COMPASS: Cardinal Orientation Manipulation and Pattern-Aware Spatial Search

Kent O'Sullivan*
osullik@umd.edu
University of Maryland
USA

Nicole R. Schneider*
nsch@umd.edu
University of Maryland
USA

Hanan Samet
hjs@cs.umd.edu
University of Maryland
USA

ABSTRACT

The *Spatial Pattern Matching* paradigm offers a promising direction for searching with incomplete or imperfect information, but it is badly constrained by dependence on graph-based representations and computationally intensive search algorithms like subgraph matching and constraint satisfaction. To address these limitations, we present *COMPASS*, a suite of data structures and algorithms that enable pattern-based search by encoding the directional relationships between objects in abstracted matrix representations rather than graph structures. We provide a series of recursive search algorithms that leverage our matrix representations to enable spatial search queries with directional constraints, which are typically too dense for previous graph-based approaches. Our search methods find matches even when the query pattern is not aligned to the global coordinate system, resulting in perfect recall in our evaluation. Computationally, our search methods scale with the number of query objects times the number of database objects squared in the worst case, which is significantly better than previous methods. Our empirical measurements show that the performance is typically even better, approaching logarithmic in the number of query terms.

CCS CONCEPTS

• **Information systems** → **Geographic information systems**; *Query representation*; *Incomplete data*; • **Computing methodologies** → *Spatial and physical reasoning*.

KEYWORDS

Spatial Pattern Matching, Pictorial Querying, Searching Geospatial Objects

ACM Reference Format:
Kent O'Sullivan, Nicole R. Schneider, and Hanan Samet. 2023. COMPASS: Cardinal Orientation Manipulation and Pattern-Aware Spatial Search . In *2nd ACM SIGSPATIAL International Workshop on Searching and Mining Large Collections of Geospatial Data (GeoSearch '23), November 13, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3615890.3628537

*Both authors contributed equally to this research.

GeoSearch '23, November 13, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0352-2/23/11.
https://doi.org/10.1145/3615890.3628537

1 INTRODUCTION

Modern spatial search methods are derived from traditional search techniques that were not designed with spatial search constraints in mind. Keyword querying, which is a natural constraint in relational data, can be applied to narrow spatial regions, but the full richness of the spatial information is lost [2, 13, 18, 26]. Likewise, simple metric or topological constraints like "near", "within x miles of", etc. cannot capture the inherent complexity of spatial data, where every entity relates to every other entity through its position in a region [1, 3, 4]. Current methods for searching spatial data require encoding schemes that collapse rich spatial information down to basic relationships that fail to capture the most important aspect of the data - how multiple spatial entities relate to each other directionally.

Under this paradigm, natural spatial pattern queries like "Where is there a pond, a statue, and a bridge arranged in a triangle?" are impossible to answer. Previous work in spatial pattern matching, which handles exactly this type of multi-entity directional query, requires encoding the data in densely connected graphs or multigraphs, or segmenting space with respect to every object individually [5, 11, 11, 12]. Because of their underlying encoding schemes, these approaches are intractable for any reasonable number of objects and query constraints. This gap between the natural spatial expressiveness of directional queries and the intractibility of resolving them for nontrivially-sized collections of spatial data is the reason modern search engines do not support them.

We present ***C**ardinal **O**rientation **M**anipulation and **P**attern-**A**ware **S**patial **S**earch*, (*COMPASS*)[1], a suite of data structures and algorithms that begin to address the scalability limitations of directional spatial pattern matching. To accomplish this, we encode the directional relationships between objects in abstracted matrix representations rather than the graph-based structures that most current approaches use. We enable search over these matrix representations through a series of recursive spatial pattern-matching algorithms that traverse the objects, searching *if* there exists at least one set of objects that satisfies the constraints (rather than retreiving all sets that do). This simplification of the problem allows us to achieve worst case complexity of $O(n^2)$ in the number of objects, a significant improvement over previous methods. By simplifying the problem, we enable users to search over geospatial objects using directional constraints at scale, which has not been achieved previously.

Further, when objects are grouped semantically at a 'location' the *if* question can be used to retrieve any locations that contain at least one set of objects matching the query constraints. For instance, "Find all parks with a pond, a statue, and a bridge arranged in a

[1] https://github.com/osullik/COMPASS_GEOSEARCH

triangle" becomes answerable. To encourage evaluation of future methods against our initial matrix-based solution, we provide a data generator that produces location and object databases and pictorially-specified directional spatial queries to run against them.

We begin the rest of this paper by describing relevant background on spatial search in section 2. Sections 3 and 4 outline our proposed solution. We evaluate each algorithm's theoretical complexity and efficiency on the datasets we generate in sections 5 and 6 and briefly summarize relevant related work in section 7 before concluding.

2 SPATIAL SEARCH

Before introducing the data structures and algorithms we use to accomplish spatial search, we define the spatial entities and relations discussed throughout the paper and briefly describe how spatial search queries are specified as input to the spatial search task.

