Spatial-reasoning for Agents in Multiple Dimensions

Debasis Mitra
Department of Computer Sciences
Florida Institute of Technology
Melbourne, Florida, USA
dmitra@cs.fit.edu

Gerard Ligozat
LIMSI/CNRS, Universite de Paris-Sud
ORSAY Cedex, France
ligozat@limsi.fr

Abstract: Suppose a group of mobile agents situated in some Euclidean space does not have any idea on where they are exactly located within that space. However, they do have some notion about their relative positions with respect to each other. This problem may be formulated as a multi-dimensional point-based qualitative reasoning problem with disjunctive constraints. In this article we have developed a set of incremental algorithms for finding feasible positions of a new agent relative to the other existing agents (located in 1D, 2D and the generalized d-D dimensional space for d>=1), given some qualitative spatial constraints between the new one and the other agents. Our approach is a *domain-theoretic* one, similar to that used in the traditional constraint-based reasoning works (CSP). This approach differs from the algebraic approach - that is traditionally deployed in the spatio-temporal reasoning areas. We have also obtained some tractability results here for the full binary constraint satisfaction problem (rather than the incremental problem, which is polynomial) based on a notion of strong pre-convexity. The article also hints toward many future directions for this work.

ACM Category: I.2.4: Knowledge Representation Formalisms and Methods

Key Words: Multi-dimensional spatial reasoning, Reasoning with disjunction, Qualitative reasoning, Multi-agent reasoning

1 Introduction

Consider a situation where some of the agents are cell locations (as in mobile telephone communications) and others are mobile computers radio-connected via those communication-cells. A mobile agent in a real space may have to find its position from relative spatial constraints with respect to other agents. For example, in a 2-dimensional space one could think of nine such relative positions: {East, North, West, South, Northeast, Northwest, Southwest, Southeast, SameSpot}.

Often an intelligent agent has to work under a noisy environment also. Within the field of Artificial Intelligence two types of noises are typically addressed: incompleteness and uncertainty. Incompleteness of information appears when it is disjunctive, e.g. agent A is located {East, Northeast, or North} of agent B. Uncertainty may be represented with additional certainty parameters over each of the

disjunctive elements. In this paper we will address some of the issues in incomplete qualitative (or relative) spatial reasoning in a multi-dimensional (d-D) Cartesian space. For an overview of spatial knowledge representation and reasoning with disjunctions see Cohn and Hazarika [2001].

It is difficult to provide appropriate syntax for the basic regions in a space with more than two dimensions. For a space with dimension d, the number of such basic regions is 3^d. For example, one can easily verify that this number to be 27 for a 3D case. In this paper we will desist from using any natural language elements for this reason (e.g., for the 2D case the elements are East, West, etc.), rather we will address the problem of reasoning from an abstract point of view. Although in reality it is difficult to perceive any agent's location in a space beyond 4 dimensions (e.g. submarines moving in 3D space and in time), our treatment here is targeted to any arbitrary number of dimensions.

The following section (2) introduces the incremental problem of adding a new point in a space with existing points and with a set of qualitative relations with the new point. Section 3 and 4 describe the problems of doing such reasoning in 1D and 2D space respectively. Section 5 generalizes them into a d-D space for arbitrary integer value of d. The two subsequent sections (6 and 7) delve into the incremental reasoning and into solving the full constraint-satisfaction problem (CSP) respectively in the d-D space. A separate section attempts to create two application scenarios where such constraint propagation techniques could be deployed. The article concludes with some discussion on our results and on some possible future directions of the work.

2 The Incremental Problem

A typical binary constraint satisfaction problem (BCSP) has a set of variables and some constraints between some pairs of them specifying what are the allowed pairs of values between the respective variables of each of these constrained pairs. The decision problem in BCSP is to check whether the variables may have any satisfiable assignments or not following the constraints. In the case addressed here, the variables are the points (agents) in a d-D space (d>=1) and the constraints are disjunctive subsets of the set of basic relations. However, the problems we are addressing in this article are not only the full BCSP version in the domain (points in d-D space), but also an incremental version of it.

In the incremental problem we have (n-1) variables (point-agents) in a space having a complete (non-disjunctive) relationship with respect to each other. The variables' exact coordinate may not be provided (hence, qualitative reasoning). A new variable, the n-th one, needs to be entered in this space. Given are the disjunctive relations between the n-th variable and each of the (n-1) variables in the space. The decision problem is to find whether there exists any set of valid regions for the n-th variable in the space satisfying the constraints or not. Note that, in lack of quantitative values, the regions are specified only relative to the existing (n-1) points. In case the answer to the decision-question in the problem is "True," we would also like to know the set of valid regions for the n-th point. The reader may like to take a look at the

Examples 3 and 4 for the 2D case in a subsequent section, for a clearer picture of this problem.

The next step to solving such an incremental problem is obviously (for the user) to commit to one of the valid regions, where n would be located. This is equivalent to choosing one of the basic relations with respect to each of the (n-1) variables. After that, the space will contain n variables (point-agents), fixed relative to each other. In the next cycle the (n+1)-th variable would be introduced in the space in a similar fashion. Incremental problem is solved by an incremental algorithm. Such an incremental algorithm could be then used for solving a full BCSP, although that is not necessary. The full BCSP may use some *short cut* algorithms also without utilizing any incremental algorithm.

Incremental version of the d-D point-based reasoning, as stated above, is polynomially solvable. This is because the number of total regions grows polynomially with respect to the number of points existing in the space (it is easy to check that), and finding a valid region for the new point may constitute checking over all of these regions (in absence of any heuristics, in the worst case). However, it is obvious that for solving the full BCSP one may have to run the incremental algorithm exponential number of times with backtracking. The full BCSP is NP-hard (true for d>1).

