Consistency Checking for Euclidean Spatial Constraints: A Dimension Graph Approach *

Xuan Liu
IBM T.J. Watson Research Center
Hawthorne, NY 10532, USA
xuanliu@us.ibm.com

Shashi Shekhar
Computer Science Department
University of Minnesota
Minneapolis, MN 55455, USA
shekhar@cs.umn.edu

Sanjay Chawla
Vignette Corporation
Waltham, MA, USA
schawla@vignette.com

Abstract

In this paper, we address the problem of consistency checking for Euclidean spatial constraints. A dimension graph representation is proposed to maintain the Euclidean spatial constraints among objects. The basic idea is to project the spatial constraints on both X and Y dimensions, and to construct a dimension graph on each dimension. Using a dimension graph representation transforms the problem of consistency checking into the problem of graph cycle detection. Consistency checking can be achieved with O(N+E) time as well as space complexity, where N is the number of spatial objects, and E is the number of spatial predicates in the constraint. The proposed approach is faster than $O(N^2)$ when the number of predicates is much smaller than N^2 and there are few disjunctions in the spatial constraint. The dimension graph and consistency checking algorithm can be used for points, intervals and polygons in two-dimensional space. The algorithm can also guarantee global consistency.

Keywords: Euclidean spatial constraint, consistency checking, dimension graph, directional relationship

1 Introduction

The goal of spatial database [7, 9, 15] management systems is the effective and efficient management of data related to physical space(in geography, urban planning, astronomy) or conceptual information space (in a multi-dimensional decision support system, fluid flow, or an electro-magnetic field). The distinguishing features of a spatial database management system are the use of complex data types like points, lines, and polygons to represent spatial objects and the existence of many potential relationships between spatial objects.

An important means of maintaining the integrity of databases in general and spatial databases in particular is consistency checking, the identification of contradictory information in a database. For example, if A, B and C are three spatial objects and if B is west of A and C is west of B and if the database indicates that C is east of A then the information is inconsistent. The existence of many potential spatial relationships implies that consistency checking is more challenging in spatial databases than in traditional relational databases.

Currently most consistency checking is based on Allens's propagation algorithm[1], which was originally devised for checking temporal relationships on one-dimensional objects.However, spatial relationships are typically formulated among multi-dimensional objects and the straightforward extension of Allen's algorithm to spatial relationships is prohibitively expensive.

In this paper, we address the problem of consistency checking for directional spatial constraints in two-dimensional Euclidean space. We propose a dimension graph representation for maintaining the spatial constraints among objects. Basically, the spatial constraints are projected on two dimensions(X and Y), and the derived constraints on each dimension(X/Y) are maintained in different graphs(X/Y graph). The problem of constraint consistency checking is thus transformed into a graph cycle detection problem on dimension graphs. Cycle detection can be solved by traversing the graph in linear time. As we will see in later sections, the proposed consistency checking algorithm is efficient in terms of both time and space. The algorithm also guarantees the global consistency.

*This work is sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

1.1 Problem Definition

In this paper, we explore consistency checking for Euclidean spatial constraints among points, intervals, and 2D Minimum Bounding Rectangles(MBRs) in spatial databases. An MBR of a spatial object is the smallest axis parallel rectangle which covers the objects.

Consistency Checking Problem:

Given: A collection of 0-th order[1] spatial constraints in terms of disjunctions and/or conjunctions of spatial predicates

Find: Consistency, i.e., return TRUE if the constraints are consistent, False otherwise

Objective: Reduce computational complexity

Constraint:

(a) 2D Euclidean space.

(b) 2D region objects are approximated by MBRs

(c) Spatial objects are of homogeneous types, (i.e., point pairs, interval pairs, or MBRs in 2D)

Consider an example of directional constraints among point objects A, B, and C. Assume one constraint says A is southeast of B, B is northwest of C, and A is northeast of C. This constraint is consistent, and hence the result will be TRUE. An example of the possible spatial configurations satisfying this constraint is shown in Figure 1.

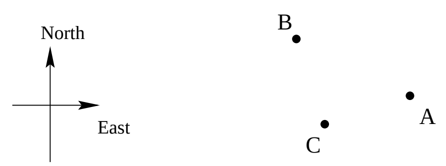


Figure 1. One possible spatial configuration for point objects A, B, and C

Suppose we have another constraint among A, B, and C: A is strictly north of B, B is northwest of C, and A is northeast of C. This constraint is inconsistent since there does not exist any spatial configuration of A, B, and C satisfying this constraint. The consistency checking algorithm should return FALSE.

1.2 Related work and our contributions

Most of the previous studies on consistency checking are based on Allen's consistency checking algorithm [1] for constraints among intervals. The basic approach used a transitive closure algorithm, which incurs a high order of time and space complexity. Later, Hernandez [8] presented mechanisms to maintain the consistency of a knowledge base of spatial information based on a qualitative representation of 2D positions. His approach improved Allen's algorithm by using the heuristic of the rich structure of the spatial domain. Bowman and et al.[4, 3] addressed the problem of consistency checking between multiple viewpoints using strategies based on unification. Other work[10, 6, 13] has focused on the problem of consistency checking for more general constraints, such as 1-th order constraints. Manandhar[12] discussed deterministic consistency checking for LP constraints. Beneventano and et al. [2] focused on the problem of providing a theoretical framework for consistency checking of integrity constraints in a complicated object database environment. We briefly describe Allen's Algorithm since it is basic to many algorithms used in consistency checking.