2.1 Spatial Entities

Spatial entities can be classed into three main types: *Points*, which consist of an (x, y) coordinate pair in Cartesian space, *Lines*, which represent the shortest path between two points, and *Regions*, which represent the area inside a polyline joining several points.

2.2 Spatial Relations Between Entities

Points, Lines and *Regions* relate to each other spatially through the following types of relationships [1, 3, 4]:

- *Topological relations* that describe how regions, lines, and points interact (intersect, contain, touch, etc),
- *Metric relations* that describe the distances between objects (ten miles, near, far etc.), and
- *Directional relations* that describe how objects are positioned relatively in space (north, left, behind etc.).

Topological relations are typically binary (e.g., a road intersects another road, or it does not). *Metric* relations can be quickly binned from continuous to discrete values (e.g., a train station is near or far from a house). *Directional* relations are typically complex (e.g., a sign, statue, and fountain are arranged in a triangle), making them highly descriptive of the world, at the cost of being challenging to represent. The focus of this paper is directional relations between points.

2.3 Encoding Directional Relations

There are two dominant approaches to encoding directional relations, *space segmentation* and *link encoding* [6, 7]. *Space segmentation* divides the space around one or many reference point(s), creating a shared list of 'objects that are <direction>' of that point. Generalizing space segmentation to capture relations between encoded objects requires each encoded object to act as a reference point, storing the direction of all other objects with respect to it. *Link encoding* creates a *Graph* or *Qualitative Constraint Network* (QCN) to explicitly encode pairwise directional relations between objects as links. Encoding directional relations requires a fully connected graph or multigraph to capture all the relevant relations. Most encoding schemes assume a global coordinate system (e.g., cardinal directions); those that do not are *Cardinality Invariant*.

2.4 Spatial Query Specification

Spatial queries consist of one or more *constraints*. Constraints are either *attributes* required of objects in the result set, or spatial relations required between objects in the result set. *Attribute constraints* are typically *keywords* that describe the objects of interest like 'hospital', 'restaurant', or 'school'. *Relation constraints* are directional relationships like 'school West of Pond'. Individually encoding these query constraints becomes cumbersome as the number of query objects grows. A more natural approach to encoding directional query constraints is to do so implicitly, with a picture or sketch map [18]. A *pictorial query* accomplishes this using a canvas of points representing query objects configured in a pattern of interest, which implicitly captures the pairwise directional constraints between the objects.

2.5 Spatial Search Task

Spatial search is the task of retrieving entities from a database that meet the constraints specified by a spatial query. More specifically, spatial pattern matching is a type of *spatial search* that is typically defined as a graph matching problem, where vertices have keyword labels and edges have distance intervals and signs (*exclusion in, exclusion out, mutual exclusion* and *mutual inclusion*) [11]. To address the scalability barrier of spatial pattern matching for directional constraints, we re-frame the problem as a question of satisfiability per location (akin to per graph of objects). In other words, instead of retrieving every set of objects meeting the spatial search constraints, we seek to determine *if* a given set of objects associated with a location meets the constraints of the query.

3 COMPASS DATA STRUCTURES

To enable spatial search without relying on costly graph representations, *COMPASS* encodes spatial entities using *Location-Object* data structures and *Object-Object Concept Map* data structures. Our data structures are implemented hierarchically, with *Objects* as the atomic elements. Objects are semantically grouped into *Locations*, which are grouped into *Regions*, which segment the world. For example, the 'Washington DC' region might have 'park' and 'hospital' locations, and the 'park' might have 'swing', 'slide', and 'bench' objects, with directional relations defined between them. For a more detailed analysis of the motivation and methods for assigning objects to locations, see *Geospatially Enhanced Search with Terrain Augmented Location Targeting* (***GESTALT***) [18].

3.1 Location-Object Data Structure

A *Location-Object* data structure is a per-location set-based representation of the directional relations between objects and their locations. It subdivides the space around a location into the four cardinal quadrants (NorthWest, NorthEast, SouthWest, SouthEast) centered on the location coordinate.

Each location-object relationship is encoded independently of other objects at that location. Figure 1 shows how we encode objects (1a) based on their relative position to the location coordinates (1b).

3.2 Object-Object Concept Map Data Structure

An *Object-Object concept map* (CM) is an $(n \times n)$ matrix representation that implicitly encodes the m directional relations between

n stored objects, with similar intuition to image concept map representations [25]. Matrix entries are either 0, representing empty space, or an object ID, representing the relative position of that object to the $(n-1)$ other objects. Using Algorithm 2, we assign each object to a position (i, j) in the matrix where i is its order of appearance from north to south and j is its order of appearance from west to east. This method preserves the relative ordering of objects with respect to the original Cartesian representation. Figure 2 shows how a set of objects (2a) are encoded in a concept map (2b), preserving their directional relations.

