A Constraint Decomposition Method for Spatio-Temporal Configuration Problems

Toshikazu Tanimoto

Digital Equipment Corporation Japan
1432 Sugao, Akigawa, Tokyo, 197 Japan
tanimoto@jrd.dec.com

Abstract

This paper describes a flexible framework and an efficient algorithm for constraint-based spatio-temporal configuration problems. Binary constraints between spatio-temporal objects are first converted to constraint regions, which are then decomposed into hierarchical data structures; based on this constraint decomposition, an improved backtracking algorithm called HBT can compute a solution quite efficiently. In contrast to other approaches, the proposed method is characterized by the efficient handling of arbitrarily-shaped objects, and the flexible integration of quantitative and qualitative constraints; it allows a wide range of objects and constraints to be utilized for specifying a spatio-temporal configuration. The method is intended primarily for configuration problems in user interfaces, but can effectively be applied to similar problems in other areas as well.

Introduction

Spatio-temporal configuration problems based on constraints arise in many important application areas of artificial intelligence, such as planning, robotics and user interfaces. Although most generic problems are known to be NP-complete, a number of practical techniques suitable for special cases have been developed. In addition to frameworks for constraint-based geometric reasoning [13, 14], potentially useful formalisms have been developed for temporal reasoning [1, 4]. As far as configuration problems are concerned, temporal and spatial frameworks share many important features. In fact, a qualitative formalism for temporal reasoning [1] has been extended to spatial qualitative formalisms [9, 19]; a quantitative formalism based on linear binary inequalities [4] has been extended to a multi-dimensional formalism in a similar way [22].

The multi-dimensional extension of temporal formalisms is certainly an attractive approach to spatio-temporal problems. The approach is based on the expectation that a seemingly difficult multi-dimensional problem can be decomposed into a set of 1-dimensional problems, which can be solved much more easily than the original problem. However, direct extensions attempted in [9, 22] have restricted target problems to orthogonal domains; in other words, objects must be rectangular in a certain coordinate system. Obviously, this restriction is not always desirable in practical situations. Thus arises the need for a framework to handle arbitrarily-shaped objects, with little sacrifice in computing resources. Techniques to handle spatial objects have been studied in the area of computational geometry. Spatial objects can be handled efficiently by hierarchical data structures; many powerful techniques have been devised [20]. However, further study will be needed to fit the techniques into the constraint-based framework.

Another attractive feature of the multi-dimensional extension of temporal formalisms is the possibility of integrating qualitative and quantitative constraints. In many practical situations, qualitative expressions are not enough to specify spatio-temporal constraints. This is particularly true when we must solve a configuration problem for user interface objects [8]. On the other hand, it is also true that there are cases where qualitative constraints suffice. It is therefore desirable to handle both qualitative and quantitative constraints in the same framework. There have been attempts to integrate qualitative and quantitative formalisms for temporal reasoning [12, 18]. However, it is not immediately obvious how to extend these frameworks to spatial counterparts.

In this paper, we propose a framework and an algorithm to address these issues; in common with other constraint-based formalisms, the proposed method handles spatio-temporal configurations based on constraints between objects. The method has three stages. Firstly, a binary constraint between two spatio-temporal objects, such as left and after, is converted to a constraint region; this step is called the interpretation of a constraint. Secondly, the constraint region is decomposed into a hierarchical data structure called a constraint region tree; this representation enables the use of metric features of spatio-temporal constraints. Finally, a hierarchical backtracking algorithm called

HBT is applied to region trees; HBT is effective either by itself or in combination with other algorithms, including PC-2.

The proposed method can be characterized by two major advantages; it can efficiently handle arbitrarily-shaped objects, not just rectangular ones; and it integrates quantitative and qualitative constraints through the interpretation of constraints. Although the proposed method is intended primarily for constraint-based spatio-temporal configuration problems in areas such as user interface management, automated animation and multimedia presentation generation, it would be possible to apply the method to a wider range of configuration problems in areas such as scheduling and space planning.