3 Reasoning in 1-D Space

3.1 The Language

The simplest case is that for 1-D, which is a case of well-known *point algebra* [Vilain and Kautz, 1986]. The language for the point-based reasoning in 1D is designed to express constraints between a node A_n with some other nodes, say, A_1, A_2, … A_{n-1}. The set is $\{(A_n\ R_i\ A_i) \mid 1 <= i <= n-1\}$, where any relation R_i is a non-null disjunctive subset of the set of qualitative relations between a pair of points {<, >, =}. Thus, R_i is an element of the set {"<", "<=", ">", ">=", "=", "<>", "<=>"}. The first four relations indicate in which direction, left or right, the new point. A_n should lie with respect to the point A_i. The equality is a strict relation making A_n coincide with A_i, and thus, be ignored, subject to pointers between the two equivalent points (say, for the purpose of answering any query about A_n). The last two relations cause A_n to ignore A_i in the process of finding its position in the sequence. This issue will be further explained later.

A set of valid regions for A_n would be expressed as: $\{(x_{j1}, x_{j2}) \mid 1<=j<= m\}$, m being the total number of valid regions, with m <= 2n-1, since the total number of available regions is 2(n-1)+1 = 2n-1. Typically for any region: $x_{j1} < x_{j2}$. However, it is possible to have a region with $x_{j1} = x_{j2}$, which means that the valid region is a point only.

Example 1:
Input: S = {3, 7, 9, 11, 15, 18, 22}, R = {(A_n >= 3), (A_n <=> 7), (A_n >= 9), (A_n <> 11), (A_n < 15), (A_n <> 18), (A_n <= 22)}.

Output: ValidRegSet = {(9, 9), (9, 11), (11, 15)}. The *box* (see Lemma 1), which is not necessarily a complete valid region, here is [9, 15].

Example 2:
Input: S = {3, 7, 9, 11, 15, 18, 22}, R = {(A_n >= 3), (A_n <=> 7), (A_n >= 9), (A_n <> 11), (A_n < 15), (A_n > 18), (A_n <= 22)}.
Output: ValidRegSet = Null. The relation (A_n > 18) in the input spoilt any attempt to find a consistent *box* (see line 9 of the Algorithm-1D below).

The following two Lemmas and the Theorem 1 [Mitra et al, 2001] describe some interesting results in 1D case.

Lemma 1: Valid regions for a point (on a real line, under disjunctive constraints with respect to a sequence of points existing on the line) are contiguous, thus forming a convex interval (we call this interval the ***box***), i.e., if it exists. However, some of the existing *points* within the *box* may be excluded themselves as valid regions.

Inductive Proof:
Base. When S = {A_1}, only one point is in it, the lemma is trivially true. The *box* is [-infinity, A_1], [A_1, +infinity], [A_1, A_1], or [-infinity, +infinity].
Hypothesis: Suppose the lemma is true for n-1 points. Suppose, without the point A_{n-1}, i.e., for S = {A_1, A_2, .., A_{n-2}}, the box_{n-1} for A_n, a contiguous valid regions subject to some excluded point boundaries between them, is [A_k, A_j], for 1<=k<=j<=(n-2).
Step: The ignored point A_{n-1} could be at one of the five regions with respect to this interval (when A_k =/= A_j) : (1) left of A_k, (2) on A_k, (3) within (A_k, A_j), (4) on A_j, and (5) right of A_j. Now, for cases (1) or (2), if (A_n ">=" A_{n-1}) or (A_n ">" A_{n-1}), then the box_n for A_n for all n-1 "previous" points {A_1, A_2, .., A_{n-2}, A_{n-1}} remains the same as box_{n-1}. Alternatively, if the relation is (A_n "<=" A_{n-1}) or (A_n "<" A_{n-1}) for these two cases, then the box_n will vanish to Null - an inconsistent situation. Symmetrically opposite situations exist for cases (4) and (5). For the case (3), if the relation is (A_n "<=" A_{n-1}) or (A_n "<" A_{n-1}), then the box_n shrinks to [A_k, A_{n-1}], or if the relations is (A_n ">=" A_{n-1}) or (A_n ">" A_{n-1}) then the box_n shrinks to [A_{n-1}, A_j].

The last situation here is for (A_n "=" A_{n-1}) whence the box_n shrinks to [A_{n-1}, A_{n-1}] for case (3) and to Null (inconsistency) for other cases.

Note that the box_n remains unaffected for (A_n "<>" A_{n-1}) or (A_n "<=>" A_{n-1}). However, the set of valid regions gets split into more **contiguous regions**, subject to the exclusion of the point A_{n-1}, depending on if the relation is "<>" (A_{n-1} is excluded from the box_n) or "<=>".

A last scenario is when the box_{n-1} for A_n is a point [A_k, A_k] to start with, Then there are three regions for A_{n-1}: (1) left of A_k, (2) on A_k, or (3) right of A_k. It is trivial to see that A_n in that case either will remain coincided with A_k for (A_n "=" A_{n-1}) with case (2), for (A_n ">" or ">=" A_{n-1}) with case (1), for (A_n "<" or "<=" A_{n-1}) with case (3), or be inconsistent. *End of proof for Lemma 1.*

Lemma 2: There is no consistent region if the *box* does not exist (in other words the *box* is Null).

Proof: Form the proof of the Lemma 1 it is clear that all the valid regions must lie within the *box*. Hence, if the *box* is Null, then there is no valid region or consistent solution for the problem. *End proof.*

Theorem 1: A set of valid regions could be found for a new point A_n having point-to-point disjunctive relations with a sequence of points $\{A_1, A_2,\ldots, A_{n-1}\}$, as expressed in our language, if and only if a valid convex interval on the sequence (defined as the *box* before) exists for A_n.