4 COMPASS SEARCH ALGORITHMS

Our choice of data structures allows us to employ search methods that would otherwise be unsuitable for graph structures. This section outlines the methods used to accomplish *COMPASS*'s three forms of spatial search.

4.1 Location-Object Search

Location-Object search allows a user to retrieve locations by specifying directional constraints for objects relative to the center of the location that 'owns' them. It assumes a globally aligned coordinate system and the existence of a *Location-Object* data structure for each location in the region of interest. To issue a *Location-Object* query, a user inserts query objects around the 'center' of the query canvas, encoding directional constraints relative to the desired location (NW/NE/SW/SE). *Location-Object* search is effectively a set intersection operation. It proceeds by iterating through the candidate locations, searching each location's (NW/NE/SW/SE) quadrants for object-quadrant relationships that match the directional constraints specified in the query. Figure 1c shows how to query the Location-Object data structure, and the process is formalized by Algorithm 1. The search function returns candidate locations ranked by the number of query terms that match the appropriate quadrant for the location and can return fuzzy partial matches.

4.2 Object-Object Search

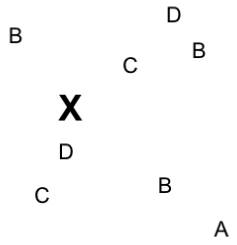
Object-Object search allows a user to retrieve locations by specifying directional constraints between objects and comparing them to the directional relationships of objects in the underlying database. It assumes compliance with a globally aligned coordinate system. The *Object-Object* query is specified pictorially and is converted into a concept map matrix to capture its directional constraints. The *Recursive Grid Search* (RGS) (Algorithm 3) uses the input concept map query to identify which locations have objects matching the directional constraints specified in the query. The search process is depicted in figure 2c, but in general terms, the *Recursive Grid Search* traverses the database objects starting from the north. Once the northernmost query object is encountered, any database objects previously encountered are irrelevant to the query and can be pruned. The same holds for the western direction. Pruning occurs recursively in alternating directions until all query objects are found or the traversal completes without matching all query terms.

4.3 Cardinality-Invariant Object-Object Search

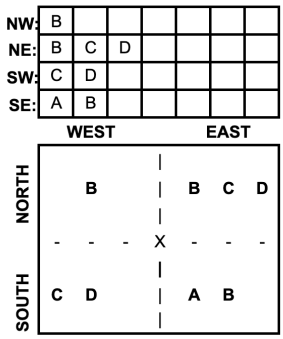
Cardinality Invariance extends *Object-Object* search (Algorithm 3) by removing the assumption of compliance with a globally aligned coordinate system. We address the coordinate system alignment issue by rotating the query pattern the minimal number of times needed to cover the distinct configurations relative to the actual cardinal directions. Algortihm 4 describes the process of rotating the points of the query term around their centroid, and a visual depiction is in figure 3. We generate the minimum bounding rectangle for the query points and impose three reference points: 'North', 'West', and 'Northwest' on the bounding rectangle relative to the centroid. For each point in the query, we compute the angle between it, the query centroid, and each reference point. Rotating the query configuration by each angle aligns the given point to each reference direction. The unique set of query configurations obtained in this manner represents the minimum set of object-object

Algorithm 1 Location-Object Search

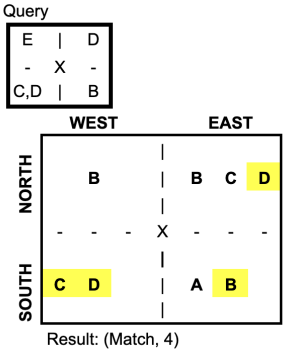
```
Q A query of Objects with names and coordinates
D A database of Objects with names and coordinates
Q.Loc Midpoint of query canvas, representing position of Location sought
D.Loc Location to which Objects in D are assigned
- - - - -
procedure objectLocationSearch(Q, D, Q.Loc, D.Loc)
    D.Dict ← makeLocationCentricStructure(D,D.Loc)
    Q.Dict ← makeLocationCentricStructure(Q,Q.Loc)
    matches ← 0
    for Each quadrant in [NW, NE, SW, SE] do
        for Each point P in Q.Dict[quadrant] do
            if D.Dict[quadrant] contains P then
                matches ← matches + 1
                Pop P from D.Dict[quadrant]
            end if
        end for
    end for
    return matches ▷ Used to rank against other candidate locs
end procedure
————————————————————————————————
D A database of Objects with names and coordinates
D.Loc Location to which Objects in D are assigned
- - - - -
procedure MakeLocationCentricStructure(D, D.Loc)
    Dict ←{ NW: [ ], NE:[ ], SW: [ ], SE: [ ] }
    for Obj in D do
        if Obj.y < D.Loc.y and Obj.x < D.Loc.x then
            Add Obj.Name to Dict.SW
        else if Obj.y < D.Loc.y and Obj.x >= D.Loc.x then
            Add Obj.Name to Dict.SE
        else if Obj.y >= D.Loc.y and Obj.x < D.Loc.x then
            Add Obj.Name to Dict.NW
        else
            Add Obj.Name to Dict.NE
        end if
    end for
    return Dict
end procedure
```



(a) A candidate location X has named objects A-D with the spatial layout depicted above.

