TR 00-039

Consistency Checking for Euclidean Spatial Constraints: A Dimension Graph Approach

Xuan Liu, Shashi Shekhar, and Sanjay Chawla

July 11, 2000

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

Xuan Liu, Shashi Shekhar[‡], Sanjay Chawla
Computer Science Department, University of Minnesota
EE/CS 4-192, 200 Union St. SE., Minneapolis, MN 55455
telephone: (612)624-8307
$[xliu|shekhar|chawla]@cs.umn.edu$
http://www.cs.umn.edu/research/shashi-group

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 the dimension graph is constructed on each dimension. By using the dimension graph representation, the problem of consistency checking is then transformed to a graph cycle detection problem. The 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 to consistency checking for spatial constraints 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 2 dimensional space. The algorithm can guarantee the global consistency.

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

[*]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.

[†]Tech. Report TR00-039, University of Minnesota, Minneapolis, MN. Submitted to ICTAI

[‡]Contact author

1 Introduction

A spatial database [12, 14, 23] management system aims at the effective and efficient management of data related to a space such as the physical world (geography, urban planning, astronomy); parts of living organisms (anatomy of the human body); engineering design (very large scale integrated circuits, the design of an automobile or the molecular structure of a pharmaceutical drug); and conceptual information space (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.

Consistency checking is an important concept to maintain the integrity of databases in general and spatial databases in particular. Consistency checking is the process of identifying 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 in the context of spatial databases is more challenging *vis-a-vis* its traditional relational counterpart.

Consistency checking can also be used for spatial query processing via semantic query optimization[21] and reasoning for spatial qualitative relationships. For example, given a spatial query S and a set of of spatial constraints SC, if the query S is inconsistent with respect to SC then the answer to S is null. Currently most of consistency checking is based on Allens's algorithm [3] which was originally devised for checking temporal relationships on one-dimensional objects. Spatial relationships are typically formulated among multi-dimensional objects and straightforward extension of Allen algorithm for spatial relationships are prohibitively expensive.

In this paper, we address the problem of consistency checking for directional spatial constraints in the two-dimensional Euclidean space. We propose a dimension graph representation for maintaining the spatial constraints among objects. Basically, the spatial constraints is projected on each dimension(X and Y), the constraints that must be satisfied for each dimension(X/Y) are maintained in different graph(X/Y graph). One graph records the constraints on one dimension. The problem of constraint consistency checking is transformed to a graph cycle detection problem on the dimension graph. The cycle detection could 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 of system.

1.1 Spatial Data Model

Recent reports[12, 14, 23, 1] have described the accomplishments of spatial database research and have prioritized research needs. A broad survey of spatial database requirements and an overview of research results is provided by [23, 12, 1, 20].

A spatial data model is defined by geometric entities, spatial operations on these entities and spatial relationships between them. The three basic geometric entities are point, line and polygon which represent spatial objects in zero, one and two dimension respectively. To facilitate rapid query processing polygons are often represented by their Minimum Bounding Rectangles(MBRs). An MBR of a spatial object is the smallest axis parallel rectangle which covers the polygon.

The spatial relationships between the geometric entities are categorized according to the mathematical properties of the relationships. For example, topological relationships like *overlap* and *adjacent* are relationships which are invariant under a change of coordinate or projection system. Other types of relationships include *Metric*, *Set* and *Directional.* Some important relationships categories and their representative example are shown in 1. In this paper we will focus on consistency checking of absolute directional relationships like North, South, East, West, Northeast, Northwest, Southeast and Southwest. These relationships can be defined between any combination of point, line and polygon objects but we will restrict our attention to homogeneous point, line and MBR pairs. For constraints among intervals, we deal with Allen's 13 relationships [3]. The constraints are represented in terms of disjunctions and/or conjunctions of spatial predicates. The predicates are logically atomic.

Data model	Operator Group	Operation
Vector Object	Set-Oriented	equals, is a member of, is empty, is a subset of, is disjoint from, intersection, union, difference, cardinality
	Topological	boundary, interior, closure, meets, overlaps, is inside, covers, connected, components, extremes, is within
	Metric	distance, bearing/angle, length, area, perimeter.
	Direction	east, north, left, above, between.
	Network	successors, ancestors, connected, shortest-path

Table 1: A Sample of Spatial Operations

1.2 Problem Definition

In this paper, we intend to explore the consistency checking for Euclidean qualitative spatial constraints among points, intervals, and 2-D Minimum Bounding Rectangles(MBRs) objects in spatial database.

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 computation complexity

Constraint: (a) 2-D Euclidean space.

(b) Two-dimensional extended objects are approximated by MBRs

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

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

Consider an example of direction 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.

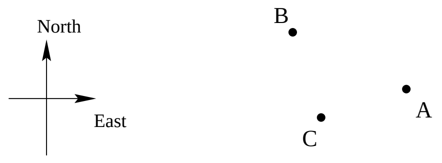


Figure 1: One possible spatial configuration for point objects A, B, and C

Suppose we have another set of 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.