1.2.1 Allen's Propagation Algorithm

Allen[1] summarized thirteen mutually exclusive relationships to express any possible relationship between intervals(Table 1).

Relationships	symbol	Symbol for Inverse	Pictorial Example
X before Y	$<$	$>$	XXX YYY
X equal Y	$=$	$=$	XXX YYY
X meets Y	m	mi	XXXYYY
X overlaps Y	o	oi	XXX YYY
X during Y	d	di	XXX YYYYYY
X starts Y	s	si	XXX YYYYY
X finishes Y	f	fi	XXX YYYYY

Table 1. Thirteen possible relationships proposed by Allen[1]

In Allen's work[1], the relationships between intervals are maintained in a network where each node N_i represents the individual interval i, and each arc N(i,j) is associated with possible relationships between the corresponding interval pair i and j. The basic algorithm Allen used for maintaining relationships was propagating new relationships by computing the transitive closure of the relationships between intervals.

Figure 2 shows a network used by Allen for consistency checking. 2(a) is the network for two inputs, i.e., *S overlaps or meets L*, and *S is before, meets, is metby, or after R*. After the second input is added, the algorithm computes the constraint between L and R, and the resulting network is shown as 2(b). If we add a new fact *L overlaps, starts, or is during*

[1] 0-th order spatial constraints means that there are no free variables but only constant objects

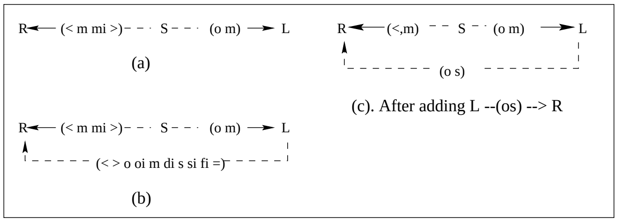


Figure 2. Examples of Allen's algorithm

R, we need to propagate its effect through the network, thus obtaining the resulting network 2(c).

As explained in Allen's paper[1], the time complexity of this algorithm is calculated as: $13 \times \frac{(N-1)(N-2)}{2}$ for N intervals, i.e., $O(N^2)$. The space requirement for the algorithm is also $O(N^2)$.

Allen noted that [1], one problem with this algorithm is that it does not detect all inconsistencies in its input. "In fact, it only guarantees consistency between three node subnetworks. There are networks that can be added which appear consistent by viewing any three nodes, but for which there is no consistent overall labeling of the network."In other words, the algorithm cannot guarantee global consistency.

1.2.2 Our Approach

In this paper, we propose a new strategy to process consistency checking for Euclidean spatial constraints among objects. We use a geometric approach by incorporating spatial domain information. We propose dimension graphs to maintain the spatial constraints among objects. Each conjunctive constraint is projected on both X and Y dimensions, and a dimension graph is constructed on each dimension. Spatial constraints in general format can be converted to the Disjunctive Normal Form(DNF)[14] , and dimension graphs are constructed for each conjunction. Using a dimension graph representation transforms the problem of constraint consistency checking into the problem of graph cycle detection on each dimension graph. Cycle detection can be solved efficiently with O(N+E) time as well as space complexity, where N is the number of spatial objects, and E is the number of spatial predicates in the constraint. Recall that Allen's algorithm has a time and space complexity of $O(N^2)$. The proposed approach is faster when the number of predicates is much smaller than N^2 and there are few disjunctions in the spatial constraint. Since the algorithm returns TRUE if and only if the dimension graph of at least one conjunction contains no cycle, which means there exists at least one consistent overall constraint. The algorithm thus guarantees global consistency.

1.3 Scope and Outline

In this paper, we address the problem of consistency checking for spatial constraints in two-dimensional Euclidean space. We deal only with 0-th order constraints. We focus on the qualitative constraints among objects, which include topological and direction relationships. We consider the constraints among point objects, interval objects, and region objects approximated by MBRs. We focus on consistency checking for homogeneous types of objects. Consistency checking for the constraints specified among mixed types of objects(e.g. the constraints between a point and an interval) is beyond the scope of the paper, and may be addressed in future work. We only discuss a specific set of predicates defined in Euclidean space. Some constraints that are inconsistent in Euclidean space may be consistent in spherical space. Consistency checking for predicates in other space is beyond the scope of this paper.

The organization of this paper is as follows: In section 2, we propose a dimension graph representation for the conjunctive constraints among points and intervals. Consistency checking for conjunctive constraints based on the dimension graph representation is introduced in section 3. In section 4, we discuss dimension graph construction and consistency checking for conjunctive constraints among MBRs. Finally, consistency checking for constraints in general format is described in section 5. The paper ends with conclusions and recommendations for future work.

2 Dimension Graphs for Conjunctive Spatial Constraints

In this section, we describe the construction of dimension graphs for conjunctive spatial constraints among points and intervals. The basic idea is to project the conjunctive spatial constraint onto each dimension and construct a dimension graph based on constraints on each dimension. The dimension graph for points in 2D space contains an X-graph and a Y-graph and the dimension graph for intervals in 1D space contains only one graph. We also analyze the computational complexity for the dimension graph construction algorithm.