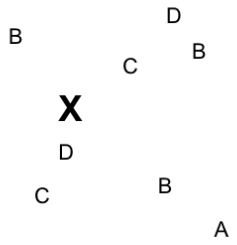


(b) The objects are binned into spatial quadrants based on their relative position to the location coordinates, X.

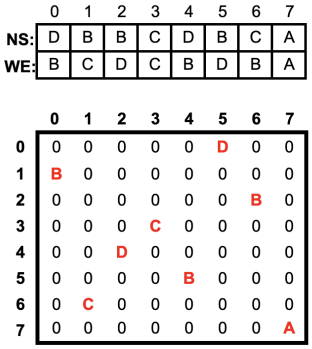


(c) Rank the locations by the number of query terms found in the correct quadrant for the location.

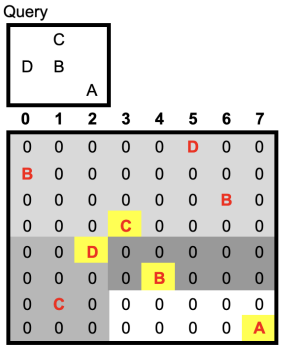
Figure 1: Location-Object Search Method. A Location-Object data structure (Figure 1b) is generated based on the cardinal relations between the objects and the location (Figure 1a). Then a pictorial query is matched against the structure (Figure 1c).



(a) A candidate location X has named objects A-D with the spatial layout depicted above.



(b) Objects associated with location X are ordered from North to South (NS) and West to East (WE) and mapped into a matrix with corresponding indices.



(c) Recursive search. Each recursion is a darker shade, with an unpruned area in white. Objects highlighted in yellow are found to match the query configuration; candidate location X is a match for the query.

Figure 2: Object-Object Search Method. A Concept Map data structure (Figure 2b) is generated by ordering the objects associated with a candidate location (Figure 2a) from North to South and West to East. The search step (Figure 2c) then recursively prunes the concept map until ANY matching configuration of objects is identified or the query constraints are not satisfied.

queries required to determine if the pattern specified in the query matches, irrespective of cardinality.

5 EXPERIMENTAL SETUP

We design our experiments to balance the power-law distribution of real-world object placement with the cleanliness required for robust experimental evaluation. We provide a data generator that produces a replicable benchmark dataset. We generate objects using a power-law distribution and generate query terms from a disjoint set of labels from those objects, enabling precision and recall measurements. When generating queries, the data generator simulates real-world user variance and error by introducing the following distortions:

(1) *Translation*, which shifts the query objects in x and y, to simulate lack of user knowledge about where in the region objects lie.
(2) *Dilation*, which expands the query from its centroid to simulate lack of user knowledge about the distance between objects.

Algorithm 2 Concept Mapping

```
D A database of Objects with names and coordinates
- - - - -
procedure createConceptMap(D)
    xList, yList ← []
    M ← [[][]]                                  ▷ A matrix of 0s
    for Obj in D do                ▷ Where D sorted (desc) on y coord
        Add Obj to yList
    end for
    for Obj in D do                ▷ Where D sorted (desc) on x coord
        Add Obj to xList
    end for
    for Obj in xlist do
        i ← index of Obj in xList
        j ← index of Obj in yList
        M[i][j] ← Obj.Name
    end for
    return xList, yList
end procedure
————————————————————————————————
xList A list of terms sorted by x-coordinate   ▷ Same number of
terms as yList
yList A list of terms sorted by y-coordinate   ▷ Same number of
terms as xList
- - - - -
procedure Order Search Terms(xList, yList)
    Traversed ← []
    dir = 'x'
    while len(Traversed) < len(xList) do
        if xList not empty and dir == 'x' then
            term = pop(xList)
            if term not in Traversed then
                Append term to Traversed
                dir = 'y'
            end if
        end if
        if yList not empty and dir == 'y' then
            term = pop(yList)
            if term not in Traversed then
                Append term to Traversed
                dir = 'x'
            end if
        end if
    end while
    return Traversed
end procedure
```

(3) *Rotation*, which spins the query about its centroid to simulate a query that is misaligned with the global coordinate systems.

We generate two benchmark datasets, one with the aforementioned distortions (*Distorted*) and one without (*Normal*). In both datasets, we vary the number of objects in the database and in the query, which allows for empirical measurements of search performance across different spatial search methods.