Constraint Regions

In typical spatio-temporal configuration problems, important constraints are binary constraints such as disjoint, left, after and before. If we do not consider the transformation of an object, such as rotation and scaling, this type of constraint can be represented by a region where a displacement between the two objects is restricted. Having mentioned that, we begin by defining several concepts (for generality, we will use real numbers in definitions, rather than integers). Let a be a d-dimensional object, and A be a set of objects; a *configuration* on A is defined as mapping ω: $A \rightarrow \boldsymbol{R}^d$; for each object a, $\omega(a) \in \boldsymbol{R}^d$ is called its *location*. Given objects a_1 and a_2, *constraint* c between a_1 and a_2 is written as $c(a_1,a_2)$. Let C be a set of constraints, and $\boldsymbol{D}$ be a set of subsets of $\boldsymbol{R}^d$; an *interpretation* on C is defined as mapping φ: $C \rightarrow \boldsymbol{D}$; $\varphi(c) \in \boldsymbol{D}$ is called the (*constraint*) *region* of constraint c. A *constraint satisfaction problem* defined by A, C and φ is written as $CSP(A,C,\varphi)$. A concept similar to the constraint region was first suggested by [15]; another was discussed by the name of *admissible region* in [23]. In this paper, we use this concept to handle d-dimensional linear binary constraints.

Definition 1 Given a set of objects A, a set of constraints C and interpretation φ, configuration ω is said to *satisfy* constraint $c(a_1,a_2) \in C$, iff $\omega(a_2)-\omega(a_1) \in \varphi(c)$ holds. If configuration ω satisfies all the constraints in C, it is said to be a *solution* of $CSP(A,C,\varphi)$. ■

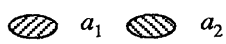
In general, a solution of $CSP(A,C,\varphi)$ can take two forms. One is a configuration satisfying the constraints as in Definition 1, and the other is a set of configurations to bound the locations of objects. The latter form is more generic, but in many situations, the former form suffices. The generic form could be obtained by merging all single solutions.

Below we define a set of *canonical constraints* and their interpretations used in our framework, but constraints and/or interpretations are not necessarily limited to those given. We can define a new constraint (either qualitative or quantitative) and/or interpretation with no changes to our framework. One of the advantages of this approach is flexibility in defining constraints specific to the problem being addressed (see Example 2).

The canonical constraints consist of eight topological and two directional constraints. Topological constraints are defined based on the relationships given in [7]: *disjoint, contains, inside, equal, meet, covers, coveredby* and *overlap*. Directional constraints *less* and *greater* are also defined for each dimension; they can be called *left, right, below, above, before* and *after*, depending on context. Figure 1 shows all the canoni-

constraint	interpretation (constraint region)	depiction
$disjoint(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1 \cap a_2=\varnothing\}$	
$contains(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1 \subseteq a_2^\circ\}$	
$inside(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1^\circ \supseteq a_2\}$	
$equal(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1=a_2\}$	
$meet(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1^\circ \cap a_2^\circ=\varnothing,\ \partial a_1 \cap \partial a_2 \neq \varnothing\}$	
$covers(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1^\circ \subset a_2^\circ,\ \partial a_1 \cap \partial a_2 \neq \varnothing\}$	
$coveredby(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1^\circ \supset a_2^\circ,\ \partial a_1 \cap \partial a_2 \neq \varnothing\}$	
$overlap(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ a_1^\circ \cap a_2^\circ \neq \varnothing,\ a_1^- \cap a_2^- \neq \varnothing\}$	
$less_i(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ \forall l_i,\ sup_i(l_i \cap a_1^\circ) > sup_i(l_i \cap a_2^\circ)\}$	
$greater_i(a_1,a_2)$	$\{x \in \boldsymbol{R}^d \mid x=\omega(a_2)-\omega(a_1),\ \forall l_i,\ inf_i(l_i \cap a_1^\circ) < inf_i(l_i \cap a_2^\circ)\}$	

Figure 1. Canonical Constraints



cal constraints (*contains*(a_1,a_2) should be read as "a_2 *contains* a_1" etc.), their interpretations and exemplary depictions. We assume that all objects are embedded in $\boldsymbol{R}^d$ and are homeomorphic to closed solid spheres. In Figure 1, ∂a, a° and a^- denote the boundary, interior and exterior of object a. For $less_i$ and $greater_i$ (i=1,...,d), l_i indicates a line parallel to i-th coordinate. If $\forall l_i, l_i \cap a_1^\circ = \varnothing \vee l_i \cap a_2^\circ = \varnothing$, the constraint region is defined as $\varnothing$.