Proof: Trivial. Lemma 1 proves the "if" part, and the Lemma 2 proves the "only-if" part. *End proof*

The Theorem 1 is used to preprocess the constraints in finding the convex interval - the *box*. Lemma 1 provides additional power in extracting the valid regions within this *box*. The valid regions are created by splitting the *box* with respect to those points having "<>" and <=>" relations with respect to the new point A_n

3.2 Algorithm

Input: (1) A non-empty sequence of points $S = \{a_1, a_2,\ldots, a_n\}$, with $n >= 1$. (2) A new point a_{new} with relations set $\{(a_i\ r_i\ a_{new}): 1 <= i <= n\}$

Output: A valid region set for a_{new}, ValidRegSet = $\{(a_{i1}, a_{i2}): 1<= k <= i1 <= i2 <= j <= n\}$, for i1 and i2 within some bound between k and j (following Lemma 1), OR, ValidRegSet = Null, indicating inconsistency.

An example ValidRegSet = $\{(a_3, a_4), (a_4, a_4), (a_4, a_5), (a_5, a_6), (a_6, a_6)\}$, a region from a_3 (k=3) through a_6 (j=6) excluding points a_3, and a_5. Mathematically a correct notation for the second region above should have been $[a_4, a_4]$, a closed interval, but we will ignore brackets in favor of parentheses in our syntax in the ValidRegSet, for the sake of uniformity.

We call the interval $[a_k, a_j]$ as the *box*.

Example 1:

Input: S = {3, 7, 9, 11, 15, 18, 22}, R = {(A_n >= 3), (A_n <=> 7), (A_n >= 9), (A_n <> 11), (A_n < 15), (A_n <> 18), (A_n <= 22)}.

Output: ValidRegSet = {(9, 9), (9, 11), (11, 15)}. The *box*, which is not an output, here is [9, 15].

Example 2:

Input: S = {3, 7, 9, 11, 15, 18, 22}, R = {(A_n >= 3), (A_n <=> 7), (A_n >= 9), (A_n <> 11), (A_n < 15), (A_n > 18), (A_n <= 22)}.

Output: ValidRegSet = Null. The relation (A_n > 18) in the input spoilt any attempt to find a consistent *box* (see line 9 of the Algorithm-1D below).

Algorithm-1D:

(1) validRegSet = Null;
(2) // FIND THE BOX FIRST: PRE-PROCESSING
(3) l = - infinity; r = + infinity;

```
// [l, r] is the box, initialized with the two extremes
(4) state = "findLeft";
// the state variable to keep track of the status

(5) for each a_i in the sequence S do
(6)   if (a_i "<=" a_new) or (a_i "<" a_new) then
(7)     if (state == "findLeft") then
(8)       l = a_i
        else
(9)       return validRegSet;   // Null, INCONSITENCY;
        end if;
      end if;
(10)  if (a_i "=" a_new) then
(11)    l = a_i; r = a_i;
        state = "foundEq";
      end if;
(13)  if (a_i ">=" a_new) or (a_i ">=" a_new) then
(14)    if (state == "foundEq") then
                { }       // ignore
        else
(15)            r = a_i;
(16)            state = "foundRight";
        end if;
      end if;
(17)  if (a_i "<>" a_new) or (a "<=>" a_new) then
        { };     // ignore
    end for;  // from step 5

(18) if (l == r) then       // case of equality, and consistent
(19)   validRegSet = {(l, l)};
(20)   return validRegSet;
     end if;

// FIND VALID REGIONS NOW
(21) if (a_new "<=" l) then
       validRegSet = {(l, l), (l, nextPoint (l) )};

(22)  for each a_p starting from nextPoint(a_l) through previousPoint(r) do
                // this loop may never execute when r is next point to l in S
(23)    if (a_new "<=>" a_p) then
          validRegSet = validRegSet ∪ {(a_p, a_p)};
        else { };    // ignore a_p as a region when a_new "=/=" a_p
        end if;
(24)    validRegSet=validRegSet∪{(a_p, nextPoint (a_p))};
      end for;

(25) if (a_new ">=" r) then
```

validRegSet = validRegSet ∪ {(r, r)};

(26) return validRegSet;
End Algorithm.

In the first part of the algorithm the for-loop runs over all the nodes, leading to an O(n) complexity, for n points in input S. The second for-loop in the last part runs over a subset of the points (only within the *box*), also having worst case complexity O(n). Hence, the asymptotic time-complexity of the algorithm is O(n).

The above algorithm inserts a new object/point in a sequence of objects/points. However, it has to also check for inconsistency while doing so. That is where it primarily differs from any traditional number-insertion algorithm, and that is the reason why it has to pass over all the points (in the first part) even when it has found the bound within which the solution is supposed to lie (finding the *box* in the first part). Also, it has disjunctive relations like "<>" or "<=>". Points with those relations are ignored initially (in the first part), but they are used to extract separate valid regions from within the *box* in the second part.

4 Reasoning in 2-D Space

4.1 The Language

The 2D-case is the case of Cardinal-directions algebra [Ligozat, 1998]. In two dimensions the disjunctive relations on a particular dimension could not be expressed (in general) independent of the relations in the other dimension. Thus, ((A_x <> B_x) and (A_y >= B_y)) expresses four regions ((A_x < B_x) and (A_y > B_y)), ((A_x < B_x) and (A_y = B_y)), ((A_x > B_x) and (A_y > B_y)), OR ((A_x > B_x) and (A_y = B_y)). Dropping any one of these four relations will make it impossible to collapse disjunctive relations over different dimensions without losing information. For example, we might only have ((A_x > B_x) and (A_y > B_y)) OR ((A_x < B_x) and (A_y = B_y)), that cannot be expressed as ((A_x <> B_x) OR (A_y <= B_y)) or any such compact expression.

A constraint between a node A_n and a set of other nodes A_1, A_2,…,A_{n-1} will be expressed as: {((A_{nx} R_{xij} A_{ix}) AND (A_{ny} R_{yij} A_{iy})): 1<=j<=9, 1<=i<n-1}, where a relation R is an element of the point-relations {<, >, =}, and j could run over up to nine (3x3) disjunctive possibilities on the two dimensions. (We have used a notation with flattened suffices/indices for the sake of convenience.)