Algorithm 3 Object-Object Search

```
Q A query of Objects with names and coordinates
D A database of Objects with names and coordinates
- - - - -
procedure objectObjectSearch(Q, D)
    D.M ← createConceptMap(D)
    Q.M ← createConceptMap(Q)
    Q.L ← orderSearchTerms(M)
    return recursiveConceptMapSearch(D.M,Q.L)
end procedure
————————————————————————————————
D.M A ConceptMap Matrix with objects or 0s; [0,0] is NW most
point
Q.L A NW to SE ordered list of query objects
Dir The alternating cardinal direction to prune from, 'N' (default)
or 'W'
- - - - -
procedure recursiveConceptMapSearch(D.M,Q.L,Dir)
    if Q.L has only 1 item then                    ▷ Base Case
        if pop(Q.L) in D.M then
            return True
        else
            return False
        end if
    end if
    P ← pop(Q.L)
    if P not in D.M then
        return False
    end if
    D.M′ ← Prune all objects Dir of P
    Dir ← changeDirection(Dir)
    return recursiveConceptMapSearch(D.M′,Q.L,Dir)
end procedure

procedure changeDirection(Dir)
    if Dir == 'N' then
        return 'W'
    else
        return 'N'
    end if
end procedure
```

6 RESULTS

Our theoretical analysis and experiments show that *Cardinality Invariant Search* performs well and scales better than expected.

6.1 Theoretical Complexity

Our **locationObjectSearch** implementation searches each quadrant of the candidate location for matching objects from the query, resulting in a time complexity of $O(G \times (Q+n))$ for Q query points and n objects for each of G locations. As the number of query points is usually much smaller than the number of objects per location, we expect to see $O(G \times n)$ time complexity in most scenarios.

Our **objectObjectSearch** implementation traverses a matrix of size $(n \times n)$ from the North and West, slicing the matrix smaller and

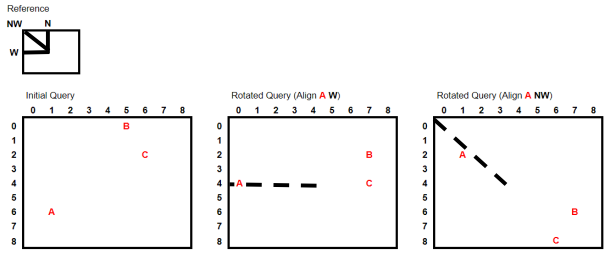


Figure 3: Cardinality-Invariant Object-Object search Method. The query configuration is rotated at most $3Q$ times (where Q is the number of query terms) to align each term to the North, West, and Northwest reference points. The original query configuration (left) is Pictured above, followed by two of the nine possible rotations for these three query terms.

Algorithm 4 Cardinality-Invariant Object-Object Search

Q *A query of Objects with names and coordinates*
D *A database of Objects with names and coordinates*
- - - - -
procedure CARDINALITYINVARIANTOBJECTOBJECTSEARCH(Q, D)
  $B \leftarrow$ Bounding box of points in Q
  $C \leftarrow$ centroid of points in Q
  $N \leftarrow$ North Center point of B
  $W \leftarrow$ West Center point of B
  $NW \leftarrow$ Northwest Center point of B
  $D.M \leftarrow$ createConceptMap(D)
  for Each Dir in $[N, W, NW]$ **do** ▷ Reference **Dir**ection
    for Each point P in Q **do**
      $\theta \leftarrow \angle\ RCP$ ▷ Measured clockwise
      $Q' \leftarrow$ Rotate(Q, θ) ▷ Rotate counterclockwise
      $M \leftarrow$ createConceptMap(Q')
      $Q'.L \leftarrow$ orderSearchTerms(M)
      if recursiveObjectObjectSearch($D.M$, $Q'.L$) == True **then**
        return True
      end if
    end for
  end for
  return False
end procedure

smaller as it traverses. Since no part of the matrix is looked at more than once, this results in a time complexity of $O(G \times (Q + n^2))$ for Q query points and n objects for each of G locations. This reduces to $O(G \times n^2)$ when $Q \ll n$. Our **cardinalityInvariantObjectObjectSearch** implementation performs **objectObjectSearch** at most $3Q$ times in the worst case (when every query alignment to each of three reference points is necessary), resulting in an expected complexity of $O(GQ \times (Q + n^2))$ for Q query points and n objects for each of G locations, which reduces to $O(G \times n^2)$ when $Q \ll n$.

6.2 Empirical Performance

We empirically demonstrate the performance of the three search methods by comparing the precision, recall, and execution times across a varying number of query constraints.

6.2.1 Precision and Recall Performance. Table 1 shows the precision and recall of all three spatial search methods on the data we generated. *Location-Object* search and *Object-Object* search both rely on the cardinal direction of the query matching the cardinal direction of the actual configuration of the objects under search. The cardinal direction is rarely aligned with the query direction by chance, reflected in the lower precision and recall of these methods. *Cardinality Invariant Object-Object* search is entirely independent of cardinal direction and therefore achieves perfect recall, finding all the locations matching the query configuration, regardless of cardinal direction. The precision of *Cardinality Invariant* search is high but not perfect due to collisions during the data generation process, where query terms inserted in the database may, by chance, match the spatial configuration of the query. The rarity of collisions (less than 10%), measured across various query sizes, indicates the discriminative power of directional relationship patterns.

