8 and 1.6. The execution time is broken down by the execution cost of dataflow functions, as illustrated in Figure 5. In the plot, bars show the average of the elapsed-time among eight working nodes. The dataflow function bars are presented in sequence: BT (build time of the quadtree), PC (build the buckets) and BJ (spatial bucket join), the post processing cost is embedded in BJ.

The experiment leads to two observations: First, the cost of building the quadtree is significant, but less than the cost to form the buckets. Second, the cost of running the spatial bucket join is negligible, at least when the selectivity of the distance constraints causes the sizes of the buckets to be small. We expect such high selectivity for any situation in which the errors are small. Thus, the choice of the spatial join algorithm to be applied has little impact on the overall execution time. The last filtering step that applies global constraints also costs little.

So in scenarios like this, most time will be spent building buckets. The time complexity is O($k \cdot H \cdot N \cdot S$), where k is the number of query points; H is the number of leaf nodes ($4 * *treeheight$); N is the maximum number of stars in a leaf node so $H \cdot N$ is the number of candidate anchors; S is the number of neighbors which must be tested to determine which go into each bucket.

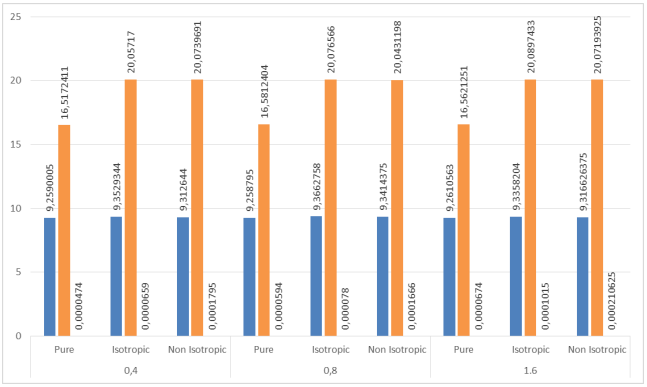



**Figure 10: Scalability of Algorithms vs Additive Error: the blue bar is the time to build the quadtree (roughly the same for all), the orange line is the time to fill the buckets of candidates with respect to an anchor star (or pair of stars in the case of isotropic and non-isotropic queries), the (mostly invisible) green bar shows the time to perform the bucket join. Non-isotropic stretch factors may range from 1 to 1.3.**

## 5.6 Scalability as a function of the number of allocated nodes

Figure 11 shows the behavior of the algorithms when the number of allocated nodes increases from 16 to 128 nodes. The 20 million stars dataset was used and we depict the average of 10 runs. The non-isotropic algorithm shares a single deformation matrix with stretching factors $m_{i,j} = \{m_{1,2} = 1.2, m_{1,4} = 1.1, m_{2,3} = 1.3\}$ and 1 otherwise. The performance of the Pure algorithm improves linearly from 16 to 64 nodes, but flattens out at 128 nodes. The behavior of isotropic and non-isotropic is similar, as expected, because most of the time is spent filling in buckets. These results show that isotropic and non-isotropic may produce more solutions than Pure with only a small increase in time.

## 5.7 Comparison to Point Set Registration

Because the local point set registration algorithms, such as ICP [3] and RPM [5], take advantage of the fact that two images being matched tend to have points that are geometrically close, one would expect them to be faster than our constellation queries algorithms. Surprisingly, that is not the case as Figure 12 shows. In this experiment, algorithms ICP and RPM cannot run with even 250 thousand stars, we used datasets of size: 100, 900, 8100 and 87500 stars. We