A set of valid regions for A_n would be expressed as: {((L_{xj}, L_{yj}), (R_{xj}, R_{yj})) | 1<=j< m}, for m valid regions. L stands for the lower left corner of a rectangular region, and R stands for the upper right corner of that region. The corner points and the bounding line segments of the region are excluded from the latter (open region). Here, L_{xj}=R_{xj} and L_{yj}=R_{yj} indicates a point, whereas an equality on only one dimension indicates a line segment.

The concept of Box in 2D (or in d-D, d>=2) is very similar to that of box in 1D. It is the possible region in which the solutions may lie. Suppose all expressions like ((A_x > B_x) and (A_y > B_y)) OR ((A_x < B_x) and (A_y = B_y)) is “collapsed” to ((A_x <>

B_x) and (A_y >= B_y)) (thus, loosing information). If we run the 1D algorithm on such projected relations on each axis i and take a cross product of the corresponding *box_i*, then that region in 2D (or in d-D) would be called the "Box".

Lemma 3: There does not exist any valid region if the Box is empty [Mitra et al, 2001].

Proof: It can be trivially proved that the Box is empty if and only if its projections Box_x and Box_y are also empty. Any valid region will have valid projections on X and Y axes, such that all relations on X-axis (in R_x) and on Y-axis (in R_y) will support it. When either Box_x or Box_y is empty then there is no such universal support in R_x or in R_y, or in both of them. Thus, no valid region for the point a_N can exist. *End proof*.
This lemma is used in the Algorithm-2D to preprocess for checking inconsistency and extracting the Box. Lemma 3 is a 2D version of the Lemma 2.

Alternative version of Lemma 3: No valid region can exist outside the Box.
Proof: Can be easily proved by contradiction.

A crucial aspect of the Lemma 1 for 1D-case, namely, the contiguous-ness (subject to the possible exclusions of some existing points) property, is not valid in 2-D. That is, even if a Box exists, (1) the valid regions within it may not be contiguous (adjacent to each other with possibly separated by lines), and (2) no valid region may exist at all. This could be easily verified with the following examples.

Example 3: S = {a_1, a_2}, such that S_x ={a_2x, a_1x}, S_y = {a_1y, a_2y}.
R = {((a_3x ">" a_1x) and (a_3y ">" a_1y), OR (a_3x "<" a_1x) and (a_3y "<" a_1y)),
((a_3x ">" a_2x) and (a_3y ">" a_2y), OR (a_3x "<" a_2x) and (a_3y "<" a_2y)) }
The valid regions will be non-contiguous here (with the Box having projections [-infinity, +infinity] on both axes).

Example 4: S = {a_1, a_2, a_3}, such that S_x ={a_2x, a_3x, a_1x}, S_y = {a_1y, a_3y, a_2y}.
R = {((a_4x ">" a_1x) and (a_4y ">" a_1y), OR (a_4x "<" a_1x) and (a_4y "<" a_1y)),
((a_4x ">" a_2x) and (a_4y ">" a_2y), OR (a_4x "<" a_2x) and (a_4y "<" a_2y)),
((a_4x ">" a_3x) and (a_4y "<" a_3y), OR (a_4x "<" a_2x) and (a_4y ">" a_3y))}.
One could find valid regions (for a_4) independently for relationships with any pair of points from (a_1, a_2, and a_3), but their overlap (set intersection) is Null. This makes the CheckRegion algorithm (see below) necessary within the Algorithm-2D. Once again, the Box is fully open infinite region in the 2-dimensions in this example.

4.2 Algorithm

Input: (1) A non-empty list of points in the 2D space: S = {(a_{ix}, a_{iy}) | 1 <= i <= n}, with n >= 1. Note that a_{ix} and a_{iy} are strictly ordered in their respective dimensions, although the orderings may not be the same.
(2) A new point a_N: (a_{Nx}, a_{Ny}) and its relations with the list in S,

R = {((a_{ix} r_{i1x} a_{Nx}) && (a_{iy} r_{i1y} a_{Ny})) || … up to || ((a_{ix} r_{i9x} a_{Nx}) && (a_{iy} r_{i9y} a_{Ny})) | 1 <= i <= n}, with any r_{ikx} or r_{iky} is one of the {"<", ">", "="}, k may run up to 9 because those many combinations are possible for each of the three values over r_{ikx} and r_{iky}. ["&&" is the logical AND, and "||" is the logical OR] (note slight differences in notations with respect to those in the previous sub-section).
Output: A valid region set for a_N, ValidRegSet = {(v_{q1x}, v_{q1y}), (v_{q2x}, v_{q2y}) | 1 <= q <= m}, where m is the number of regions bound by $(2n-1)^2$, the number of total regions created in two dimension by the points in S. A Null set for the validRegSet would indicate inconsistency. Examples: see the examples 3 and 4.

Algorithm-2D:

```
// BOX EXTRACTION:  PREPROCESSING
// sort the x-projections
(1) Say, Sx = {Ax1, Ax2, … Axn} = Sort(aix , 1<= i <= n);
(2) Say, Rx = {(Axi  rix  aNx) | 1 <= i <= n}, where rix = (ri1x ∪… up to ∪ ri9x);
                    // Union x-relations with respect to each point (collapsing)
// run Algorithm-1D to find the x-component of the box
(3) Boxx = Algorithm-1D (Sx, Rx, aNx);
(4)  if  (Boxx  ==  Null)  then
(5)    ValidRegSet  =  Null;
(6)    return  ValidRegSet;              // INCONSISTENCY DETECTED
     end if;
(7*) /// Repeat steps (1) through (6) for finding Sy, and
          Ry, and for finding and checking Boxy
      // Now the solution regions must lie within the
           Box = (Sx X Sy), if both Sx and Sy exist

(8) ValidRegSet = Null;
(9) for each  vpx  in Sx starting from  lx  through  previousPoint (rx, Sx) do
          // linear regions on the  x = vpx line
(10)  if  (aNx  "<="  vpx) ∈ Rx  then
          // initialize corners, see "output" above
(11)     vq1x = vpx;  vq2x = vpx;
(12)  else  if (aNx  "<"  vpx) ∈ Rx  then
(13)     vq1x = vpx;  vq2x = nextPoint(vpx, Sx);  // ignore the line on vpx

(14)          for  each vpy  in Sy  starting from ly  through  previousPoint(ry, Sy)  do
(15)              if  (aNy  "<="  vpy) ∈ Ry  then
(16)                  vq1y = vpy;  vq2y = vpy;
(17)              else  if (aNx  "<"  vpx) ∈ Rx  then
(18)                  vq1y = vpy;  vq2y = nextPoint(vpy, Sy);
                  end if;
(19)              if  CheckRegion ((vq1x, vq1y),  (vq2x, vq2y))  then
```