6.2.2 Query Time. The query response time for *Location-Object* search and *Object-Object* search is negligible with increased query terms (figure 4). Cardinality Invariant *Object-Object* search grows logarithmically with the number of query terms, which is significantly better than the theoretical complexity, which indicates the search time should scale with $O(Q^2)$ in the worst-case query configuration. The empirical results show significantly better scalability than the theoretical worst-case because the number of rotations needed to ensure invariance for cardinal directionality is less than Q in practice. The worst case is when query points are arranged in a circle, and every point must be aligned to the **N**, **W**, and **NW** reference points, resulting in $3Q$ runs of the regular *Object-Object* search. In practice, a pictorial query will typically contain well-dispersed points over the query canvas. In those cases, aligning many of the points in the "middle" of the query space to a given reference point will result in the same Concept Map representation as aligning an outer point to it, and so that Concept Map only needs to be checked against the database once.

6.2.3 Hardware specifications. All experimental results were obtained using a 2023 Apple MacBook Pro with CPU Apple M2 Max and 32GB RAM, MacOS Ventura 13.52.

Search Method	Precision	Recall
Location-Object	0.160	0.109
Object-Object	0.160	0.109
Cardinality Invariant Object-Object	**0.913**	**1.000**

Table 1: Performance Metrics Across *COMPASS*'s 3 Types of Spatial Search. Cardinality Invariant Object-Object search outperforms the other two methods, which fail when the query configuration is not provided in the same cardinal alignment that the objects exist in.

7 RELATED WORK

Most practical applications of spatial search aim to answer queries like "Where are the Italian restaurants near me?". Most of the work

Algorithm	Implementation			Relationship Constraints				Search Features			Complexity
	Encoding	Search	Objects	Keyword	Metric	Topological	Directional	Fuzzy	Negation	Card. Inv.	
SKECa+ [13]	N/A	SKQ	**P**	X	X				X	N/A	$O(rn^Q)$
PQL [8]	Set	SI	*P,L,R*	X	X	X	X	X	X		Unclear
McheckSsl [21, 22, 24]	Set	SI	*P*	X	X				X		$O(n'^2 + 2^{n'})$
GESTALT$_{SI-Basic}$ [18]	Set	SI	*P*	X						N/A	$O(Gn)$
GESTALT$_{SI-Ranked}$ [18]	Set	SI	*P*	X						N/A	$O(G(n + n'Q))$
GESTALT$_{SI-Fuzzy}$ [18]	Set	SI	*P*	X				X		N/A	$O(QGn)$
PQIS [12]	Link	SGM	*P*	X	X		X		X		$O(m^m)$
Spacekey$_{MPJ}$ [10, 11]	Link	SKQ & SGM	*P*	X	X			X	X		$O(m\zeta^2 + \xi)$
Spacekey$_{SPJ}$ [10, 11]	Link	SKQ & SGM	*P*	X	X			X	X		$O(n^4 + mn^2 + \xi)$
ESPM [5]	Link	SKQ & SGM	*P*	X	X			X	X		$O(n'^n)$
MSJ$_{MSJ}$ [19]	Link	CSP	*P,L,R*		X	X	X	X			$O(n^Q)^\star$
MSJ$_{WR}$ [19]	Link	CSP	*P,L,R*		X	X	X	X			$O(n^m)^\star$
MSJ$_{JWR}$ [19]	Link	CSP	*P,L,R*		X	X	X	X			$O(n^m)^\star$
STARVARS [16]	Segment	CSP	*P*				X			X	$O(m^n)$
SketchMapia [15, 20]	Link	SGM	*P,L,R*	X		X	X	X		X	Unclear
OSS [17]	Segment	Other	*P,R*		X	X	X			X	$O(n)^\star$
SRQL [6, 7]	Segment	Other	*R*	X		X	X		X		Unclear
COMPASS$_{LO}$ [ours]	Set	SI	*P*	X			X		X		$O(G(Q + n))$
COMPASS$_{OO}$ [ours]	CM	RGS	*P*	X			X				$O(G(Q + n^2))$
COMPASS$_{CI}$ [ours]	CM	RGS	*P*	X			X			X	$O(G(Q^2 + Qn^2))$

Table 2: Summary of related work.
Where the authors do not provide worst-case complexity, we estimate (denoted with *). n is the number of spatial objects in the database, m is the number of relations, G is the number of object collections (locations) to search over, Q is the number of query objects, n' is the subset of objects matching a keyword query, ζ is a sampling threshold in $[0, 1]$ and ξ is the maximal number of partial matches to a query