1.3 Related work and our contributions

Most of previous study on consistency checking is based on Allen's consistency checking algorithm [3] for constraints among intervals. The basic approach for consistency checking is to use transitive closure algorithm, which incurs high order of time and space complexity. Hernandez [13] presented mechanisms to maintain the consistency of a knowledge base of spatial information based on a qualitative representation of 2-D positions. His approach improved Allen's algorithm by using the heuristic of rich structure of the spatial domain. Bowman and etc.[6, 5] addressed the problem of consistency checking between multiple viewpoints using strategies based on unification. Some other work[15, 10, 17] focused on the problem of consistency checking for more general constraints, such as, 1-th order constraints. Manandhar[16] discussed deterministic consistency checking for LP constraints. Beneventano and etc.[4] focused on the problem of providing a theoretical framework for consistency checking of integrity constraints in a complex object database environment.

The consistency checking algorithm is dependent on the set of spatial predicates. This paper, we deal with basic direction predicates for absolute directions[22]. There are some other directional predicates. The research work on direction modeling has been carried out in several areas such as geographic information systems and image analysis. Most of the studies is on how to capture the semantics of direction relations, and further, how to do spatial reasoning based on direction predicates [8, 9, 11]. There are two major direction reference frames used to model direction in 2D space: the cone-based model[18], and the projection-based model[9, 11]. Frank[2] compared these two models and found the projection-based reference frame to be better in many aspects. The most common way to model directions between extended objects is through the object's Minimum Bounding Rectangle(MBR), where direction relations are obtained by applying Allen's [3] interval relations along the x and y axis, in which case, 169 different relations[8] can be distinguished. We'll briefly describe Allen's Algorithm since it is basic to many algorithms.

1.3.1 Allen's Propagation Algorithm

Allen[3] summarized thirteen mutually exclusive relationships to express any possible relationships between intervals as shown in table 2.

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 2: Thirteen possible relationships proposed by Allen[1]

In Allen's work[3], the relationships between intervals are maintained in a network where each node N_i represents 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 he used for maintaining relationships was propagating new relationships by computing the transitive closure of the relationships between intervals.

Figure 2 shows the 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

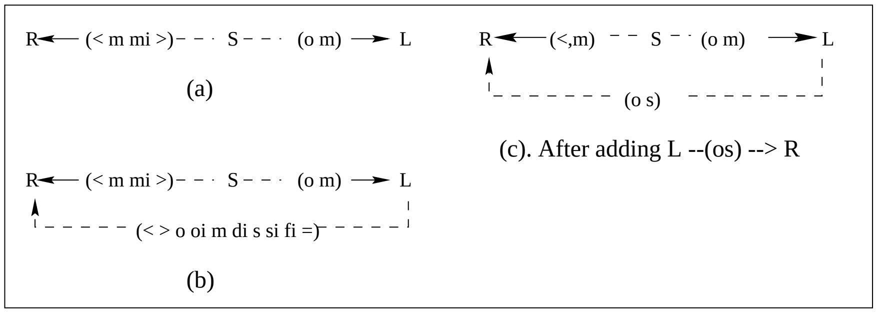

R ← (< m mi >) - - S - - (o m) → L
(a)
R ← (< m mi >) - - S - - (o m) → L
(< > o oi m di s si fi =)
(b)
R ← (<,m) - - S - - (o m) → L
(o s)
(c). After adding L --(os) --> R

Figure 2: Examples of Allen's algorithm

was added, the algorithm computed 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 R*, we need to propagate its effect through the network, and thus obtaining the resulting network 2(c).

As explained in Allen's paper[3], 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 $O(N^2)$.

As Allen stated in [3], one problem with this algorithm is that it does not detect all inconsistencies in its input. Quote the phrases in [3] "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 can not guarantee global consistency.

1.3.2 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 the spatial domain information in consistency checking. We propose dimension graphs to maintain the spatial constraints among objects. Each conjunctive constraint is projected on both dimensions, and a dimension graph is constructed for the constraint on each dimension. For spatial constraints in general format, the constraints can be converted to its Disjunctive Normal Form(DNF)[19] , and dimension graphs are constructed for each conjunction. By using the dimension graph representation, the problem of constraint consistency checking is then transformed to a graph cycle detection problem on each dimension graph. The cycle detection could 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 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. The dimension graph and consistency checking algorithm can be used for points, and MBRs in 2 dimensional space. Since the algorithm returns TRUE if and only if the dimension graph of at least one conjunction contains no cycle, which means there exist at least one consistent overall constraint. The algorithm can guarantee the global consistency.