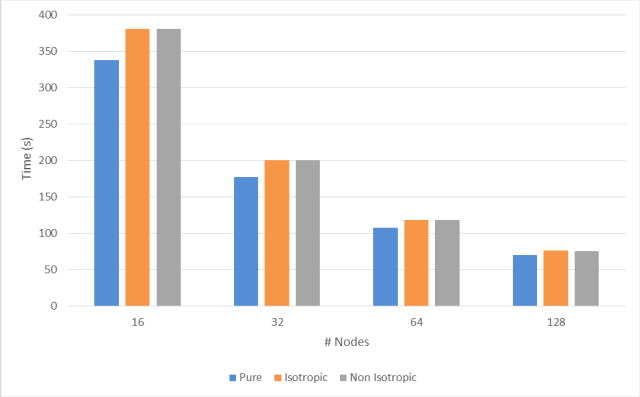



**Figure 11: Scalability of Algorithms: the blue bar represents the time for the pure algorithm, the orange bar is the time for the isotropic, very close to non-isotropic, in gray.**

run Pure, Isotropic and Non-Isotropic algorithms, as well as ICP and RPM in single node mode. As Figure 12 shows, constellation algorithms are always faster than local point set registration. With a dataset of 87500 stars, constellation algorithms are approximately nine times faster than ICP and RPM.

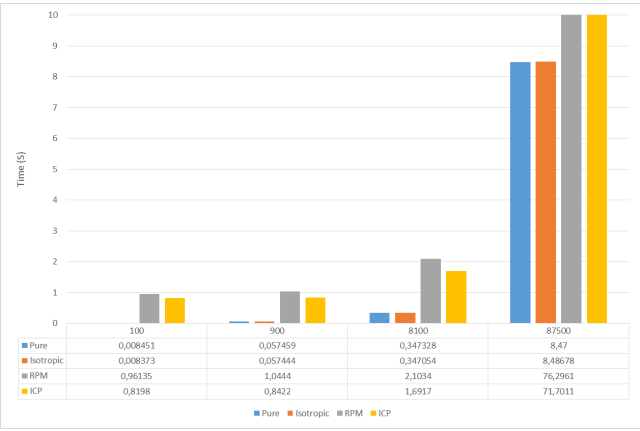



**Figure 12: constellation queries (and even isotropic and non-isotropic constellation queries) are faster than ICP and than RPM on the experiments described in the text.**

## 6 RELATED WORK

Finding collections of objects having some metric relationship of interest to one another has many applications. The problem has different names depending on the discipline, including *Object Identification* [19], *Graph Queries* [22], *Point Set Registration* [12, 20] and, in general, *Pattern Recognition* [4].

The main motivation of point set registration comes from image processing in which a scan digitizes the same object smoothly moving around the object, leading to sets of points slightly displaced in two or three dimensions. The goal of point set registration is to align the points of different images together. The result of a point set registration process is a single image with the various input images put together. Each image is expected to have some noisy data due to problems in scan capture and may also capture only part of the data. Nevertheless, every input image has approximately the same set of points.

The main local approach to Point Set Registration is the Iterative Closest Point algorithm [3, 20]. The algorithm finds for each point in one image the closest point in the target image. The results may miss the correct alignment if the displacement between the points is larger compared with the average distance among points.

By contrast, for constellation queries, the absolute locations of pattern points are not important. Only pairwise distances are important. So the techniques used by algorithms like Iterative closest Point do not apply, because they depend on the absolute locations of pattern points. On the hand, global Point Set Registration algorithms such as 4PCS [1, 13] do alignment without regard to the absolute locations of pattern points and look at collections of 4 points in the target database. So this means that if the target has $D$ points, then the time would be approximately $D^4$. Moreover, though the 4PCS algorithm performs isotropic-style queries, 4PCS requires a query pattern to be a sizeable fraction of the database (as stated on page 7 of the [13]). We are interested in small patterns matching to large databases.

Pattern recognition research focuses on identifying patterns and regularities in data [4]. Graphs are commonly used in pattern recognition due to their flexibility in representing structural geometric and relational descriptions for concepts, such as pixels, predicates, and objects [6, 8]. In this way, problems are commonly posed as a graph query problem, such as subgraph search, shortest-path query, reachability verification, and pattern match.

In a subgraph query, a query is a connected set of nodes and edges (which may or may not be labeled). A match is a (usually non-induced) subgraph of a large graph that is isomorphic to the query. While the literature in that field is vast [[21],[22], [7]], the problem is fundamentally different from ours, since there is no notion of geometrical distance (so data structures like quadtrees are useless) and certainly no distance notion of scale or stretching. Nevertheless, our queries and graph motifs often tend to be most interesting when they consist of a small number of points.