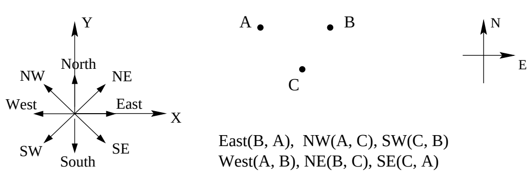
2.1 Dimension Graphs for Constraints Among Point Objects

We start by defining a set of absolute direction predicates for point objects in terms of coordinates. Here, we assume the global coordinate system are aligned with the reference frame of absolute directions, i.e., $North$ aligns with the y-axis, and $East$ align with the x-axis. The definition for each predicate is given in Table 2. The first column of the table enumerates the direction predicates. The second and third

Direction predicates	A_x, B_x	A_y, B_y
$SP(A, B)$	$=$	$=$
$North(A, B)$	$=$	$>$
$South(A, B)$	$=$	$<$
$East(A, B)$	$>$	$=$
$West(A, B)$	$<$	$=$
$NE(A, B)$	$>$	$>$
$NW(A, B)$	$<$	$>$
$SE(A, B)$	$>$	$<$
$SW(A, B)$	$<$	$<$

Table 2. Direction Predicates for point objects in terms of coordinates

columns represent the relationships between the two point objects on the X and Y dimensions respectively. A_y, B_y are the y-components of A and B, and A_x, B_x are the x-components of A and B. Figure 3(a) illustrates the definition. The predicates are defined using direction equivalence



(a) Definitions of the predicates

(b) Examples of predicates

Figure 3. Illustration of the predicates

classes [16] by partitioning the space. SP(A,B) means A and B are on the same position. $North$, $South$, $East$, and $West$ represent exact directions, while NE, NW, SE, and SW can point to any direction in their respective quadrants. Figure 3(b) shows examples of predicates describing the directional relationships among points A, B and C, i.e., B is east of A, A is northwest of C, and C is southwest of B.

The spatial constraints represented in terms of the conjunctions of predicates can be maintained on two graphs, the X-graph and the Y-graph. The nodes in both graphs represent the objects forming the constraints. The direction constraints are represented as directed edges in each graph according to the symbol in columns 2 and 3 of Table 2. The edge extends from the node with a smaller value to the node with a larger value. If the symbol is '$=$', the two nodes are merged into one node. Figure 4 shows the graph representation for the constraint North(A,B) ∧ NW(B,C) ∧ SE(B,D). The dimension graph is a union of the X-graph and Y-graph, and is constructed according to the definition of each predicate in Table 2. Algorithm 1 is the pseudo-code of the graph constructing procedure. The input of the algorithm

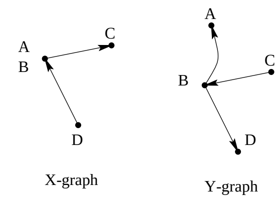


Figure 4. Dimension graph for North(A,B) ∧ NW(B,C) ∧ SE(B,D)

Algorithm 1 Constructing Dimension Graph from conjunctive spatial constraints for points: **constructGraphPoint**

Input: $conjConstraint$ is a set of conjunctive predicates
Output: $constraintGraph$ consists of the corresponding X/Y-graphs.

```
Graph constructGraphPoint(Predicates conjConstraint) {
        Graph constraintGraph = ∅;
        for each entry p ∈ conjConstraint
                add_a_predicate_point(p, &constraintGraph);
        return constraintGraph;
}

add_a_predicate_point(Predicate p, Graph* aGraph ) {
        fObject = getFirstObject(p);
        sObject = getSecondObject(p);
        addNode(fObject, sObject, aGraph.graphX);
        symbol =findXconstraint(Table 2, p);
        addEdge(symbol, fObject, sObject, aGraph.graphX);
        addNode(fObject, sObject, aGraph.graphY);
        symbol =findYconstraint(Table 2, p);
        addEdge(symbol, fObject, sObject, aGraph.graphY);
}
```

is the conjunctive constraint and the output is the dimension graph. For each predicate in the conjConstraint, the algorithm invokes the sub-function **add_a_predicate_point** to add the predicate to the dimension graph. This function calls subfunction $addNode$ to add nodes that do not exist in the dimension graph into the graph, and calls function $addEdge$ to add the spatial relationships between nodes into the dimension graph. The implementation of $addNode$ and $addEdge$ is omitted here.

We can easily summarize the computational complexity for this algorithm. Let N be the number of spatial objects involved and E be the number of spatial predicates in the conjunction.

- Time complexity = O(N+E);
  Any of the sub-functions **addNode** and **addEdge** takes constant time(O(1)). **findXconstraint** and **findYconstraint** are essentially table lookup functions, which can also be executed in constant time. The whole algorithm, therefore, has the time bound of O(E). The generated graph has at most N nodes and E edges each

in the X-graph and Y-graph.

- Space complexity O(N+E).
  We can process one dimension graph at a time, the space requirement is also linear to the graph elements.

2.2 A Dimension Graph for Conjunctive Spatial Constraints Among Intervals

The relationships between intervals proposed by Allen [1] can be defined in terms of the endpoints of intervals. Table 3 shows the definition of the 13 relationships, where A_1, A_2 and B_1, B_2 represent the start and end points of the intervals A and B respectively. Allen's symbols are used here.