1.4 Scope and Outline

In this paper, we address the problem of consistency checking for spatial constraints in the two-dimensional Euclidean space. We deal with only 0-th order constraints. We focus on the qualitative constraints among objects, which include topological and direction relationships(See Table 1). Distance-based constraints are not discussed. We consider the constraints among point objects, interval objects, and region objects approximated by MBRs. We focus on addressing the consistency checking for homogeneous types of objects, i.e., the constraints among point objects, or intervals, or MBRs. The consistency checking for the constraints specified among mixed types of objects(e.g. the constraints between a point and an interval) is out of the scope of the paper, which may be addressed in the future work. We only discuss a specific set of predicates defined in the paper. The predicates are defined in Euclidean space. Different predicate sets or in different space may be different. Some constraints that are inconsistent in Euclidean space may be consistent in spherical space. The consistency checking for predicates in other space is out of the scope of this paper.

The organization of this paper is as follows: In section 2, we propose dimension graph representation for the conjunctive constraints among points, intervals. The consistency checking for conjunctive constraint based on the dimension graph representation is introduced in section 3. In section 4, we discussed the dimension graph construction and consistency checking for conjunctive constraints among MBRs. Finally, the 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 will describe the construction of dimension graph for conjunctive spatial constraints among points and intervals. The basic idea is to projected the conjunctive spatial constraint onto each dimension and record the spatial constraints that much be satisfied on each dimension in a dimension graph respectively. The dimension graph for points in 2D space contains X-graph and Y-graph. The dimension graph for intervals in 1D space contains only one graph. We also analyze the computation complexity for the dimension graph construction algorithm.

2.1 Dimension Graphs for Constraints Among Point Objects

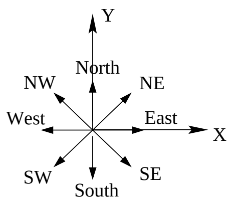
We start by defining a set of absolute direction predicates for point objects in terms of coordinates. Here, We assume the global coordinate systems aligned with the reference frame of absolute directions, i.e., $North$ aligns with y-axis, and $East$ align with x-axis. The definition for each predicate is given in Table 3. The first column of the table enumerates the direction predicates. The second and third columns

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

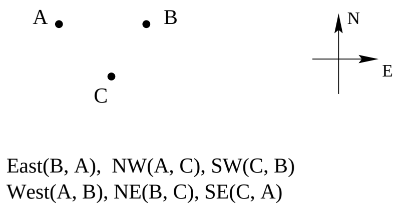
Table 3: Direction Predicates for point objects in terms of coordinates

represent the relationships between two point objects on X and Y dimensions respectively. A_y, B_y are the y-components of A and B, and A_x, B_x are x-components of A and B. Figure 3(a) illustrates the intuition of the definition. The predicates are defined using direction equivalence classes [22] 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. This predicate set is complete. Figure 3(b) shows examples of predicates described 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 conjunctions of predicates can be maintained in two graphs: X-graph and Y-graph. The nodes in both graphs represent the objects forming the constraints.



(a) Definitions of the predicates



A
B
C
N
E
East(B, A), NW(A, C), SW(C, B)
West(A, B), NE(B, C), SE(C, A)

(b) Examples of predicates

Figure 3: Illustration of the predicates

The direction constraints are represented as directed edges in each graph according to the symbol in the column 2 and 3 of Table 3. The edge goes from the node with smaller values to the node with larger values. If the symbol is '=', the two nodes are merged to one node. Figure 4 shows the graph representation

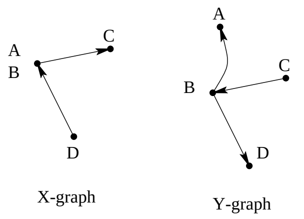


Figure 4: Dimension graph for North(A,B) ∧ NW(B,C) ∧ SE(B,D)

for the constraint North(A,B) ∧ NW(B,C) ∧ SE(B,D). The dimension graph is a union of X-graph and Y-graph, and is constructed according to the definition of each predicate in Table 3. Algorithm 1 is the pseudo-code of the graph constructing procedure. The input of the algorithm is the conjunctive constraint and the output is the dimension graph. For each predicate in the conjunctionConstraint, the algorithm invokes the sub-function **add_a_predicate_point** to add the predicate to the dimension graph. This function calls function *addNode* to add nodes that do not in the dimension graph into the graph, and calls function *addEdge* to add the spatial relationships between nodes into dimension graph.

We can easily summarize the computation 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 could also be accomplished in constant time. The whole algorithm therefore have the time bound of O(E). The generated graph has at most N nodes and E edges each in X-graph and Y-graph.

