Modeling of multi-dimensional relational constraints between point objects

Debasis Mitra (1), Gerard Ligozat (2), and Lail Hossain (1)

(1) Department of Computer Science
Jackson State University, P.O. Box 18839
Jackson, MS 39217, USA
E-mail. dmitra@ccaix.jsums.edu, **lshossain@yahoo.com**

(2) LIMSI, Universite Paris-Sud
Bldg. 508, P.O. Box 133
F-91403 ORSAY Cedex, France
E-mail. ligozat@limsi.fr

Abstract

Most of the approaches in the spatio-temporal reasoning area use a relational algebraic framework where the domains of variables are typically not given an explicit considerations (for the purpose of constraint processing) in favor of the relational constraints between those variables. However, some recent works have shown the deficiency of those approaches in finding a globally consistent solution. For example, in cyclic time-interval problems even path-consistent singleton models are not globally consistent. In this article, we have developed a domain-theoretic approach, which is routinely deployed in the traditional discrete-domain constraint satisfaction problems (CSP), for point-based relations in a multi-dimensional Cartesian-space. Our algorithms are also developed for the insertion problem (insert an object in a set of existing ones), which is at the core of any incremental approach for checking global consistency (Mitra, 2001). The results might be useful in a real-life modeling activities in any relevant area (e.g., visualization or data-modeling).

Key Words: Constraint Satisfaction Problem (CSP), Spatial Reasoning, Cardinal Algebra, Continuous-domain CSP.

1. Introduction

A spatio-temporal constraint satisfaction problem (STCSP) is a binary CSP where the domain of every variable is mapped onto the same space or time-related continuous - and often dense and infinite - domain. The constraints here are disjunctive subsets of a predefined exhaustive set of relational constraints that are possible between any two points in the respective domain (e.g., for 1-dimensional points the relations are {<, >, =}). A typical discrete-domain binary CSP is solved by instantiating the variables for their respective values, which typically come from some discrete and finite domains. However, STCSP problems are typically handled within the relational algebraic framework (see Chittaro and Montanari, 2000). The fundamental operation in this framework is a pre-defined composition operation between any pair of relations (e.g., compose ("<", "<") = "<", for three points).

While finding some approximate consistent (say, 3-consistent, between every triplet of variables) solution is easy (often polynomial) in a relational algebraic framework, extracting a global solution is quite tricky there. So far in the literature it was presumed that if we can extract a 3-consistent singleton model (single relation between each pair of variables), then that indicates the presence of a globally consistent solution. From such a singleton model one could then extract an instantiation for each variable satisfying the constraints. However, recent works in the area of cyclic time (Balbiani and Osmani, 2000) shows that this situation is not universally true. In cyclic-time (where the time-line wraps onto itself) time-intervals may have a 3-consistent singleton model that is not globally consistent. This is the primary motivation behind developing domain-theoretic algorithms for STCSP as is done in the traditional discrete-domain CSP. This article makes a first attempt in such a direction - for the domain of points in a real space of one and two dimensions (extendible to any arbitrary number of dimensions, which is touched upon in the article). We have not addressed the domain of cyclic time here. Our work is also likely to have an impact on the modeling areas, e.g., in graphics and visualization, GIS, and data modeling in multi-dimensional databases, by providing a disjunctive information handling capability.

We start with the 1D case for establishing the background. The next section deals with the 2D case, followed by some discussion on the higher-dimensional cases. Some relevant discussions are provided next. The article concludes with a separate section.

2. 1D-case

2.1 Language

This is a case of point algebra (Vilain and Kautz, 1986), we are discussing it here primarily for the sake of creating a background for the higher dimensional cases. Some new results are also reported here. The language is designed to express constraints between a node A_n with

From: FLAIRS-01 Proceedings. Copyright © 2001, AAAI (www.aaai.org). All rights reserved.

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 answering any query about A_n). The last two relations cause A_n to ignore Ai 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 region is a point only. Some examples (1 and 2) are provided later.

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 without the point A_{n-1}, i.e., for S = {A_1, A_2, ... A_{n-2}}, and the *box* (convex interval, with contiguous valid regions subject to some excluded point boundaries between them) for An is [A_k, A_j], for 1<=k<=j<=(n-2). *Steps.* Then the next 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. For cases (1) or (2), if (A_n ">=" A_{n-1}) or (A_n ">" A_{n-1}), then the *box* for A_n remains the same. Alternatively, if the relation is (An "<=" A_{n-1}) or (A_n "<" A_{n-1}) for these two cases, then the *box* will vanish - an inconsistent relation. Symmetrically opposite situations exist for cases (4) and (5). For the case (3), if the relation is (An "<=" A_{n-1}) or (A_n "<" A_{n-1}), then the *box* 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* shrinks to [A_{n-1}, A_j].