Predicate name	predicates	point relationships
before	$<(A, B)$	$A_2 < B_1$
equal	$=(A, B)$	$(A_1 = B_1) \wedge (A_2 = B_2)$
overlaps	$o(A, B)$	$(A_1 < B_1) \wedge (A_2 > B_1)$ $\wedge (A_2 < B_2)$
meets	$m(A, B)$	$(A_2 = B_1)$
during	$d(A, B)$	$(A_1 > B_1) \wedge (A_2 < B_2)$
starts	$s(A, B)$	$(A_1 = B_1) \wedge (A_2 < B_2)$
finishes	$f(A, B)$	$(A_1 > B_1) \wedge (A_2 = B_2)$
after	$bi(A, B)$	$B_2 < A_1$
overlapby	$oi(A, B)$	$(B_1 < A_1) \wedge (B_2 > A_1)$ $\wedge (B_2 < A_2)$
metby	$mi(A, B)$	$(B_2 = A_1)$
duringby	$di(A, B)$	$(B_1 > A_1) \wedge (B_2 < A_2)$
startby	$si(A, B)$	$(B_1 = A_1) \wedge (B_2 < A_2)$
finishedby	$fi(A, B)$	$(B_1 > A_1) \wedge (A_2 = B_2)$

Table 3. Spatial relationships for intervals, where $<$ and bi describe directional relationships and others are topological relationships

The relationships represented in terms of conjunctions of predicates can be maintained in directed graphs, where the nodes represent start or end points of the individual intervals, and the directed edges represent the constraints between the two points. Each edge is added to the graph according to the definition in Table 3. The edges extend to the nodes with larger values from the nodes with smaller values. The nodes with the same values are merged into one node. It is worth noting that there is an intrinsic constraint between the start point and the end point of an individual interval, i.e., start-point < end-point. Figure 5 shows the graph representation for the constraint $before(S, R) \wedge meets(S, L)$. The graph is constructed according to the definition of each predicate in Table 2. The dashed arrow represents the intrinsic constraint of the start point and end points. The pseudocode of the graph constructing procedure is given in Algorithm 2.

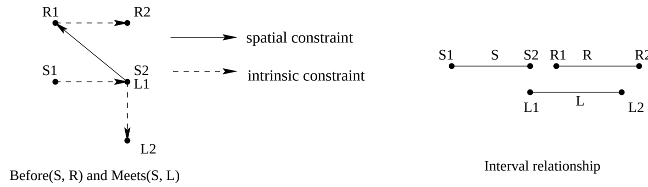


Figure 5. $before(S, R) \wedge meets(S, L)$

Algorithm 2 Constructing Graph from conjunctive constraints for intervals: **constructGraphInterval**

Input: $conjConstraint$ is a set of conjunctive predicates
Output: $constraintGraph$ is the corresponding graph

```
Graph constructGraphInterval(Set of Predicates conjConstraint) {
        constraintGraph = ∅;
        for each entry p ∈ conjConstraint
                add_a_predicate_interval(p, &constraintGraph);
        return constraintGraph;
}

add_a_predicate_interval(Predicate p Graph* aGraph ) {
        fObject = getFirstObject(p);
        sObject = getSecondObject(p);
        addIntervalNode(fObject, sObject, constraintGraph);
        pointPredicates = convertToPoint(p); //according to Table 3
        for each r(k, l) in pointPredicates {
                if (k < l) add directed edge (k, l) to aGraph;
                if (k > l) add directed edge (l, k) to aGraph;
                if (k = l) merge node k and l in aGraph;
        }
}
```

The input of the algorithm is the conjunctive constraint and the output is the corresponding dimension graph. For each predicate in the conjConstraint, the algorithm calls the subfunction **add_a_predicate_interval** to add the predicate to the dimension graph. This function adds nodes that are not in the dimension graph into the graph, and adds the spatial relationships between nodes into the graph.

Let N be the number of objects(intervals) and E be the number of interval predicates in conjConstraint. Following a similar argument in Algorithm 1, we can derive the time complexty O(E) and space complexity O(N+E).

3 Consistency checking for conjunctive constraints

In the previous section, we described dimension graph construction for conjunctive Euclidean spatial constraints among point objects or intervals. In this section, we describe the consistency checking algorithm based on the dimension graph representations.

Let's revisit the example given in Figure 4. If we add a new constraint $NE(A, C)$, the new constraint graph is given in Figure 6. A new edge is added to each graph.

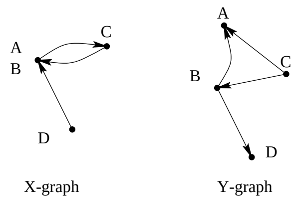


Figure 6. North(A,B) ∧ NW(B,C) ∧ SE(B,D)∧ NE(A,C)

As can be seen in the figure, a cycle is constructed in the X dimension graph. In other words, the constraint of $North(A, B) \wedge NW(B, C) \wedge SE(B, D) \wedge NE(A, C)$ requires that the x-value of A be smaller than the x-value of C, and that the x-value of C be smaller than the x-value of A. This is a contradiction, which means the constraint is inconsistent. In general, we can characterize this feature as Theorem 1.

Theorem 1 *A conjunctive constraint is consistent if and only if there exists no cycle in its corresponding dimension graphs, i.e., the graphs are all directed acyclic graphs(DAG).*

Proof: Omitted to save space. Please refer to [11] for details.

3.1 Basic Algorithm

We now describe the algorithm for consistency checking for conjunctive spatial constraints among a set of points or a set of intervals based on dimension graph representation. Consistency checking for conjunctive constraints involves two steps: 1) Constructing the corresponding dimension graph; 2) Performing cycle detection on each graph. The constraint is consistent if none of the graphs contains a cycle. The pseudo-code is described in Algorithm 3.