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

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

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

Graph constructGraphPoint(Set of Predicates conjunctionConstraint) {
    Graph constraintGraph = $\emptyset$;
    for each entry $p \in$ conjunctionConstraint
        add_a_predicate_point($p$, &constraintGraph);
    return constraintGraph;
}

add_a_predicate_point(Predicate aPredicate, Graph* aGraph ) {
    firstObject = getFirstObject($p$);
    secondObject = getSecondObject($p$);
    addNode(firstObject, secondObject, aGraph.graphX);
    symbol =findXconstraint(Table 3, p);
    addEdge(symbol, firstObject, secondObject, aGraph.graphX);
    addNode(firstObject, secondObject, aGraph.graphY);
    symbol =findYconstraint(Table 3, p);
    addEdge(symbol, firstObject, secondObject, aGraph.graphY);
}

addNode(Node n1, Node n2, Graph aGraph) {
    if n1 $\notin$ aGraph        Add n1 to aGraph;
    if n2 $\notin$ aGraph        add n2 to aGraph;
}

addEdge(char symbol, Node n1, Node n2, Graph, aGraph) {
    if (symbol == '=')
        merge nodes n1, n2 to one;
    else if (symbol == '<')
        add directed edge of (n1, n2) to aGraph;
    else
        add directed edge of (n1, n2) to aGraph;
}
```

2.2 Dimension Graph for Conjunctive Spatial Constraints Among Intervals

The relationships between intervals proposed by Allen [3] can be defined in terms of the endpoints of intervals. Table 4 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.

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 4. The edge point to the nodes with larger values from the nodes with smaller values. The nodes with 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 $before(S, R) \wedge meets(S, L)$.

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 4: Spatial relationships for intervals, where $<$ and bi describe directional relationships and others are topological relationships

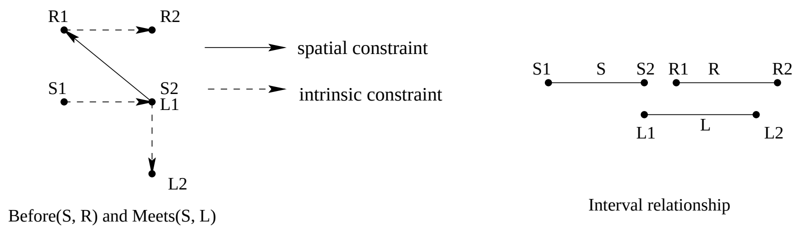


Figure 5: $before(S,R) \wedge meets(S,L)$

The graph is constructed according to the definition of each predicate in Table 3. The dashed arrow represents the intrinsic constraint of the start point and end points. The pseudo-code of the graph constructing procedure is given as in Algorithm 2.

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

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

- Time complexity = O(E);
  According to table 4, there are at most three point predicates should be satisfied for each interval predicate, and hence at most three edges added for each interval predicate. The time complexity for this algorithm is therefore O(E).

- Space complexity O(N+E).
  The resulting constraintGraph consists of at most 2N nodes and 3E edges. The space requirement is linear to the number of nodes and edges, roughly 2N+3E, which is essentially O(N+E).

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

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

Graph constructGraphInterval(Set of Predicates conjunctionConstraint) {
    constraintGraph = $\emptyset$;
    for each entry $p \in$ conjunctionConstraint
        add_a_predicate_interval($p$, &constraintGraph);
    return constraintGraph;
}

add_a_predicate_interval(Predicate $p$ Graph* aGraph ) {
    firstObject = getFirstObject($p$);
    secondObject = getSecondObject($p$);
    addIntervalNode(firstObject, secondObject, constraintGraph);
    pointPredicates = convertToPoint($p$); //according to Table 4
    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;
    }
}
```

3 Consistency checking for conjunctive constraints

In the previous section, we construct dimension graph for conjunctive Euclidean spatial constraints among point objects or intervals. In this section, we will 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 represented by dot line is added into the graph. As can be

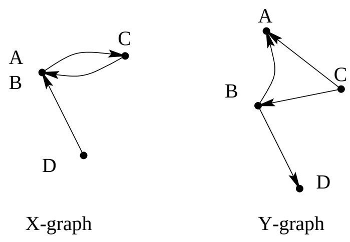


Figure 6: North(A,B) ∧ NW(B,C) ∧ SE(B,D)∧ NE(A,C)

seen, a cycle is constructed in the X dimension graph in Figure 6. 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 is smaller than the x-value of C, and at the same time, the x-value of C is 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:

$\Longrightarrow$: Suppose that the dimension graph contains no cycle, i.e., the graph is a directed acyclic graph(dag), we can construct a topological sort of the dag [7] using depth-first search algorithm. A topological sort of a dag is a linear ordering of all its nodes such that if the dag contains an edge (u, v), then u appears before v in the ordering. In other words, there exists at least one spatial configuration satisfying the constraints, the constraint is consistent.

$\Longleftarrow$: Suppose that the dimension graph contains a cycle. For the nodes involved in the cycle, there exist no topological sort, i.e., no linear ordering of the nodes is possible. In other words, we can not give a global labeling for the nodes to meet the requirement that all directed edges go from left to right. No spatil configuration can satisfy the constrain. Therefore, the constraint is inconsistent. ■

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. The consistency checking for conjunctive constraints contains two steps: 1. Construct the corresponding dimension graph; 2. Perform cycle detection on each graph. The constraint is consistent if none of the graph contains cycle. The pseudo-code is described as in Algorithm 3.