It is easy to see that the logical closure (by $\neg$, $\wedge$ and $\vee$) of the canonical constraints for d=1 over single intervals contains Allen's 13 relationships [1] (e.g. "a_2 *started-by* a_1" in Allen's notation can be written as "*covers*(a_1,a_2)$\wedge$*less*(a_2,a_1)" in our notation). As a spatial formalism, the canonical constraints are much more powerful than linear constraints between the surfaces of polyhedra [17], but in some cases, cross-product relationships of Allen's 13 relationships can be more suitable [9]. Note that the cross-product relationships are in fact interpretable and can be added to our framework as well.

Example 1 Suppose the following binary constraints between 2-dimensional objects a_1, a_2, a_3 and a_4 shown in Figure 2 (in this paper, for simplicity, all examples are taken from d=2 cases, though they appear spatial rather than spatio-temporal):

c_1: *inside*(a_1,a_2)$\vee$*coveredby*(a_1,a_2)
c_2: *inside*(a_1,a_3)$\vee$*coveredby*(a_1,a_3)
c_3: *inside*(a_1,a_4)$\vee$*coveredby*(a_1,a_4)
c_4: *disjoint*(a_2,a_3)$\wedge$*right*(a_2,a_3)
c_5: *disjoint*(a_3,a_4)$\wedge$*below*(a_3,a_4)
c_6: *disjoint*(a_2,a_4)

A possible solution ω satisfying the above constraints is also shown in Figure 2. In order to find ω, we first have to obtain the constraint regions of the above constraints as follows:

1) determine the reference points of objects (usually their bottom-left corners); $\omega(a)$ is represented by coordinate values of the reference point of a.
2) compute a constraint region for each item in a constraint expression using the interpretations in Figure 1.
3) if the constraint expression contains logical operations, perform corresponding set operations between the regions.

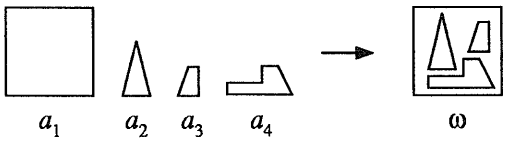


Figure 2. A Sample Configuration Problem

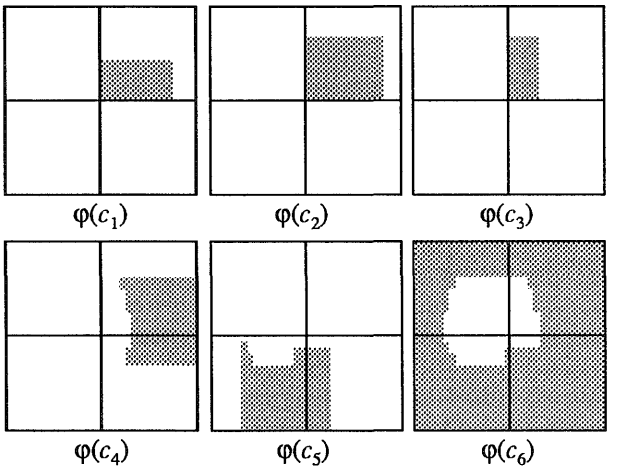


Figure 3. Constraint Regions

The objects are usually pre-processed to employ algorithms such as plane sweeping [20] in Step 2 (in some cases, however, it can be more efficient to compute a constraint region directly from the arithmetic definitions of objects). Figure 3 shows the resulting constraint regions (meshed area) obtained after discretizing the objects. ■

In our framework, all constraints are eventually represented by their constraint regions, which are simply subsets of $\boldsymbol{R}^d$. The interpretation of constraints is basically a geometric problem. This framework is much simpler than existing approaches to integrate qualitative and quantitative constraints. The following example is quantitative constraints which can be used in combination with the canonical constraints:

Example 2 Distance constraints could be defined as:

$near(a_1,a_2) \Leftrightarrow disjoint(a_1,a_2) \wedge |\omega(a_2)-\omega(a_1)| < \delta_1$
$far(a_1,a_2) \Leftrightarrow disjoint(a_1,a_2) \wedge |\omega(a_2)-\omega(a_1)| > \delta_2$

where δ_1 and δ_2 are constants. ■

Note that a unary constraint to specify an absolute location of an object can be represented as a binary quantitative constraint between the object and a special object used as a reference frame.