The algorithm first constructs the dimension graph by invoking subfunction **constructGraphPoint** or **constructGraphInterval** according to the type of the object. The function **detectCycle** performs cycle detection on the corresponding dimension graph. The algorithm returns TRUE if no cycle is detected.

A nice property of this algorithm is its efficiency. Consistency checking is simply graph cycle detection, which can easily be done in linear time. As in the previous section, let N be the number of spatial objects and E be the number of spatial predicates involved in the conjunctive constraint.

- Time complexity = O(N+E);
  O(N+E) is the time for cycle detection in a directed graph with N nodes and E edges[5]. As we explained in section 2, each X-graph and Y-graph for a set of points has at most N nodes and E edges, and the dimension graph for intervals has at most 2N nodes and 3E edges. Therefore, O(2N+3E) is the upper bound time complexity for consistency checking for a conjunctive constraint, which is the same as O(N+E).
- Space complexity =(N+E).
  We can process on one dimension graph at a time, the space requirement is also linear to the graph elements.

Algorithm 3 Consistency checking: **conjunctionConsistencyCheck**

Input: $conjunctionConstraint$ is a set of conjunctive predicates
Output: TRUE if consistent, FALSE otherwise

```
conjunctionConsistencyCheck(Set of Predicates conjunctionConstraint) {
    Graph constraintGraph = ∅;
    if the constraint is among points
        constraintGraph = constructGraphPoint (conjunctionConstraint);
    else //the constraint is among intervals
        constraintGraph = constructGraphInterval (conjunctionConstraint);
    if !detectCycle(constraintGraph)
        return TRUE;
    return FALSE;
}
```

Since our method of consistency checking is based on cycle detection in the corresponding dimension graph, the algorithm can always detect inconsistent constraints. The algorithm thus guarantees global consistency.

4 Consistency Checking for Conjunctive Constraints Among MBRs

In the previous section, we discussed the dimension graphs for conjunctive constraints among 2D points and 1D intervals, and also described the dimension graph based consistency checking algorithm. In this section, we extend our dimension graph based approach to 2D spatial objects approximated by MBRs. Examples of 2D spatial objects include polygon regions.

It is common in spatial databases to approximate 2-D regions by minimum bounding rectangles(MBRs) which are orthogonal with respect to the global coordinate system.

By using MBR approximation, we can use two representative points, namely lower-left and upper-right corners, to determine the corresponding object. In the rest of the paper, we use the notation of A_{ll} and A_{ur} to represent the lower-left and upper-right corners of the MBR for any object A. The notations of $A_{ll.x}, A_{ll.y}, A_{ur.x}$ and $A_{ur.y}$ are used to represent the x and y coordinates for the lower-left and upper-right corners of MBR A.

The directional relationship between MBRs can be determined by the relationships between the representative points. Table 4 shows the definitions of the direction predicates based on the representative points of the MBR of the objects. The first column of the table enumerates the direction predicates. The second column gives the constraints that should be satisfied by the representative points of the objects.

Predicates	conditions
$SP(A,B)$	$(A_{ll.x} = B_{ll.x}) \wedge (A_{ll.y} = B_{ll.y})$ $\wedge(A_{ur.x} = B_{ur.x}) \wedge (A_{ur.y} = B_{ur.y})$
$North(A,B)$	$(A_{ll.y} \geq B_{ur.y}) \wedge (A_{ll.x} \geq B_{ll.x})$ $\wedge(A_{ur.x} \leq B_{ur.x})$
$South(A,B)$	$(A_{ur.y} \leq B_{ll.y}) \wedge (A_{ll.x} \geq B_{ll.x})$ $\wedge(A_{ur.x} \leq B_{ur.x})$
$East(A,B)$	$(A_{ll.x} \geq B_{ur.x}) \wedge (A_{ll.y} \geq B_{ll.y})$ $\wedge(A_{ur.y} \geq B_{ur.y})$
$West(A,B)$	$(A_{ur.x} \leq B_{ll.x}) \wedge (A_{ll.y} \geq B_{ll.y})$ $\wedge(A_{ur.y} \geq B_{ur.y})$
$NE(A,B)$	$(A_{ll.x} \geq B_{ur.x}) \wedge (A_{ll.y} \geq B_{ur.y})$
$SE(A,B)$	$(A_{ll.x} \geq B_{ur.x}) \wedge (A_{ur.y} \leq B_{ll.y})$
$NW(A,B)$	$(A_{ur.x} \leq B_{ll.x}) \wedge (A_{ll.y} \geq B_{ur.y})$
$SW(A,B)$	$(A_{ur.x} \leq B_{ll.x}) \wedge (A_{ur.y} \leq B_{ll.y})$

Table 4. Direction Predicates for MBR

4.1 Dimension Graphs for Conjunctive Constraints Among MBRs

The dimension graph for each direction predicate contains an X-graph and a Y-graph which maintain the constraints that must be satisfied on the X and Y dimensions respectively. The nodes in these graphs represent the objects forming the constraints. The constraints are represented as directed edges according to the conditions specified in column 2 of Table 4. Similar to the X-graph and Y-graph for points, a directed edge goes from the node with a smaller value to the node with a larger value. If the value associated with two nodes is same, the two nodes are merged into one node. We introduce a new '**thick arrow**' edge here to represent the new relationships of $\leq$ or $\geq$. If the relationship between two nodes p and q is $p \leq q$, a 'thick arrow' directed edge is added from p to q. If the relationship between p and q is $p \geq q$, a 'thick arrow' directed edge is added from q to p. The dashed arrow is used for the intrinsic constraint between the lower-left point and the upper-right point of an MBR.