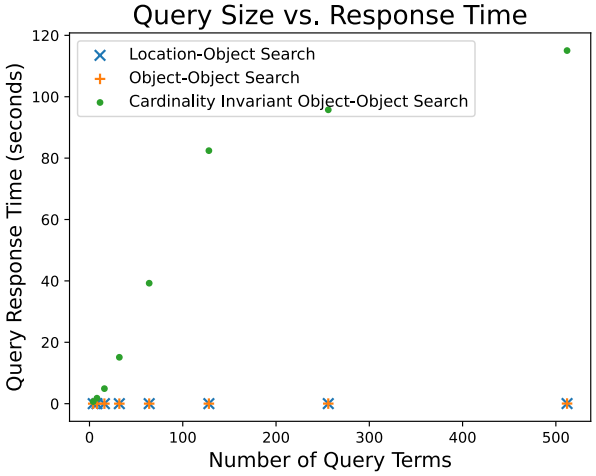


Figure 4: Empirical Scalability of *COMPASS*'s three search methods as number of query terms (constraints) increases. Cardinality Invariant Object-Object search is the only method with increased response times as query terms grow.

in this area is based on Spatial Keyword Queries (SKQ), which handle topological and metric constraints but lack support for directional constraints [2, 13, 18, 26], or conventional spatial joins, which do not typically capture more than two-way directional relations [14]. Directional constraints are densely interrelated, and current methods for encoding and searching over them (set intersection, subgraph matching, and constraint satisfaction problems) are impractical at scale. We describe the notable works in these areas and include a complexity analysis in Table 2.

7.1 Set Intersection (SI)

Set-intersection (SI) approaches allow users to specify a combination of keyword, topological, metric, and directional constraints to query against a database of objects [8, 21–24]. These approaches are limited by the requirement to extract the sets matching each specified constraint from the underlying representation before intersecting them. Most modern approaches employ set intersection as an initial keyword filter before conducting the more expensive search operations described below [18, 20].

7.2 Subgraph Matching (SGM)

Subgraph matching (SGM) approaches assume the database is encoded as a graph of objects and relations, and queries are encoded as a subgraph of objects and constraints of interest. These methods then seek to identify where the query pattern exists in the database graph. Spatial SGM methods like PQIS [12] and Spacekey$_{MPJ}$ [11] approach exponential in the number of constraints, and ESPM and SpaceKey$_{MSJ}$ are bound by the number of nodes in exponential [5]

or quartic [11] time. Fuzziness and partial matching adds further complexity [11].

7.3 Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSP) for spatial pattern matching find an assignment of valid variables (database objects) given the query constraints. In the worst case, where constraints are numerous, specificity is low, and constraints are poorly ordered, CSP approaches perform poorly. Most CSP are at least cubic in the number of objects [9]. Some approaches like StarVars, are exponential in the number of objects [16]. *Forward Checking* algorithms, like MSJ_{WR}, MSJ_{JWR}, and $_{MSJ}$ are exponential in the number of constraints or in the number of query terms [19]. *Sketchmapia*, which combines a qualitative constraint network representation with a subgraph matching solver, we estimate performs similarly [15, 20].

7.4 Summary

Table 2 shows that *COMPASS* offers a combination of directional constraints, cardinality invariance and better worst-case complexity than systems with comparable functionality. Formulating spatial pattern matching problems as graph or constraint network problems has fostered search methods dependent on those data structures. Set intersection, subgraph matching, and constraint satisfaction problems all degrade when the number of constraints and query terms increase, making them unsuitable for dense directional relations. The data structures and algorithms presented in *COMPASS* re-frame the underlying representation of spatial pattern matching problems so that new search methods can be applied that scale reasonably for directional relation-based search.

8 CONCLUSION

We present *COMPASS*, a collection of data structures and algorithms that enable search over directional spatial relations at scale. To accomplish this, we develop the matrix-based *concept map* data structure to implicitly encode the otherwise unmanageable number of pairwise directional relations between objects. We further simplify the previously intractable directional spatial pattern matching problem by encoding geospatial objects hierarchically, dividing them into semantically meaningful groups called *locations*. We provide a detailed comparison of existing approaches to spatial pattern matching, and show that the theoretical worst-case complexity of our approach is significantly better than any previous approaches that handle directional relations. We hope that by taking the first step in solving this simplified version of the directional spatial pattern matching problem, this work will inspire further attempts to break the complexity barrier of this powerful but computationally challenging method of spatial search.

ACKNOWLEDGMENTS

This work was sponsored in part by the NSF under Grants IIS-18-16889, IIS-20-41415, and IIS-21-14451, and the Australian-American Fulbright Commission. Its contents do not necessarily represent the official views of the Fulbright Program.