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

Note that the *box* 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*) or "<=>".

A last scenario is when the *box* 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 coincide 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 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,..., 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.

2.2 Algorithm

Input: (1) A non-empty sequence of points S = {a_1, a_2,..., 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 bracket in favor of parenthesis 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;
(10)    if ($a_i$ "=" $a_{new}$) then
(11)        l = $a_i$;        r = $a_i$;
(12)        state = "foundEq";
(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";
(17)    if ($a_i$ "<>" $a_{new}$) or (a "<=>" $a_{new}$) then
            { };   // ignore
    end for;
(18)    if (l == r) then
        // case of equality, and consistent
(19)        validRegSet = {(l, l)};
(20)        return validRegSet;
        // FIND VALID REGIONS NOW
(21)    if ($a_{new}$ "<=" l) then
            validRegSet = {(l, l), (l, nextPoint ($a_k$) )};
(22)    for each $a_p$ starting from nextPoint($a_k$)
        through previousPoint($a_j$) do
// this loop may never execute when r is next point to l in S
(23)    if ($a_{new}$ "<=>" $a_p$) then

            validRegSet = validRegSet $\cup$ {($a_p$, $a_p$)};
        else { };
        // ignore $a_p$ as a region when $a_{new}$ "=/=" $a_p$

(24)        validRegSet=validRegSet$\cup${($a_p$, nextPoint ($a_p$))};
        end for;
(25)    if ($a_{new}$ ">=" r) then

            validRegSet = validRegSet $\cup$ {(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.

3. 2D-case

3.1 Language

This is the case of Cardinal algebra (Ligozat, 1998). In two dimensions disjunctive relations on a particular dimension could not be expressed (in general) independent of the relation 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. 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 $A_1, A_2, \ldots, A_{n-1}$ 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 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.