The dimension graph for a conjunctive constraint is constructed by adding the constraints specified in each predicate in the X-graph and Y-graph. Figure 7(a) and (b) shows an example of the dimension graph for a conjunctive constraint of $North(A,B) \wedge NE(B,C) \wedge SW(C,A)$. Figure 7(c) is a possible spatial configuration among MBR A, B and C.

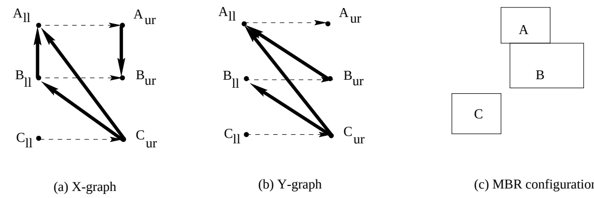


Figure 7. Example MBR configuration and its corresponding dimension graphs, thick arrow represents $\leq$, thin arrow represents $<$

We now describe dimension graph construction for conjunctive constraints for MBRs. The pseudo-code of the graph constructing procedure is given in Algorithm 4. For each predicate in the conjunctionConstraint, the algorithm invokes the sub-function **add_a_predicate_MBR** to add it to the dimension graph. The basic strategy of this function is first to transform the MBR predicate into a point-predicate set, and then to add each point predicate into the graph. To accomplish this, $addNode$ and $addEdge$ are invoked. Please refer to [11] for the detailed algorithm.

Algorithm 4 Constructing Graph for MBR constraints: **constructGraphMbr**

Input: $ConjunctionConstraint$ is a set of conjunctive predicates between MBRs
Output: $constraintGraph$ consists of the corresponding X/Y-graphs.

```
Graph constructGraphMbr(Set of Predicates conjunctionConstraint) {
    Graph constraintGraph = ∅;
    for each entry p ∈ conjunctionConstraint
        add_a_predicate_MBR(p, &constraintGraph);
    return constraintGraph;
}

add_a_predicate_MBR(Predicate aPredicate, Graph* aGraph ) {
    pointPredicates= findfromTable(aPredicate);
    for each predicate r(k, l) ∈ pointPredicates
    addNode(k, l, aGraph.graphX);
    addEdge(r(k, l), aGraph.graphX);
    addNode(k, l, aGraph.graphY);
    addEdge(r(k, l), aGraph.graphY);
}
```

Let N be the number of objects(i.e. MBRs) and E be the number of predicates in the conjunctionConstraint. We can summarize the complexity as follows:

- Time complexity = O(E);
  According to Table 4, at most four point predicates should be satisfied for each MBR direction predicate, and hence at most four edges added for each predicate. The time complexity for this algorithm is there-

fore O(E).

- Space complexity = O(N+E).
  The resulting constraintGraph consists of at most 2N nodes and 4E edges. The space requirement is linear to the number of nodes and edges, roughly 2N+4E, which is O(N+E).

4.2 Consistency Checking for Conjunctive Constraints Among MBRs

After constructing dimension graphs for conjunctive constraints among MBRs, we can check the consistency by detecting cycles in the dimension graphs. There are three possible situations and corresponding results:

Case 1: There exists no cycle (constraint predicates are consistent)

Case 2: Every cycle consists of only thick edges (constraint predicates are consistent)

Case 3: At least one cycle consists of non-thick edges(constraint predicates are inconsistent)

Case 1 is obvious. No cycle means there exists a consistent spatial configuration among all objects. Figure 7 is an example of consistent constraints whose dimension graphs contain no cycle.

In case 2, since every cycle detected contains only thick edges. Recall that any thick edge $p \rightarrow q$ represents the fact that the relationships between p and q is '$\leq$', i.e., either $p = q$ or $p < q$ could hold. If we label all thick edges as '$=$', all the nodes involved in the cycle will have same value. We can then merge this cycle with a big node consisting of all nodes involved in the cycle. The transformed graph contains no cycle, and the corresponding constraint is consistent. Figure 8 shows such an example. Figure 8 (a) is the dimension graph for constraint $North(A, B) \wedge South(B, A)$, it contains a cycle with only thick edges in the X-graph. Figure 8 (b) illustrates the transformed X-graph after each cycle is merged into one node. There is no cycle in this resulting graph. This constraint is obviously consistent. Figure 8 (c) is a sample configuration.

Case 3 is easy to understand. If there is a cycle containing at least one non-thick edge, no matter how we label the thick edges in the cycle, we cannot remove the cycle. There must be conflicts among coordinates of the objects involved in the cycle, and hence, the constraint is inconsistent. Figure 9 shows an example of the constraints among A, B, and C. The constraint is: $North(A, B) \wedge NE(B, C) \wedge SW(A, C)$. As can be seen, there exist cycles in both the X and Y dimension graphs of the constraint. No spatial configuration of A, B, C can satisfy this constraint. Thus the constraint is inconsistent.

Based on the above arguments, we can now describe the consistency checking algorithm for MBRs in Algorithm 5. The algorithm first calls function **constructGraphMbr** to construct the dimension graph, and then uses **detectCycle** to detect cycles in the dimension graph. It returns TRUE if no cycle is detected. If a detected cycle contains thin edges, the algorithm returns FALSE, otherwise, returns TRUE.