Constraint Region Trees

Once all constraints are converted to constraint regions, the next step is to construct data structures suitable for constraint satisfaction algorithms. Due to metric features of spatio-temporal problems, we expect that points located close to each other within a constraint region will behave similarly as far as constraint satisfaction is concerned (in other words, as far as the existence of a solution including the point is concerned). We define a partial order relation between problems to make use of this similarity. Given *CSP*(A,C,φ) and *CSP*(A,C,φ'), if the inconsistency of

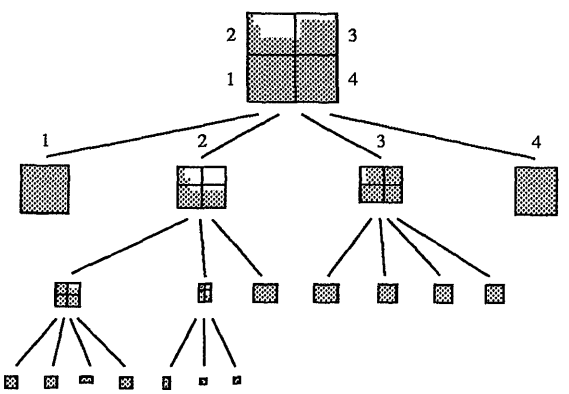


Figure 4. A Constraint Region Tree

$CSP(A,C,\varphi)$ implies the inconsistency of $CSP(A,C,\varphi')$, $CSP(A,C,\varphi)$ is said to *dominate* $CSP(A,C,\varphi')$. Clearly, if $\forall c \in C$, $\varphi(c) \supseteq \varphi'(c)$ holds, $CSP(A,C,\varphi)$ dominates $CSP(A,C,\varphi')$.

If we can construct a hierarchy of problems where any higher level problem dominates its lower level problems, the entire problem will be solved in a hierarchical way. In order to construct such a hierarchy, we employ hierarchical data structures developed in computational geometry. There are many such structures designed for specific purposes [2, 5, 6, 11, 20], but all of them have in common the use of recursive decomposition. We define one of the variations, called a (*constraint*) *region tree*.

Definition 2 Given a constraint region $\varphi(c)$, the *region tree* of $\varphi(c)$ is a tree where each node corresponds to a d-rectangle; it is recursively constructed as follows:

1) if the region is a d-rectangle, return a node corresponding to the d-rectangle.
2) create a node corresponding to the minimum d-rectangle including the region.
3) divide each edge of the d-rectangle into two intervals.
4) obtain sub-regions by dividing the region with 2^d sub-d-rectangles.
5) do 1-5 for all the sub-regions (children nodes).
6) return the node with its children. ■

If each edge is always divided into two intervals of the same length, the region tree is equivalent to a quadtree (d=2) or octree (d=3) corresponding to the constraint region [20]. However, in cases where constraint regions are dominantly rectilinear, division at the corner of constituent rectangles can reduce the size of the region tree and can significantly improve computational efficiency. The maximum length of the edges of rectangles corresponding root nodes is written as r, which is called *range* (the multi-dimensional extension of *range* in [4]).

Note that by bounding and dividing each interval using rational numbers, all extreme points of rectangles can be represented by multiples of a unit interval; we assume this condition in the rest of this paper (a unit d-rectangle will be called a *pixel*).

Example 3 Figure 4 shows a constraint region tree constructed from $\varphi(c_5)$ in Figure 3; non-leaf nodes are divided in a similar way to quadtrees and all nodes correspond to rectangles. ■

A hierarchy of problems is constructed as follows: given $CSP(A,C,\varphi)$ and its region trees, by selecting a node from the region tree of $\varphi(c)$ and defining $\varphi'(c)$ as the rectangle corresponding to the node, we obtain $CSP(A,C,\varphi')$ as a *sub-problem* of $CSP(A,C,\varphi)$.