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;
}
```

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

A nice property of this algorithm is its efficiency. The consistency checking is just a graph cycle detection which can easily be done in linear time. As in 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[7]. As we explained in section 2, each of 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 same as O(N+E).

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

3.2 Examples

Consider the example of spatial constraints among intervals:

(S meets L) and (S is metby R).

The consistency checking for this constraint is then accomplished by two steps. The function **constructGraphInterval** is first invoked to construct the corresponding dimension graph, which is illustrated in Figure 7 (a). The **detectCycle** function checks the cycle in the graph. Since there is no cycle found,

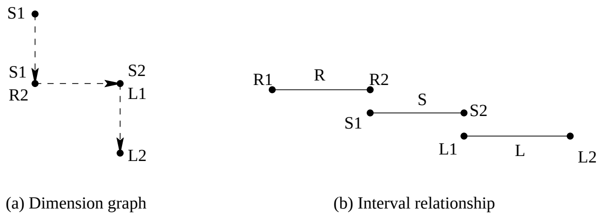


Figure 7: Dimension Graph for $meets(S, L) \wedge metby(S, R)$

the constraint is consistent. We can construct an interval configuration which satisfies the constraint as in Figure 7 (b).

Assume a new fact $overlaps(L, R)$ is added into the system. We call the function of **add_a_predicate_interval** to add this new constraint to the dimension graph, resulting in a new dimension graph as in Figure 8.

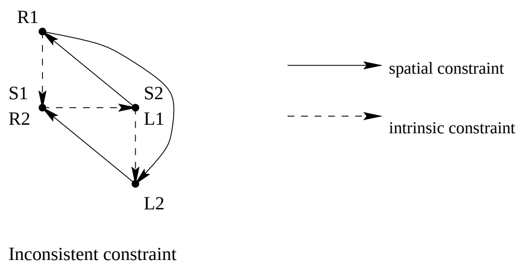


Figure 8: Dimension Graph after adding overlaps(L,R)

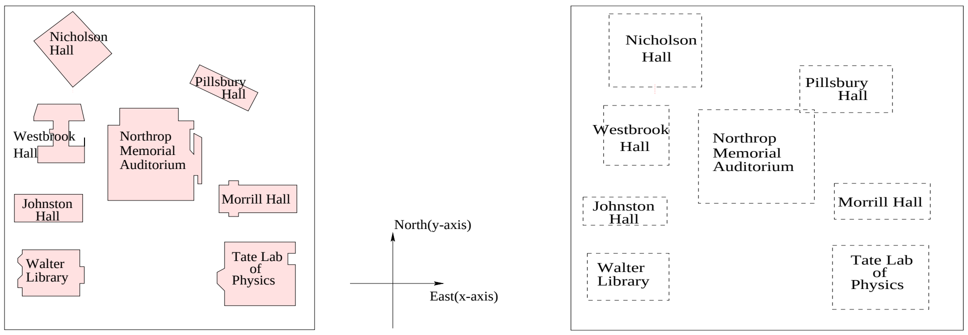
The cycle detection is performed on the new dimension graph to check consistency of the status. Since cycles of (R1, S1, L1, R1) and (R2, L1, L2, R2) are detected in the figure as shown. The new constraint added is not consistent with the system, therefore, the new constraint is inconsistent. There exist no interval configuration satisfying this constraint.

Since the consistency checking is performed based on the cycle detection of the corresponding dimension graph, the algorithm can always detect inconsistent constraints. As we stated in Theorem 1, for any dimension graph without cycle, there is a consistent configuration among objects. The algorithm can guarantee global consistency.

4 Consistency Checking for Conjunctive Constraints Among MBRs

In the previous section, we discuss the dimension graphs for conjunctive constraints among 2D points and 1D intervals, and also describe the dimension graph based consistency checking algorithm. In this section, we will extend our dimension graph based approach to 2D spatial objects approximated by MBRs. Examples of 2D spatial objects includes 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. Figure 9(a) shows a small portion of the campus maps of University of Minnesota. Figure 9(b) replaces all the buildings in Figure 9(a) by their corresponding $MBRs$.



(a) Campus map of University of Minnesota

(b) Approximate buildings by MBRs

Figure 9: 2D region objects vs. MBRs

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 lower-left and upper-right corners of MBR A.

The direction relationship between MBRs can be determined by the relationships between the representative points. Table 5 shows the definitions of the direction predicates based on the representative points of MBR of the objects. The first column of the table enumerates the direction predicates. The

Direction 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 5: Direction Predicates for MBR

second column gives the constraints that should be satisfied by their representative points.