Another area of intense work related to constellation query resolution is spatial joins. In [11], alternatives for spatial joins are discussed according to the availability of indexes prior to query evaluation. The seeded tree join [9] uses an existing R-tree index to build a second one on the join relation. In the case in which no index exists (our scenario) the spatial hash-join algorithm [10] hashes one of the relations into a pre-determined number of buckets and probes it with the other relation, as is done in a relational hash-join. This setting is similar to the implementation of bucket-join in our algorithms. The candidate stars in bucket-join are mapped to buckets corresponding to the distance classes from the anchor and then joined together to apply the global distance constraints. However, as has been shown in section 5.5, the cost of running

bucket join is negligible compared to the costs of BT (building the quadtree) and PC (filling the buckets). So, the particular spatial join algorithm has little impact.

Other relevant work is querying spatial structures (SS)[15]. That work tries to give information about the relative placement of objects (e.g. object A is within object B or object C is to the right of object D). Their problem is more difficult in one sense because their objects have dimensionality. Combining dimensionality with the isotropic and non-isotropic constellation queries is a promising area for future work.

## 7 CONCLUSION

In this paper, we introduce *General constellation queries*, which are geometrical queries against a large point-set. We illustrate the application of Constellation Queries in astronomy. The objective of a constellation query is to find sequences of dataset elements that form a spatial structure geometrically similar to the one given by the query pattern.

The main novelty of our work is the extension of pure constellation queries (in which a solution must match the query up to an additive error) to isotropic and non-isotropic queries which allow an uncountably infinite number of scale factors and (for non-isotropic) an uncountably infinite number of stretch factors. Our discrete algorithm provably finds all sequences in the dataset at every scale that matches any of these constellation queries within an error tolerance.

In order to reduce the potential $n^k$ comparisons,we use a quadtree to filter out distant neighbors. The quadtree data structure is very helpful in two or three dimensions, but would not work well with ten dimensions or more. In that case, we would use a near neighbor data structure, such as [2], which would give, for each data point x, the other data points y1, y2, .... yk that are reasonably close to x (to be of interest in any query response) and their distances. Then each data point x could be a candidate anchor.

Our parallel experiments on a subset of the SDSS dataset show that the filtering techniques based on quadtrees are enormously beneficial. Moreover, the algorithms scale roughly linearly with the number of processing cores.

To the best of our knowledge, this is the first work to investigate spatial queries with free scaling factors (isotropic) and spatial queries with free scaling and stretch factors (non-isotropic). There are numerous opportunities for future work, especially in optimization for higher dimensions and similar queries on geometric objects with non-zero dimensionality.

## ACKNOWLEDGMENTS

This research is partially funded by EU H2020 Program and MCTI/RNP-Brazil(HPC4e Project - grant agreement number 689772),CNPq (PQ 310109/2015-9),CAPES, INRIA(SciDISC associated team) and the Computational Biology Institute (www.ibc-montpellier.fr), INRIA international chair, U.S. National Science Foundation MCB-1158273, IOS-1139362 and MCB-1412232. This support is greatly appreciated.

## REFERENCES

[1] Dror Aiger, Niloy J. Mitra, and Daniel Cohen-Or. 2008. 4pointss Congruent Sets for Robust Pairwise Surface Registration. *ACM Transactions on Graphics* 27, 3, Article 85 (Aug. 2008), 10 pages. DOI: http://dx.doi.org/10.1145/1360612.1360684

[2] Kristin P. Bennett, Usama Fayyad, and Dan Geiger. 1999. Density-based Indexing for Approximate Nearest-neighbor Queries. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*. ACM, New York, NY, USA, 233–243. DOI: http://dx.doi.org/10.1145/312129.312236

[3] Paul J. Besl and Neil D. McKay. 1992. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 14, 2 (Feb. 1992), 239–256. DOI: http://dx.doi.org/10.1109/34.121791

[4] Christopher Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, NY, USA.