3.2 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 validRegSet would indicate inconsistency. Examples: see at the end of this section (examples 3 and 4).

```
Algorithm 2D:
        // BOX EXTRACTION: PREPROCESSING
// sort the x-projections
(1)Say, $S_x = \{A_{x1}, A_{x2}, \ldots A_{xn}\}$ = Sort($a_{ix}$ , 1<= i <= n);
(2)Say, $R_x = \{(A_{xi}\ r_{ix}\ a_{Nx})\ |\ 1 <= i <= n\}$, where $r_{ix} = (r_{i1x} \cup \ldots$ up to $\cup\ r_{i9x})$;
// Union x-relations with respect to each point (collapsing)
// run 1D algorithm to find the x-component of the box
(3)$Box_x$ = Algorithm_1D ($S_x$, $R_x$, $a_{Nx}$);
(4)  if ($Box_x$ == Null) then
(5)      ValidRegSet = Null;
(6)      return ValidRegSet;        // INCONSISTENCY
(7*)     /// Repeat steps (1) through (6) for finding $S_y$, and
         $R_y$, and for finding and checking $Box_y$
         // Now the solution regions must lie within the
         Box $\subseteq$ ($S_x$ X $S_y$), if both $S_x$ and $S_y$ exist
(8)      ValidRegSet = Null;
(9)      for each $v_{px}$ in $S_x$ starting from lx through
          previousPoint ($r_x$, $S_x$) do
// linear regions on the x = $v_{px}$ line
(10)        if ($a_{Nx}$ "<=" $v_{px}$) $\in R_x$ then
// initialize corners, see "output" above
(11)            $v_{q1x} = v_{px}$; $v_{q2x} = v_{px}$;
(12)        else if ($a_{Nx}$ "<" $v_{px}$) $\in R_x$ then
(13)            $v_{q1x} = v_{px}$; $v_{q2x}$ = nextPoint($v_{px}$, $S_x$);
                          // ignore the line on $v_{px}$
(14)        for each $v_{py}$ in $S_y$ starting from ly through
             previousPoint($r_y$, $S_y$) do
(15)           if ($a_{Ny}$ "<=" $v_{py}$) $\in R_y$ then
(16)              $v_{q1y} = v_{py}$; $v_{q2y} = v_{py}$;
(17)           else if ($a_{Nx}$ "<" $v_{px}$) $\in R_x$ then
(18)              $v_{q1y} = v_{py}$; $v_{q2y}$ = nextPoint($v_{py}$, $S_y$);
(19)           if CheckRegion (($v_{q1x}$, $v_{q1y}$), ($v_{q2x}$,
               $v_{q2y}$)) then
(20)              validRegSet = validRegSet $\cup$ {(($v_{q1x}$,
                  $v_{q1y}$), ($v_{q2x}$, $v_{q2y}$))};
(21)     if ($a_{Ny}$ ">=" ry) $\in R_y$ then //boundary point
(22)          $v_{q1y}$ = ry; $v_{q2y}$ = ry;
(23)       if CheckRegion (($v_{q1x}$, $v_{q1y}$), ($v_{q2x}$, $v_{q2y}$)) then
(24)     ValidRegSet = validRegSet $\cup$ {(($v_{q1x}$, $v_{q1y}$), ($v_{q2x}$,
$v_{q2y}$))};
         end for;
   end for;
(25)if ($a_{Nx}$ ">=" rx) $\in R_x$ then    // right boundary point
(26)      $v_{q1x}$ = rx; $v_{q2x}$ = rx;
(27*)     /// repeat the for-loop from steps (14) through
(28) return validRegSet;
End Algorithm.
```

```
Algorithm CheckRegion ((x1, y1), (x2, y2))
(1)for each $A_x \in S_x$ (where $A_x \equiv a_{ix}$ in S) do
(2)    if ($a_{ix}$ is before x1 in $S_x$) then
(3)         say, (($a_{ix}$ "<" $a_{Nx}$) && ($a_{iy}$ r $a_{Ny}$)) $\in$ 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
(6)     if ($a_{ix}$ is after x2 in $S_x$) then
(7)          say, (($a_{ix}$ ">" $a_{Nx}$) && ($a_{iy}$ r $a_{Ny}$)) $\in$ R,
             and say, ($a_{iy}$ r2 y2) in $S_y$;
(8)          if (r =/= r2) then
(9)                return False; // INCONSISTENCY
(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 each axes and their collapsed relations with respect to a_{Nx} and a_{Ny}. If the Box is not empty then it picks up regions within it one by one to check the latter's 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 $O(n^3)$.

Lemma 3: There does not exist any valid region if the Box is empty.

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 Algorithm 2D to preprocess for checking inconsistency and extracting the Box. Lemma 3 is a 2D version of lemma 2.

Alternative version of Lemma 3: No valid region can exist outside the box.

Proof: Can be easily proved by contradiction. *End proof.*

Lemma 3 is used to reduce the processing time for further constraint propagation by working only within the Box, in the second part of Algorithm 2D (as well as in Algorithm 1D). Lemma 3 is an extension of a part of the lemma 1 for 1D-case. However, a crucial aspect of the lemma 1, namely, the contiguous-ness (subject to the possible exclusions of some existing points) property, is not valid in 2D. That is, even if a Box exists, (1) the valid regions within it may not be contiguous, 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 output will have non-contiguous valid regions (with box being = [-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 necessary within the Algorithm 2D. Once again, the Box is fully open infinite region in the 2-dimensions in this example.

4. n-D case for n > 2

The language and algorithm for two dimensions can be trivially extended toward any d-dimensional space with d>2. All the results will have a corresponding extension. However, the complexity of the algorithm will differ.

The box extraction in preprocessing part is done in each axis independently. Hence, the complexity of that part will remain same O(n) for n points in S, although the loops from lines (1) through (6), as in Algorithm 2D, will run separately (as in lines (7*)) for d times. The sub-algorithm CheckRegion will also have the same complexity O(n). However, the loops (as in lines (9) and (14)) in creating the higher dimensional regions will have as many nesting as the number of dimensions. So, the total complexity will be $O(n^{d+1})$.

5. Discussion

What we have discussed here is the problem of inserting a new point with disjunctive relations with respect to a set of points in d-dimensional space (d = 1, 2 or >2), where all those old points have non-disjunctive complete relationships with respect to each other (as expressed by the strict orderings of their projections on the respective axes). Once a valid region is chosen for the new point in the space out of the validRegSet, that would extend the old set to a new one including the new point, where their mutual relations once again are complete non-disjunctive ones. This is true because each valid region can be defined by a set of complete relationships with respect all the existing points (and the inverse of a complete relation is another complete relation).