For example, in Figure 9(b), MBR of " Morrill Hall" is north of MBR of "Tate Lab", and the MBR of "Pillsbury hall" is northeast of the MBR of "Northrop Auditorium". The same relationships hold for buildings in Figure 9(a). These are described by predicates North(Morrill Hall, Tate Lab), and NE(Pillsbury hall, Northrop Auditorium).

4.1 Dimension Graphs for Conjunctive Constraints Among MBRs

The dimension graph for each direction predicate contains X-graph and Y-graph which maintain the constraints that must be satisfied on X and Y dimension 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 5. Similar to the X-graph and Y-graph for points, an directed edge goes from the node with smaller values to the node with larger values. If the value associated with two nodes are same, the two nodes are merged to one node. We introduce a new '**thick arrow**' edge here to represent 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 intrinsic constraint between the lower-left point and the upper-right point of a MBR.

The dimension graph for a conjunctive constraint is constructed by adding the constraints specified in each predicate in X-graph and Y-graph. Figure 10(a) and (b) shows an example of dimension graph for a conjunctive constraint of $North(A,B) \wedge NE(B,C) \wedge SW(C,A)$. Figure 10(c) is a possible spatial configuration among MBR A, B and C.

We now describe the dimension graph construction for conjunctive constraints for MBRs. The pseudo-code of the graph constructing procedure is given as 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 to point predicate set, and add each point predicate into the graph. To accomplish this, *addNode* and *addEdge* are invoked.

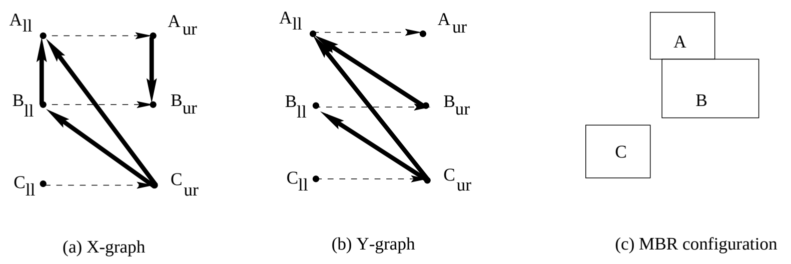


Figure 10: Example MBR configuration and its corresponding dimension graphs, thick arrow represents $\leq$, thin arrow represents $<$

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

- Time complexity = O(E);
  According to table 5, there are 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 therefore 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 cycle in the dimension graphs. There are three possible situations and corresponding results:

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

Case 2: Every cycle consists of only thick edge (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 exist a consistent spatial configuration among all objects. Figure 10 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 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 11 shows such an example. Figure 11 (a) is the dimension graph for constraint $North(A, B) \wedge South(B, A)$, it contains a cycle with only thick edges in X-graph. Figure 11 (b) is the transformed X-graph after merging each cycle to one node. There is no cycle in this resulting graph. This constraint is obviously consistent. Figure 11 (c) is a sample configuration.

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 = $\emptyset$;
    for each entry $p \in$ conjunctionConstraint
        add_a_predicate_MBR($p$, &constraintGraph);
    return constraintGraph;
}

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

addEdge(Point Predicate $pp$, Graph* aGraph ) {
    $r(k,l)$= getRelationSymbol($pp$);
    if ($r ==' <'$) add directed edge $(k,l)$ to aGraph;
    if ($r ==' >'$) add directed edge $(l,k)$ to aGraph;
    if ($r ==' ='$) merge node $k$ and $l$ in aGraph;
    if ($r ==' \leq'$) add thick directed edge $(k,l)$ to aGraph;
    if ($r ==' \geq'$) add thick directed edge $(l,k)$ to aGraph;
}
```