(20) validRegSet = validRegSet ∪ {((v_{q1x}, v_{q1y}), (v_{q2x}, v_{q2y}))};
end for; // from line 14

(21) if (a_{Ny} ">=" ry) ∈ R_y then //boundary point
(22) v_{q1y} = ry; v_{q2y} = ry;
end if;
(23) if CheckRegion ((v_{q1x}, v_{q1y}), (v_{q2x}, v_{q2y})) then
(24) ValidRegSet = validRegSet ∪ {((v_{q1x}, v_{q1y}), (v_{q2x}, v_{q2y}))};
end for; // from line 9

(25)if (a_{Nx} ">=" rx) ∈ R_x then // right boundary point
(26) v_{q1x} = rx; v_{q2x} = rx;

(27*) /// repeat the for-loop from steps (14) through (26);

(28) return validRegSet;
End Algorithm.

Algorithm CheckRegion ((x1, y1), (x2, y2))
(1) for each A_x ∈ S_x (where A_x ≡ a_{ix} in S) do
(2) if (a_{ix} is before x1 in S_x) then
(3) say, ((a_{ix} "<" a_{Nx}) && (a_{iy} r a_{Ny})) ∈ R, and say, (a_{iy} r2 y2) in S_y;
// a_{Ny} and y2 are not having the same rel with a_{iy}
(4) if (r =/= r2) then
(5) return False; // INCONSISTENCY
end if;
(6) if (a_{ix} is after x2 in S_x) then
(7) say, ((a_{ix} ">" a_{Nx}) && (a_{iy} r a_{Ny})) ∈ R,
and say, (a_{iy} r2 y2) in S_y;
(8) if (r =/= r2) then
(9) return False; // INCONSISTENCY
end if;
end for; // from line 1

(10) return True;
End Algorithm.

The Algorithm-2D first collapses x-relations and y-relations. For example, ((A_x > B_x) and (A_y > B_y)) OR ((A_x < B_x) and (A_y = B_y)) becomes ((A_x <> B_x) OR (A_y <= B_y)). Next it extracts Box_x and Box_y from the corresponding total orders on the axes and their collapsed relations with respect to a_{Nx} and a_{Ny}. If the Box is not empty then it picks up the regions within it, one by one, in order to check each of the regions validity. These regions within the Box are created by the points with "<>" and "<=>" relations on each axis with respect to a_{Nx} and a_{Ny}. Checking a region's validity is done by noting where does it lie with respect to each point, and if a_N could lie in that space with respect to that point. All such valid regions are collected in the ValidRegSet.

CheckRegion has O(n) complexity with the for-loop at line (1), and because lines (3) and (7) can be performed in constant time. Algorithm-2D has O(n logn) (actually O(n)+O(n logn)) complexity for lines (1) through (6) and for lines in (7*). Its main loops run for extracting regions from within the box with complexity $O(n^2)$, within which the CheckRegion runs. Hence the total asymptotic complexity of the Algorithm-2D is $O(n^3)$.

5 Reasoning in d-D Space

The language, the lemmas, and any relevant algorithm for two dimensions can be trivially extended toward any d-dimensional space with d>2. Most of the results will have a corresponding extension. However, the complexity of the corresponding algorithm for constraint processing will depend on the dimension (for d>1) of the space.

Although the concept of contiguous-ness of regions is not very useful in higher dimensions (d>2), a related concept - that of "pre-convexity," developed by Ligozat [Ligozat, 1996] originally for the interval-based temporal reasoning domain, is useful here. A *convex* region in any Euclidean space is where the shortest path between any two points in the region is totally contained within that region. The idea of convexity in 2D space has been studied in detail by Davis et al [Davis, 1999]. In our d-dimensional Cartesian space a hyper-ellipsoid or a hyper-rectangle would be such a convex region. Following Ligozat [Ligozat, 1998] we define a *pre-convex* region as a region, which is convex except possibly for some lower dimensional regions within it. Lines and points are lower dimensional regions in the 2D space. This is a very similar concept as defined in [Ligozat, 1996; Ligozat, 1998] for the interval-based temporal reasoning domain and later for the 2D Cardinal-directions algebra.

Note that a pre-convex region (or a convex region) in a d-D space could be lower dimensional as well. For example, a rectangular plane is a convex region not only in a 2D space but also in a 3D space or in a higher dimensional space. Correspondingly, a rectangular plane minus a straight line cutting it could be a pre-convex region in a 3D or higher dimensional space. We will make a distinction of such regions with strictly d-dimensional pre-convex (or convex) regions in a d-dimensional space. We will call such a d-dimensional pre-convex (or convex) region in a d-D space as a *strongly pre-convex* (or s*trongly convex*) region. The necessity for this distinction will be explained later (in the Conclusion).

Lemma 4: If the initial language is restricted (subset of the power set of the basic relations) in such a way that every element in it corresponds to a strongly pre-convex region, then the "Box" will always be a strongly pre-convex region.

Proof sketch (by induction) for the strongly "convex" region case:
Induction Base for the case of the second point to be inserted in a space with only one point in it: assumed to be true by the "if" part of the Lemma.
Induction hypothesis: Assume to be true when there are (n-1) points in the space, and the n-th point is to be inserted.