REFERENCES

[1] Bertella P. G. K. , Y. K. Lopes, R. A. P. de Oliveira, and A. C. Carniel. 2022. A systematic review of spatial approximations in spatial database systems. *Journal of Information and Data Management* 13, 2 (2022).

[2] Cao X. , L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. 2012. Spatial keyword querying. In *Conceptual Modeling: 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings 31*. Springer, 16–29.

[3] Carniel A. C. . 2020. Spatial information retrieval in digital ecosystems: A comprehensive survey. In *Proceedings of the 12th International Conference on Management of Digital EcoSystems*. 10–17.

[4] Chaves Carniel A. . 2023. Defining and designing spatial queries: the role of spatial relationships. *Geo-spatial Information Science* (2023), 1–25.

[5] Chen H. , Y. Fang, Y. Zhang, W. Zhang, and L. Wang. 2019. ESPM: Efficient spatial pattern matching. *IEEE Transactions on Knowledge and Data Engineering* 32, 6 (2019), 1227–1233.

[6] Della Penna G. , D. Magazzeni, and S. Orefice. 2012. A spatial relation-based framework to perform visual information extraction. *Knowledge and information systems* 30 (2012), 667–692.

[7] Della Penna G. , D. Magazzeni, and S. Orefice. 2017. A formal framework to represent spatial knowledge. *Knowledge and Information Systems* 51 (2017), 311–338.

[8] Di Loreto F. , F. Ferri, F. Massari, and M. Rafanelli. 1996. A pictorial query language for geographical databases. In *Proceedings of the workshop on Advanced visual interfaces*. 233–244.

[9] Dylla F. , J. H. Lee, T. Mossakowski, T. Schneider, A. V. Delden, J. V. D. Ven, and D. Wolter. 2017. A survey of qualitative spatial and temporal calculi: algebraic and computational properties. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 1–39.

[10] Fang Y. , R. Cheng, J. Wang, L. Budiman, G. Cong, and N. Mamoulis. 2018. SpaceKey: exploring patterns in spatial databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1577–1580.

[11] Fang Y. , Y. Li, R. Cheng, N. Mamoulis, and G. Cong. 2019. Evaluating pattern matching queries for spatial databases. *The VLDB Journal* 28 (2019), 649–673.

[12] Folkers A. , H. Samet, and A. Soffer. 2000. Processing pictorial queries with multiple instances using isomorphic subgraphs. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Vol. 4. IEEE, 51–54.

[13] Guo T. , X. Cao, and G. Cong. 2015. Efficient algorithms for answering the m-closest keywords query. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 405–418.

[14] Jacox E. H. and H. Samet. 2007. Spatial join techniques. *ACM Transactions on Database Systems (TODS)* 32, 1 (2007), 7–es.

[15] Jan S. and A. Schwering. 2015. SketchMapia: a framework for qualitative alignment of sketch maps and metric maps. In *Proceedings of the AGILE conference on Geographic Information Science*, Vol. 18. Association of Geographic Information Laboratories in Europe.

[16] Lee J. H. , J. Renz, D. Wolter, et al. 2013. Starvars-effective reasoning about relative directions. (2013).

[17] Liu X. , S. Shekhar, and S. Chawla. 2003. Object-based directional query processing in spatial databases. *IEEE transactions on knowledge and data engineering* 15, 2 (2003), 295–304.

[18] O'Sullivan K. , N. R. Schneider, and H. Samet. 2023. GESTALT: Geospatially Enhanced Search with Terrain Augmented Location Targeting. *Under Review at GeoSearch 2023* (2023).

[19] Papadias D. , N. Mamoulis, and B. Delis. 1998. Algorithms for querying by spatial structure. In *Proceedings of Very Large Data Bases Conference (VLDB), New York*.

[20] Schwering A. , J. Wang, M. Chipofya, S. Jan, R. Li, and K. Broelemann. 2014. SketchMapia: Qualitative representations for the alignment of sketch and metric maps. *Spatial cognition & computation* 14, 3 (2014), 220–254.

[21] Soffer A. and H. Samet. 1997. Pictorial query specification for browsing through image databasess. In *Proceedings of the Second International Conference on Visual Information Systems*. 117–124.

[22] Soffer A. and H. Samet. 1998. Pictorial query specification for browsing through spatially referenced image databases. *Journal of Visual Languages & Computing* 9, 6 (1998), 567–596.

[23] Soffer A. and H. Samet. 1999. Query processing and optimization for pictorial query trees. In *International Conference on Advances in Visual Information Systems*. Springer, 60–68.

[24] Softer A. and H. Samet. 1997. Handling multiple instances of symbols in pictorial queries by image similarity. In *Image Databases and Multi-Media Search*. World Scientific, 77–85.

[25] Xu H. , J. Wang, X.-S. Hua, and S. Li. 2010. Image search by concept map. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 275–282.

[26] Zhang D. , Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. 2009. Keyword search in spatial databases: Towards searching by document. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 688–699.