Case 3 is easy to understand. If there is a cycle containing at least one non-thick edges, no matter how we label the thick edges in the cycle, we can not remove the cycle. There must be conflicts among coordinates of the objects involved in the cycle, and hence, the constraint is inconsistent. Figure 12 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 we can noticed, there exist cycles in both X and Y dimension graphs of the constraint. No spatial configuration of A, B, C can satisfy this constraint. The constraint is inconsistent.

Based on the above arguments, we can now describe the consistency checking algorithm for MBRs as in Algorithm 5. The algorithm first calls function **constructGraphMbr** to construct the dimension

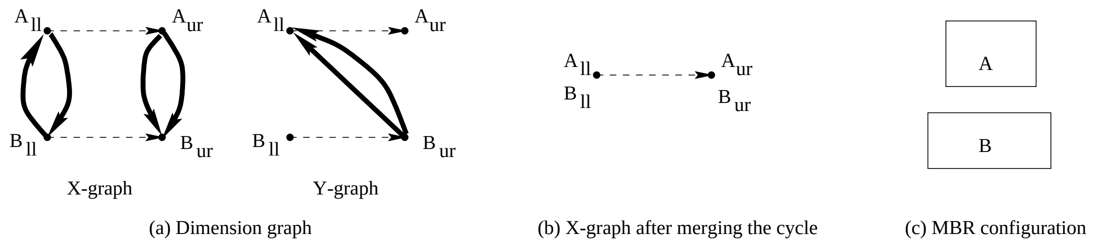


Figure 11: Dimension graphs for $North(A,B) \wedge South(B,A)$

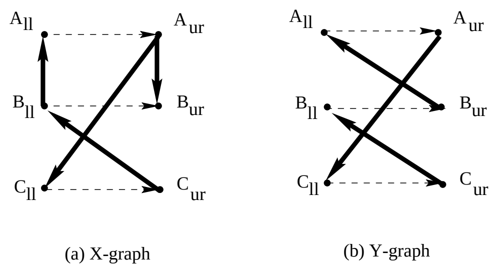


Figure 12: Dimension graphs for $North(A,B) \wedge NE(B,C) \wedge SW(A,C)$

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 conjunctionConstraint) {
    Graph constraintGraph = $\emptyset$;
    constraintGraph = constructGraphMbr (conjunctionConstraint);
    if !detectCycle(constraintGraph)
        return TRUE;
    else if the cycle contains thin edge
        return FALSE;
    else return TRUE;
}
```

graph, and then use **detectCycle** to detect cycles in the dimension graph. It returns TRUE if no cycle is detected. If the detected cycle contains thin edges, the algorithm returns FALSE, otherwise, returns 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 easily by turning on a flag if the traverse passes a thin edge. Therefore, this checking does not require extra time complexity, 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 describe the dimension graph representation and the consistency checking algorithms for conjunctive constraints among points, intervals and MBRs. In this section, we will extend the dimension graph based consistency checking algorithm to the constraints in general format.

5.1 Dimension Graphs for Constraints in General Format

The 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 the general format constraint therefore are 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.

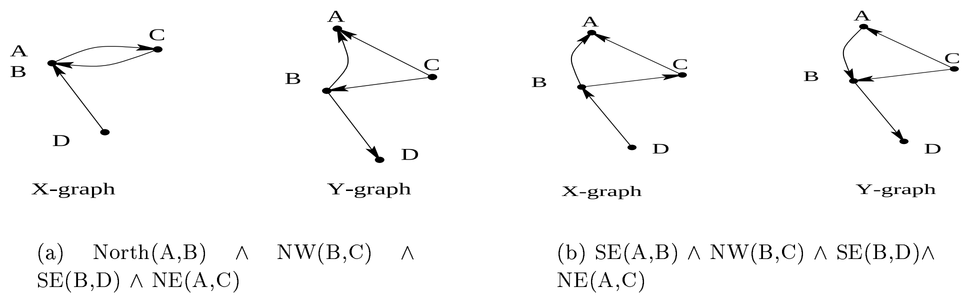

A
B
C
D
X-graph
A
B
C
D
Y-graph
(a) North(A,B) ∧ NW(B,C) ∧ SE(B,D) ∧ NE(A,C)
A
B
C
D
X-graph
A
B
C
D
Y-graph
(b) SE(A,B) ∧ NW(B,C) ∧ SE(B,D)∧ NE(A,C)

Figure 13: X/Y-graphs for $(North(A,B) \vee SE(A,B))$ and $NW(B,C)$ and $SE(B,D)$and $NE(A,C)$

Figure 13 is an example of the dimension graph maintaining the constraint of $North(A,B) \vee SE(A,B)$ and $NW(B,C)$ and $SE(B,D)$and $NE(A,C)$ for point objects A, B, C and D. The DNF format of the constraint is: (North(A,B) ∧ NW(B,C) ∧ SE(B,D) ∧ NE(A,C)) ∨ (SE(A,B) ∧ NW(B,C) ∧ SE(B,D) ∧ NE(A,C)). Figure 13(a) and (b) are X/Y-graphs for the two conjunctions, which are $(North(A,B) \wedge NW(B,C) \wedge SE(B,D) \wedge NE(A,C))$ and $(SE(A,B) \wedge NW(B,C) \wedge SE(B,D) \wedge NE(A,C))$.

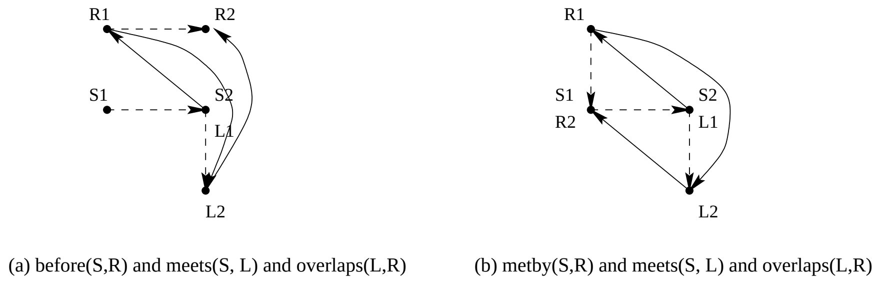

R1
R2
S1
S2
L1
L2
(a) before(S,R) and meets(S, L) and overlaps(L,R)
R1
S1
R2
S2
L1
L2
(b) metby(S,R) and meets(S, L) and overlaps(L,R)

Figure 14: Constraint graphs for (S meets L) and (S before or metby R) and (L overlaps R)

Figure 14 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 14(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 subgraph contains cycle, 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; Construct dimension graphs for each conjunction in DNF; Perform cycle detection on each graph.

Algorithm 6 Consistency checking: **consistencyCheck**