Induction Step: The final valid region for the n-th point is the set intersection of all the individual valid regions V_i for the n-th point with respect to each of the other points "i" (1<=i<=n-1), existing already in the space. Since every V_i is strongly convex their intersection has to be strongly convex also (by Helly's theorem [Chvatal, 1983]).

Extending the Lemma toward the strongly pre-convex region is based on the observation that excluded lines and points (hyper-regions of smaller dimensions) from each V_i may exclude some lines and points from the final strongly convex valid region, but does not affect the validity of the lemma as outlined above.
End proof of Lemma 4.

Note that the Lemma 4 is not true if we replace the notion of the strongly pre-convex region with that of the "contiguous" region. One can introduce a language of contiguous regions but the Lemma 4 would not have any correspondence for such a language, i.e., the set of valid regions will not necessarily be contiguous for n>2. Try with two points A and B in the space such that (A Northeast of B), and then introduce C with (C {Southeast, East, Northeast, North, Northwest} A), and (C {North, Northwest, West, Southwest, South, Southeast} B), both relations being "contiguous". The resulting valid regions for C will be two non-contiguous regions (the most Northwest corner and the most Southeast corner regions) in the space.

Another point to note here is that in the case of 1D the contiguous regions and the strongly pre-convex regions are identical.

Lemma 5: For higher dimensions (d>=2), a strongly pre-convex region is exactly same as the Cartesian product of the regions' projections on the axes of the space.

Proof: Firstly, note that a strongly pre-convex region is a region of highest dimension and so, has to have a linear (interval) projection on each axis.
Secondly, the same lemma is trivially true for a strongly convex region. By definition [Chvatal, 1983] a convex region is such that all points on the shortest line between any two points within the region lie within that region also [Chvatal, 1983]. Hence, the projections of those intermediate points on any axis will also lie between the projections of the two end points on the same axis. Thus, the lemma is true for convex regions.
Next, consider strongly pre-convex regions. Such a region may have some lower dimensional convex regions being absent from within it (e.g. a cross-section rectangular plane absent from a cube). Each such absent region will have a projection as a point only on a corresponding axis (e.g., a plane represented by Z=5, will have a projection on Z-axis at point 5). Thus, the projections of a strongly pre-convex region (P) will be linear intervals (P_x, P_y, …), but may have some points absent from them (from P_x, P_y, …). Taking a cross product of these projections will faithfully reproduce the original (strongly) pre-convex region eliminating the corresponding lower dimensional regions corresponding to the absent points from those linear-interval projections on the axes. *End proof*.

The Lemma 5 may not be true for a lower dimensional (<d) pre-convex region in a d-D space, i.e., for not a strongly pre-convex region.

6 Incremental Algorithm in d-D Space

The simplest case of 1D has a simple incremental algorithm. Regions specified by the disjunctive relations on a line (strongly pre-convex by definition) are intersected to obtain a *Null* or a strongly pre-convex *box* as the output. An algorithm for the 2D case collapses the non-disjunctive relationships on the two axes (X and Y) to disjunctive relations (explained below), then runs the 1-D algorithm on each axes, to extract corresponding box_x and box_y on them, and then goes over each region in the resulting Box (which is a cross product of box_x and box_y), in order to check for the validity of each region with respect to the input constraints. Both of these algorithms are presented here before. An algorithm corresponding to d-D will extend the 2D version of the algorithm.

The idea of collapsing mentioned explained before is utilized to do some pre-processing: as the Box-extraction task, which reduces the number of regions to go over, for the purpose of finding valid regions (according to the Lemmas 2 and 3).

Algorithm d-D incremental:

(1) Collapse non-disjunctive constraints from the new point A_n to the set of existing points in the space $\{A_1, A_2, \ldots A_{n-1}\}$ over each of the d axes of that d-D Cartesian space.
(2) Run 1-D algorithm (section 2) on each of the axes for extracting {box_1, box_2, … box_d}.
(3) Create the Box = {Cartesian product of all box_i's, 1<=i<=d}.
(4) Check for the validity of each of the regions within the Box only. Return the set of valid regions (which could be a Null set, indicating *inconsistency* of the input).

End algorithm.

As discussed before, the total number of regions is always polynomial with respect to the number of points existing in the space, for a given number of dimensions of the space. The algorithm reduces the number further in steps 1 through 3 (pre-processing), all of which runs in polynomial time. Hence, the complexity of the algorithm is polynomial. In fact, the complexity of the algorithm is $O(n^{d+1})$ [Mitra et al, 2001].

In order to solve the full BCSP (given a set of point-agents and binary relations between them check the consistency of the constraints) the above algorithm has to be run incrementally again and again in order to introduce all the points in the system. Backtracking may be necessary when an inconsistency is detected while introducing any k-th point (1<k<=n) where (k-1) points are already placed. Thus, polynomial nature of the above algorithm does not guarantee that the full BCSP is a P-class problem.

7 The Full BCSP

Lemma 6: For a strongly pre-convex language (where the allowable input disjunctive elements represent strongly pre-convex regions only) the Box is identical to the union of the set of valid regions, minus possibly some lower dimensional regions.
Proof sketch: The Box is nothing but the Cartesian product of the allowable segment on each axes separately (by definition). According to the Lemma 5, that is the set of valid regions also. Hence, the lemma is true.
End proof.

Algorithm d-D BCS:
(1) Collapse all non-disjunctive relations as disjunctive ones on each axis, as was done in the Algorithm d-D incremental.
(2) Do topological sort of the projections of points on each axis according to the disjunctive binary constraints derived in the step (1).
(3) If a topological sort on any axes fails - detecting a cycle, then return Failure (Inconsistency), else return Success, with the Cartesian product of topo-sorted points on the axes as a consistent solution.
End algorithm.