[5] Haili Chui and Anand Rangarajan. 2003. A New Point Matching Algorithm for Non-rigid Registration. *Comput. Vis. Image Underst.* 89, 2-3 (Feb. 2003), 114–141. DOI: http://dx.doi.org/10.1016/S1077-3142(03)00009-2

[6] D. Conte, P. Foggia, C. Sansone, and M. Vento. 2004. THIRTY YEARS OF GRAPH MATCHING IN PATTERN RECOGNITION. *International Journal of Pattern Recognition and Artificial Intelligence* 18, 3 (May 2004). DOI: http://dx.doi.org/https://doi.org/10.1142/S0218001404003228

[7] Rosalba Giugno and Dennis Shasha. 2002. *GraphGrep: A fast and universal method for querying graphs* (2 ed.). Vol. 16. 112–115.

[8] J.M. Jolion. 2001. Graph Matching: What Are We Really Talking About?. In *3rd Int'l Association in Pattern Recognition - Tc15 Workshop on Graph based Representations*. Ischia, Italy.

[9] Ming-Ling Lo and Chinya V. Ravishankar. 1994. Spatial Joins Using Seeded Trees. *SIGMOD Rec.* 23, 2 (May 1994), 209–220. DOI: http://dx.doi.org/10.1145/191843.191881

[10] Ming-Ling Lo and Chinya V. Ravishankar. 1996. Spatial Hash-joins. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*. ACM, New York, NY, USA, 247–258. DOI: http://dx.doi.org/10.1145/233269.233337

[11] Nikos Mamoulis and Dimitris Papadias. 1999. Integration of Spatial Join Algorithms for Processing Multiple Inputs. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*. ACM, New York, NY, USA, 1–12. DOI: http://dx.doi.org/10.1145/304182.304183

[12] Alexander Marinov and Nadezhda Zlateva. 2010. ICP Algorithm for Alignment of Stars from Astronomical Photographic Images. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies (CompSysTech '10)*. ACM, New York, NY, USA, 485–489. DOI: http://dx.doi.org/10.1145/1839379.1839466

[13] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. 2014. Super 4PCS Fast Global Pointcloud Registration via Smart Indexing. *Computer Graphics Forum* 33, 5 (Aug. 2014), 205–215. DOI: http://dx.doi.org/10.1111/cgf.12446

[14] M. Muralikrishna and David J. DeWitt. 1988. Equi-depth Multidimensional Histograms. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD '88)*. ACM, New York, NY, USA, 28–36. DOI: http://dx.doi.org/10.1145/50202.50205

[15] Dimitris Papadias, Nikos Mamoulis, and Vasilis Delis. 1998. Algorithms for Querying by Spatial Structure. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 546–557. http://dl.acm.org/citation.cfm?id=645924.671163

[16] Fábio Porto, Amir Khatibi, João R. Nobre, Eduardo S. Ogasawara, Patrick Valduriez, and Dennis E. Shasha. 2017. Constellation Queries over Big Data. *CoRR* abs/1703.02638 (2017). http://arxiv.org/abs/1703.02638

[17] Hanan Samet. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10)*. IEEE Computer Society, Washington, DC, USA, 1–10. DOI: http://dx.doi.org/10.1109/MSST.2010.5496972

[19] Parag Singla and Pedro Domingos. 2005. *Object Identification with Attribute-Mediated Dependences*. Springer Berlin Heidelberg, Berlin, Heidelberg, 297–308. DOI: http://dx.doi.org/10.1007/11564126_31

[20] W. Zhou W. Sun and M. Yang. 2011. Medical Image Registration Using Thin-Plate Spline for Automatically Detecting and Matching of Point Sets. In *Proceedings of the 5th International Conference on Bioinformatics and Biomedical Engineering*. 1–4. DOI: http://dx.doi.org/doi:10.1109/icbbe.2011.5780355

[21] Lei Zou, Lei Chen, and M. Tamer Özsu. 2009. Distance-join: Pattern Match Query in a Large Graph Database. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 886–897. DOI: http://dx.doi.org/10.14778/1687627.1687727

[22] Lei Zou, Lei Chen, M. Tamer Özsu, and Dongyan Zhao. 2012. Answering Pattern Match Queries in Large Graph Databases via Graph Embedding. *The VLDB Journal* 21, 1 (Feb. 2012), 97–120. DOI: http://dx.doi.org/10.1007/s00778-011-0238-6