```
Input: constraint is the constraint need to be checked
Output: TRUE if consistent, FALSE otherwise

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

The subfunction **normalize** preprocesses the constraint and transform the general format constraint into its DNF format. Secondly, check the consistency for each conjunction of the DNF representation by calling the Algorithm **conjunctionConsistencyCheck** or **MBRconjunctionConsistencyCheck** according to the type of the objects. If the conjunction constraint is consistent, add the conjunction to consistentConstraint. The final result of consistentConstraint contains all the consistent constraint combination, each of which represents a global consistent constraint.

In the example of constraint among point A, B, C and D shown in Figure 13, the dimension graph corresponding to conjunction North(A,B) $\wedge$ NW(B,C) $\wedge$ SE(B,D) $\wedge$ NE(A,C) contains a cycle, and hence this conjunction constraint is inconsistent. The consistentConstraint will only include the consistent conjunction, which is SE(A,B) $\wedge$ NW(B,C) $\wedge$ SE(B,D)$\wedge$ NE(A,C)(Figure 13(b)). Similarly, in the example for intervals illustrated in Figure 14, the conjunction depicted in Figure 14(b) contains cycle and hence the corresponding conjunction is inconsistent. Figure 15 shows the only consistent conjunction and one of its possible interval configurations.

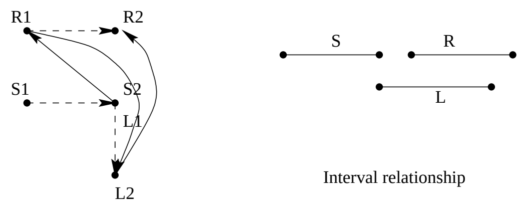


Figure 15: S meets L and S before R and L overlaps R)

Complexity Analysis

Similarly, 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.

- Time complexity = O(N+E)*O(COR);
  The consistency checking algorithm for general format constraints first generates the DNF for the constraints, and then call 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 one may process one graph 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 the spatial domain information in consistency checking. Dimension graphs are proposed to maintain the spatial constraints among objects. Each conjunctive constraint is projected on both dimensions, and a dimension 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 its conjunctions. By using this framework, constraint consistency checking is then transformed to a graph cycle detection problem on its dimension graph. The cycle detection could 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 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. The dimension graph and consistency checking algorithm can be used for points, and MBRs in 2 dimensional space. Since the algorithm returns TRUE if and only if the dimension graph of at least one conjunction contains no cycle, which means there exist at least one consistent overall constraint. The algorithm guarantees the global consistency.

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

References

[1] N. Adam and A. Gangopadhyay. *Database issues in Geographical Information Systems.* Kluwer Academics, 1997.

[2] A.Frank. Qualitative Spatial Reasoning: Cardinal Directions as an Example. *International Journal of Geographical Information Systems*, 10(3):269–290, 1996.

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

[4] 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.

[5] 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.

[6] 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.

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

[8] D.Papadias, M. Egenhofer, and J. Sharma. Hierarchical Reasoning about Direction Relations. In *Fourth ACM Workshop on Advances in Geographic Information Systems*, pages 105–112. ACM, 1996.

[9] A. Frank. Qualitative Spatial Reasoning about Cardinal Directions. In *Auto carto 10, D.Mark and D. White, eds., Baltimore, MD*, pages 148–167, 1991.

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

[11] C. Freksa. Using Orientation Information for Qualitative Spatial Reasoning. *Theories and Methods of Spatio-Temporal Reasoning Geographic Space*, 639:162–178, 1992.

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

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

[14] 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.

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

[16] 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.

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

[18] Donna J. Peuquet and Zhan Ci-Xiang. An Algorithm to Determine the Directional Relationship Between Arbitrarily-shaped Polygons in the plane. *Pattern Recognition*, 20(1):65–74, 1987.

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

[20] 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.

[21] S. Shekhar and B. Hamidzadeh. Learning Transformation Rules for Semantic Query Optimization: A Data-Driven Approach. *IEEE Trans. Knowledge and Data Eng.(Spatial Issue on Discovery in Databases)*, October 1993.

[22] Shashi shekhar, Xuan Liu, and Sanjay Chawla. Equivalence Classes of Direction Objects and Applications. Tech. Report TR99-027, University of Minnesota, Minneapolis, MN 55455.

[23] M.F. Worboys. *GIS: A Computing Perspective.* Taylor and Francis, 1995.