A complete binary CSP in this domain is where a set of points and some binary constraints between them are provided as the input. In such a problem the requirement is to check consistency and find a solution. A backtracking algorithm could easily be developed that would utilize the appropriate incremental algorithm discussed here for inserting points one by one in the partially developing solution (until all points are inserted there). Backtracking may be needed when one fails to insert a point at any stage. Backtracking may undo some of these relationships by choosing a different valid region for any such old-point with respect to the points entered before it. A failure of backtracking would indicate a global inconsistency. Although the insertion problem has polynomial algorithm as indicated here, the global consistency-checking (CSP) problem may not be so. However, we know that the 1D-case (point algebra) is polynomial whereas the higher-dimensional cases are intractable ones. We conjecture that the non-contiguous property of the valid regions in the higher-dimensional cases (d>1) is responsible for the intractability. If that is true, then we could easily find how to restrict the language in order to have any tractable case. The current methodology of studying such tractable sub-cases is semi-empirical (partly with exhaustive search by running programs, as in Nebel, 1995) and not generalizable across different STCSPs.

Note that although we are talking about complete relationship between all points "instantiated" in the space (before a new point is to be inserted) we are still handling only qualitative relationships. It is not necessary to have real co-ordinates for any point.

The fundamental property of the domain (real space in d-dimensions R_d) addressed here is: the space is a cross product of total orders on each dimension. Also, we have presumed dense infinite space. The algorithms may have to be modified if the space is not dense, and not infinite. In applications related to visualization such restrictions may create problems that we have managed to avoid in this article. Also, our algorithms took the advantage of the fact that for a complete non-disjunctive set of relationships between points, their projections on the respective axes will have strict orderings and each of these orderings is independent of the others. This is not the case if we work in the canonical space for time intervals (Ligozat, 1996) where Y-axis represents the end points of intervals and X-axis represents the starting point of the corresponding intervals. Any valid time-interval there lies in a half space above y=x line (indicating end-point must be > starting-point for any interval). This relationship will make the orderings of projections somewhat dependent on each other. An immediate future step in our work is to develop similar algorithms for time intervals (or for generalized intervals with more than two points in an interval).

The contiguous-ness property of valid regions in 1-D creates an interesting practical possibility. Suppose a user provides the input, and the output regions are displayed on a graphical user interface (over a line), where the user is supposed to pick up a valid region (say, by clicking the mouse) in order to insert the next point. Now, if the user wants to play with the possible solutions and "drags" the new point beyond a boundary of the *box*, then that should be interpreted as an attempt to force backtrack and find a new "neighboring" solution. Such a solution should be found by automatically moving the corresponding boundary point over to a different valid region within the corresponding *box* for that boundary point. Since each solution in 1D-case may be considered as a topological sort of the points, the attempt to find a *neighboring* solution demands that we consider a topological structure of such solution space itself. However, as the solutions in a higher dimensional space do not follow the contiguous-ness property as stated in the Lemma 1 for the 1D-case, the topological structure of solutions in n-D space (n>1) would be significantly different from that in the 1D-case, possibly resulting in the intractability of the corresponding CSP problem.

6. Conclusion

In this article we have presented some algorithms for inserting a new point in a space where a set of points already exist. The existing points need not be fixed in the space, rather they could have a relative positioning with respect to each other that is uniquely represented with a total order of their projections on each of the axes of the Cartesian-space. The input relationships between the new point and each of the existing points are disjunctive. The algorithms output a set of valid regions (if any exist) in the space for the new point, any one of which could be chosen later. These algorithms could be utilized incrementally in order to solve a binary CSP problem within the domain. Such algorithms are useful for doing modeling activity in a multi-dimensional real space, where disjunctive information is provided between point objects. We have implemented the incremental algorithm for the 1D-case and currently implementing the one for the 2D-case. The work is a first attempt toward developing domain-theoretic algorithms in spatio-temporal (or more generally in the continuous-domain) CSP areas, and delving into related consistency issues, e.g., the issues of finding tractable sub-algebras. The article provides some discussion toward this direction.

Acknowledgment: This work is supported by the US National Science Foundation (IIS-9733018).

References

Balbiani, P., and Osmani, A., 2000. "A model for reasoning about topological relations between cyclic intervals." Proceedings of the Seventh *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Breckenridge, CO.

Chittaro, L., and Montanari, A., 2000. "Temporal representation and reasoning in artificial intelligence: issues and approaches." *Annals of Mathematics and Artificial Intelligence.*

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

Ligozat, G., 1996. "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.

Mitra, D., 2001. "A path consistent singleton modeling (CSM) algorithm for arc-constrained networks," to appear in the *Applied Intelligence Journal.*

Nebel, B., 1995, "Reasoning about Temporal Relations: A Maximum Tractable Subclass of Allen's Interval Algebra." *Journal of Association for Computer Machinery*, Vol.42, No.1, 43-66.

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