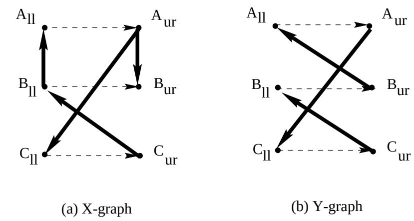


Figure 9. Dimension graphs for $North(A, B) \wedge NE(B, C) \wedge SW(A, C)$

Algorithm 5 Consistency checking for conjunctive MBR predicates: **MBRconjunctionConsistencyCheck**

Input: $conjunctionConstraint$ is a set of conjunctive predicates
Output: TRUE if consistent, FALSE otherwise

```
MBRconjunctionConsistencyCheck(Set of Predicates conjunction-
Constraint) {
    Graph constraintGraph = ∅;
    constraintGraph = constructGraphMbr (conjunctionCon-
straint);
    if !detectCycle(constraintGraph)
        return TRUE;
    else if the cycle contains thin edge
        return FALSE;
    else return TRUE;
}
```

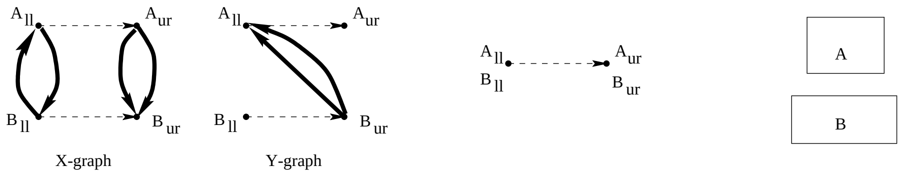
Let N be the number of objects(i.e., MBRs) and E be the number of predicates in conjunctionConstraint.

- Time complexity = O(N+E);
  The time complexity for cycle detection is O(N+E). Checking the edge type in a cycle can be done by turning on a flag if the traverse passes a thin edge. Therefore, this checking does not require extra time complexity, and the time complexity is then O(N+E).

- Space complexity = O(N+E).
  The dimension graph consists of at most 2N nodes and 4E edges. In order to record the edge type, each edge may need an extra flag, which will add another E space. The total space required is roughly 2N+4E+E, which is O(N+E).

5 Consistency Checking for Constraints in General Format

In the previous sections, we described the dimension graph representation and the consistency checking algo-



(a) Dimension graph (b) X-graph after merging the cycle (c) MBR configuration

Figure 8. Dimension graphs for $North(A, B) \wedge South(B, A)$

rithms for conjunctive constraints among points, intervals, and MBRs. In this section, we extend the dimension graph based consistency checking algorithm to constraints in general format.

5.1 Dimension Graphs for Constraints in General Format

Constraints in general format can be transformed into Disjunctive Normal Form(DNF), which is a disjunction of conjunctions where no conjunction contains a disjunction. Each conjunction in the DNF constraint can be represented by a dimension graph by applying the algorithm **constructGraphPoint** or **constructGraphInterval**. The dimension graph of a general format constraint therefore is a collection of all the dimension graphs constructed from all its conjunctions. The number of graph sets(X/Y-graphs or interval graph) is the same as the number of conjunctions in the DNF.

(a) before(S,R) and meets(S, L) and overlaps(L,R) (b) metby(S,R) and meets(S, L) and overlaps(L,R)

Figure 10. Constraint graphs for (S meets L) and (S before or metby R) and (L overlaps R)

Figure 10 is an example of the dimension graph maintaining the constraint of *"S meets L and S before or metby R and L overlaps R"* for intervals S, L and R. The DNF format of the constraint is: $(before(S, R) \wedge meets(S, L) \wedge overlaps(L, R)) \vee (metby(S, R) \wedge meets(S, L) \wedge overlaps(L, R))$. The two conjunctions correspond to Figure 10(a) and (b) respectively.

5.2 Consistency Checking for General Constraints

After constructing the corresponding dimension graph for the constraints, the cycle detection function is performed on each subgraph representing a conjunctive constraint. If each set of the subgraphs contains cycles, the constraint is inconsistent. If a subset of graphs contains cycle, the constraint combinations corresponding to those graphs containing cycles are inconsistent. The combinations corresponding to the graphs without cycles are consistent. Algorithm 6 is the pseudo-code for consistency checking for general format constraints among points or intervals. The consistency checking algorithm contains three steps: Normalizing the constraints to standard DNF formats; constructing dimension graphs for each conjunction in DNF; performing cycle detection on each graph.

Algorithm 6 Consistency checking: **consistencyCheck**

Input: $constraint$ is the constraint needed to be checked
Output: TRUE if consistent, FALSE otherwise

```
consistencyCheck(Set of Predicates constraint) {
    DNF dnfConstraint= normalize(constraint);
    Set of Predicates consistentConstraint = ∅;
    for each conjunction p ∈ dnfConstraint {
        if objects are MBRs
            if MBRconjunctionConsistencyCheck(p)
                add p to consistentConstraint;
        else if conjunctionConsistencyCheck(p)
            add p to consistentConstraint;
    }
    if consistentConstraint == ∅
        return FALSE;
    return TRUE;
}
```

The subfunction **normalize** preprocesses the constraint and transforms the general format constraint into its DNF format. Secondly, the consistency for each conjunction of the DNF representation is checked by calling the Algorithm **conjunctionConsistencyCheck** or **MBRconjunctionConsistencyCheck** according to the type of object. If the conjunction constraint is consistent, the conjunction to consistentConstraint is added. The final result of consistentConstraint contains all the consistent constraint combinations, each of which represents a globally consistent constraint.