Example 4 (Figure 1): [d=2 here] Constraints for a6 is such that a6x could be inserted between (a1x, a2x), on a2x, between (a2x, a3x), or between (a3x, a4x); and a6y could be inserted between (a2y, a4y), between (a4y, a1y), on a1y, or between (a1y, a3y). The set of valid regions here is the Cartesian product of these regions on the respective axes (the shaded area on the figure).

Theorem 2: Reasoning with a strongly pre-convex language is tractable.
Proof: The *Algorithm d-D BCS* for multi-dimensional point-based BCSP solves the problem in a polynomial time.

The algorithm is clearly polynomial, running topological sort ($O(n^2)$) d times, each time for n point-agents: $O(dn^2)$. Proving the algorithm's correctness is somewhat involved.

Lemma 7: The consistent solutions are Cartesian products of all the topological-sortings of the projections of the points on the axes.
Proof sketch:
Induction base: Trivially true for one point.
Induction Hypothesis: Suppose true for k points, and the consistent solutions are $\{T_1, T_2, \ldots, T_p\}$. Each Ti ($1<=i<=p$) is the Cartesian product of total orders of k points on all axes.
Induction step: The (k+1)-th point is to be inserted in T_i ($1<=i<=p$) following binary constraints with all other k points. The set of valid regions is a rectangular strongly pre-convex region that is the Box (Lemma 6). It is easy to verify that each valid region can be obtained by inserting the projection of the point k on each axis in T_i and then taking a Cartesian product of the orderings on the axes. Not only that, since the set of valid regions together is a strongly pre-convex region no such Cartesian product

of any set of valid sortings on axes will be left out of the set of valid regions. Hence the set of valid regions is exactly all the Cartesian products of all the valid orderings for inserting the projections of k on the axes (see the Example 4).
End of proof sketch of lemma 7.

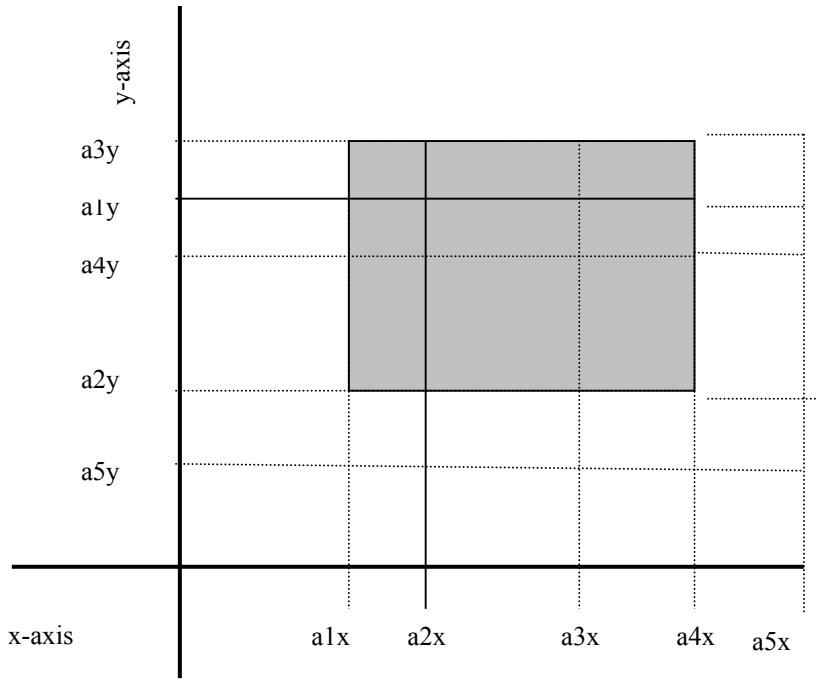

Figure 1: A solution space (shaded) for the point-agent a6 in 2-D

According to the Lemma 7, if on any axis a topological sort of the point-projections does not exist, then there would be no solution, as is correctly returned by the above algorithm. On the other hand, if topo-sorts are found on each of the axes, then their cross product must be a solution. So, anything returned by the algorithm as a Success, is a correct solution. This proves the correctness of the Algorithm *d-D BCSP*, and hence, that of the Theorem 2.
End proof of Theorem 2.

Pre-convexity is also the basis of finding a maximal tractable sub-algebra in the case of reasoning (full BCSP) with time-intervals [Ligozat, 1996]. Similar results have been observed for continuous domain CSP also [Faltings, 2000]. Our conjecture is that, this is the case for any domain where the space is a Cartesian product of some total orderings (as a real line is), and where the objects (locations of the agents) are expressed as points in the space. Note that in the 1D case the full language is strongly pre-convex (as mentioned before), and the full BCSP is tractable in the 1D case. Also, if "=/=" (inequality) is excluded from the full language in the 1D case it becomes convex, which leads to an even more efficient reasoning [van Beek, 1990].

Theorem 3: The full-BCSP is NP-hard.
Proof: Trivial. Ligozat [Ligozat, 1998] has proved that the reasoning over a full network with constraints from Cardinal-directions algebra in 2D is NP-hard. A generalization over d-D has to be NP-hard then, since 2D is a special case of d-D with d =2.
End Proof.

8 Some Suggested Applications

8.1 A Qualitative Database on Personalities

This is a database recording some attributes of the individuals. The attributes are not measured quantitatively; rather they are recorded for their relative values with respect to other individuals in the database. The attributes (dimensions) recorded for this example are height, weight and smartness. Individuals already existing in the database are Mary, John and Sheila. A new individual Paul joins the group. Relative orientations of the existing three are as follows: (Sheila is taller than John, and Mary is taller than Sheila), (Sheila is heavier than Mary, and John is heavier than Sheila), and (John is smarter than Mary, and Sheila is smarter than John). The new relationships for Paul's attributes are known (incomplete information) to be as follows: Paul is smarter than all three. However, with respect to Mary and John, either he is both taller and heavier than each, or he is both shorter and lighter. With respect to Sheila, Paul is either (taller but lighter) or (shorter but heavier). Our consistency-checking algorithm will detect inconsistency in the information about Paul.