In the example of constraints among intervals as illustrated in Figure 10, the conjunction depicted in Figure 10(b) contains a cycle and hence the corresponding conjunction is inconsistent. Figure 11 shows the only consistent conjunction and one of its possible interval configurations.

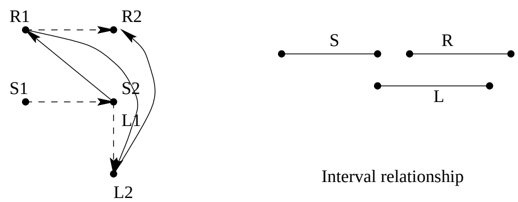


Figure 11. S meets L and S before R and L overlaps R)

Let N be the number of spatial objects and E be the number of spatial predicates being checked for consistency, and COR be the number of all possible combinations of disjunctive predicates, i.e., the number of conjunctions in DNF, the complexity of the algorithm can be summarized as follows:

- Time complexity = O(N+E)*O(COR);
  The consistency checking algorithm for general format constraints first generates the DNF for the constraints, and then calls the **conjunctionConsistencyCheck** algorithm for each conjunction. O(N+E) is the time for consistency checking for a conjunction constraint. Therefore, the total time bound is O(N+E)*O(COR).

- Space complexity = O(N+E).
  COR does not contribute to space factor, since graphs can be processed one at a time.

6 Conclusions and Future Work

In this paper, we propose a new strategy to process consistency checking for Euclidean spatial constraints among objects. We use a geometric approach by incorporating spatial domain information. Dimension graphs are proposed to maintain the spatial constraints among objects. Each conjunctive constraint is projected on both X and Y dimensions, and a graph is constructed for the constraint on each dimension. The spatial constraints in general format are maintained in a set of dimension graph constructed from their conjunctions. By using this framework, we transform constraint consistency checking into a graph cycle detection problem on dimension graph. Cycle detection can be solved efficiently with O(N+E) time as well as space complexity, where N is the number of spatial objects, and E is the number of spatial predicates in the constraint. The proposed approach to consistency checking for spatial constraints is faster when the number of predicates is much smaller than N^2 and there are few disjunctions in the spatial constraint. Since the algorithm returns TRUE if and only if the dimension graph of at least one conjunction contains no cycle, the algorithm guarantees global consistency.

In future work, we would like to explore consistency checking among mixed types of objects, e.g. consistency checking for constraints between points and intervals. We would also like to apply the dimension graph based cycle detection algorithm to image similarity-based image retrieval.

References

[1] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Domenico Beneventano and etc. Consistency Checking in Complex Object Database Schemata with Integrity. *IEEE Trans. on Knowledge and Data Eng.*, 10(4), July/August 1998.

[3] E.A. Boiten, J. Derrick, H. Bowman, and M.W.A. Steen. Constructive Consistency Checking for Partial Specification in Z. In *Science of Computer Programming*, number 1, pages 29–75, September 1999.

[4] H. Bowman, E.A.Boiten, J. Derrick, and M.Steen. Strategies for Consistency Checking Based on Unification. In *Science of Computer Programming*, pages 261–298, April 1999.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT press, McGraw-Hill Publishing Company, 1994.

[6] C. Freksa. Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*, 54:199–227, 1992.

[7] R.H. Güting. An Introduction to Spatial Database Systems. *VLDB Journal, Special issue on Spatial Database Systems*, 3(4):357–399, 1994.

[8] Daniel Hernandex. Maintaining Qualitative Spatial Knowledge. *Proc. of the European Conference on Spatial Information Theory, Elba, Italy*, pages 19–22, Sept. 1993.

[9] W. Kim, J. Garza, and A. Kesin. Spatial Data Management in Database Systems. In *Advances in Spatial Databases, 3rd International Symposium, SSD'93 Proceedings , Lecture notes in Computer Science, Vol. 692, Springer, ISBN 3-540-56869-7*, pages 1–13, Singapore, 1993.

[10] V. Kumar. Algorithms for constraint satisfication problems: A Survey. *AI Magazine*, 13(1):32–44, 1992.

[11] X. Liu, S. Shekhar, and S. Chawla. Consistency Checking for Euclidean Spatial COnstraints: A Dimension Graph Approach. Tech. Report TR00-039, University of Minnesota, Minneapolis, MN.

[12] S. Manandhar. Deterministic consistency checking of LP constraints. In *Proceesings of the 7th Conference of the European Chapter of the Association for Computional Linguistics*, pages 165–172, Dublin, Ireland, March 1995.

[13] P. Meseguer. Constraint satisfication problems: An overview. *AI Communications*, 2(1):3–17, 1989.

[14] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Inc., 1995.

[15] S. Shekhar, S. Ravada A.Fetterer, X.Liu, and C.T. Lu. Spatial Databases: Accomplishments and Research Needs. *IEEE Trans. Knowledge and Data Eng.*, 11(1):45–55, 1999.

[16] S. shekhar, X. Liu, and S. Chawla. Equivalence Classes of Direction Objects and Applications. Tech. Report TR99-027, University of Minnesota, Minneapolis, MN 55455.