8.2. Military Reconnaissance

It is learnt that the enemy is planning for three diversionary and a major attack. They are communicating about three different aspects of the attack to their cells using different channels of communication. These three aspects are time of attack, latitude and longitude of the place attack. Intelligence sources picked up some information of the relative orderings on these aspects. Timings of the three diversionary attacks are in the order of first attack, then second attack, and then the third attack. The latitude of these attacks' locations are in the same order as well. However, the longitude of the third attack is in between the other two. Now the information about the major attack has come in. It is within the first and second attacks for all the aspects. Its latitude is not same as that of the third attack but the longitude is. It will also not take place at the same time as that of the third.

The constraint propagation technique proposed in this work will be able to identify that the major attack will be bounded within the box cornered by the first and the second attack in this 3D space, and that its latitude and timing must be less than that of the third diversionary attack. This type of capability may be useful in

Command, Control, Communications, Computers, and Intelligence (C4I) architectures (http://www.fas.org/irp/agency/army/odcsint/org.htm#im).

9 Conclusion

In this article we have studied a framework for the multi-dimensional point-based spatial reasoning with incompleteness. This type of spatial reasoning is important when multiple agents would try to find their relative position in a real space (R^d) under a noisy environment. We have provided a polynomial incremental algorithm for the purpose of finding possible locations for a new agent in a space where some agents have already committed about their positions with respect to each other. Some intricately detailed algorithms for the 1D and 2D special-cases are also described here.

A next higher level of the problem is when no position for the new agent is found. In that case the existing agents may have to backtrack and reorganize themselves in order to accommodate the new agent, under the same input disjunctive constraints that existed between themselves before. This demands consideration of all solution spaces for the new agent and not only the ones according to a committed space by the older agents. This problem is the framework for traditional binary constraint satisfaction problem (full BCSP), where the question is to find out if there exists any solution for a given set of n point-agents and binary relations between them. The full BCSP is an intractable problem, unless one restricts the type of input. We have provided here such a sub-language where the full BCSP in multi-dimensional point-based disjunctive reasoning is tractable. An existing concept of pre-convexity in the literature is utilized for the purpose. We conjecture that the strong pre-convexity is the underlying basis for finding tractable sub-languages in many spatio-temporal reasoning domains. This is the future direction of our research.

Recently [Balbiani and Condotta, 2002] have come up with a counter-example that the pre-convex language is not closed under the set-intersection operation for d>2, and thus, it does not form a corresponding sub-algebra. However, their counter-example uses elements that are not strongly pre-convex as defined here. Our results presented here remain valid for the strongly pre-convex class.

An interesting case of spatial reasoning is when the basic regions are angular (e.g., between every 60-degree lines from the origin - star-shaped zoning), rather than the quadrants as in the case presented here [Mitra, 2002]. Military reconnaissance often uses reasoning with such qualitative angular zones. A convex region in such a case is a region that is enclosed within lines forming 180-degrees or less at the center (true even in our case presented here). A strongly pre-convex region is a convex region minus the regions of lower dimensions within the latter. An interesting challenge would be to prove that the BCSP is tractable for a strongly pre-convex language in this domain, as per our conjecture stated above. The technique used here to prove tractability (Theorem 2) does not work in a star-zoning case. One of the authors is also working on this problem.

Acknowledgment

This work is partly supported by the US National Science Foundation (IIS-9733018, IIS-0296042). Lail Hossein has implemented the Algorithm-1D and made interesting observations during the work. One of the authors, Mitra is also indebted to the LIMSI, CNRS laboratory in Paris for hosting him over some Summer-months, where the works presented here have started.

References

[Beek, 1990] Beek, P. van: "Exact and approximate reasoning about qualitative temporal relations," *Ph.D. dissertation.* University of Alberta, Edmonton, Canada (1990).

[Chvatal, 1983] Chvatal, V.: *Linear Programming*, W.H. Freeman (1983).

[Davis, 1999] Davis, E., Gotts, N.M., and Cohn, A.G.: "Constraint Networks of Topological Relations and Convexity," *Constraints*, 4 (3), pp 241-280 (1999).

[Faltings, 2000] Faltings, B.: "Using topology for spatial reasoning." Proceedings of the *AI and Mathematics* Conference (2000).

[Balbiani and Condotta, 2002] Balbiani, P., and Condotta, J. F.: "Spatial reasoning about points in a multidimensional setting," *Journal of Applied Intelligence*, Kluwer Publishers, to appear (personal communication) (2002).

[Hazarika, 2001] Cohn, A.G., and Hazarika, S. M.: "Qualitative Spatial Representation and Reasoning: An Overview," *Fundamenta Informaticae*, 46 (1-2), pp 1-29, (2001).

[Ligozat, 1996] Ligozat, G.: "A new proof of tractability for ORD-Horn relations." Proceedings of the Thirteenth *National Conference on Artificial Intelligence (AAAI),* Portland, Oregon, USA, pp. 395-401 (1996).

[Ligozat, 1998] Ligozat, G.: "Reasoning about Cardinal Directions," *Journal of Visual Languages and Computing.* Vol. 9, 23-44 (1998).

[Mitra et al, 2001] Mitra, D., Ligozat, G, and Hossein, L.: "Modeling of multi-dimensional relational constraints between point objects." Proceedings of the Florida AI Research Symposium (FLAIRS), Key West, Florida (2001).

[Mitra, 2002] Mitra, D.: "A class of star-algebras for point-based qualitative reasoning in two-dimensional space." Proceedings of the Florida AI Research Symposium (FLAIRS), Pensacola Beach, Florida (2002).

[Vilain and Kautz, 1986] Vilain, M., and Kautz, H.: "Constraint propagation algorithms for temporal reasoning." Proceedings of the Fifth *National Conference on Artificial Intelligence (AAAI),* Philadelphia, PA, pp. 377-382 (1986).