Lemma 1 Given $CSP(A,C,\varphi)$ and its region trees, for two sub-problems $CSP(A,C,\varphi_1)$ and $CSP(A,C,\varphi_2)$, if $\forall c \in C$, the node corresponding to $\varphi_2(c)$ is a descendant of the node corresponding to $\varphi_1(c)$, $CSP(A,C,\varphi_1)$ dominates $CSP(A,C,\varphi_2)$. ■

Proof By definition, if the node corresponding to $\varphi_2(c)$ is a descendant of the node corresponding to $\varphi_1(c)$, $\varphi_1(c) \supseteq \varphi_2(c)$. Therefore, $CSP(A,C,\varphi_1)$ dominates $CSP(A,C,\varphi_2)$. ■

A constraint region can be any subset of $\boldsymbol{R}^d$. However, if all constraint regions are *simple polygons* (polygons with non-intersecting edges and without holes [20]; we assume that the size of their boundaries is $O(r^{d-1})$), a problem becomes much easier to solve. Space and time requirements for the region tree are as follows:

Theorem 1 [20] The region tree of a constraint region has $O(r^{d-1})$ nodes (rectangles) if the region is a simple polygon, and $O(r^d)$ nodes (rectangles) in general. The construction time is $O(dr^{d-1})$ and $O(dr^d)$ respectively. ■

Hierarchical Backtracking

We can now move on to constraint satisfaction algorithms. Based on the results obtained in the previous section, constraint regions in $CSP(A,C,\varphi)$ are represented by a set of constraint region trees. The primary method adopted here is the framework developed in [4]. Let us briefly review their framework.

A simple temporal constraint satisfaction problem *STP* is defined by a set of variables A and a set of constraint intervals C. *STP* is represented by a *constraint network* where nodes are $a_1,\ldots, a_n \in A$ and each edge $i \rightarrow j$ is labeled with $c_{ij} \in C$. Its constraint network can be transformed into a directed distance graph G by labeling edge $i \rightarrow j$ with $sup(c_{ij})$ and edge $j \rightarrow i$ with $-inf(c_{ij})$. An *STP* is consistent, iff its directed distance

graph G has no negative cycles. If $c'_{ij} \subseteq c_{ij}$ for any corresponding intervals $c_{ij} \in C$ and $c'_{ij} \in C'$ of *STP* and *STP'*, *STP'* is said to be tighter than *STP*. A constraint network of the tightest problem of all equivalent (having the same set of solutions) problems to an *STP* is called its *minimal network*. A solution of an *STP* can be constructed from its minimal network, which can be computed by Warshall-Floyd's algorithm:

Theorem 2 [4] *STP* can be solved in $O(n^3)$ time, where $n=|A|$. ■

Getting back to our own framework, if all the constraint regions of $CSP(A,C,\varphi)$ are d-rectangular, $CSP(A,C,\varphi)$ is said to be *simple*. Simple $CSP(A,C,\varphi)$ is the multi-dimensional counterpart of *STP* and can be solved by decomposing it into d *STP*s and solving the *STP*s separately.

Theorem 3 [22] Simple $CSP(A,C,\varphi)$ can be solved in $O(dn^3)$ time, where $n=|A|$. ■

A general problem can be solved by applying a backtracking algorithm to the meta constraint satisfaction problem, where a domain for each variable is a set of leaf nodes (d-rectangles) in the region tree. By using a classical backtracking algorithm (referred as BT) given in [4], we obtain:

Theorem 4 A solution of $CSP(A,C,\varphi)$ is obtained in $O(dn^3r^{m(d-1)})$ time if all the constraint regions are simple polygons and in $O(dn^3r^{md})$ time in general, where $n=|A|$ and $m=|C|$. ■

Proof According to Theorem 1, $\varphi(c)$ has $O(r^{d-1})$ rectangles if it is a simple polygon. In the worst case, the backtracking algorithm checks all combinations, which are $O(r^{m(d-1)})$. Each step takes $O(dn^3)$ time according to Theorem 3. Thus total time complexity is $O(dn^3r^{m(d-1)})$. Similarly, total time complexity is $O(dn^3r^{md})$ in general. ■

Backtracking algorithms incorporating techniques such as variable ordering, value ordering and network-based heuristics have been used to improve average performance in constraint satisfaction [3, 10]. However, most of them are designed for discrete domain problems and cannot utilize metric features of a problem. As described in the previous section, after constructing region trees from constraint regions, each node in a constraint region tree corresponds to a rectangle; any sub-problem is in fact simple and can be solved in polynomial time (according to Theorem 3). By using this characteristic, average running time can be significantly improved.

Based on the preparation we have made so far, we define a hierarchical backtracking algorithm (abbreviated as HBT). *SOLVE-SIMPLE* computes a solution for each simple sub-problem and returns *true* if the sub-problem is consistent and *false* if inconsistent. Algorithm HBT consists of two functions: *Forward* and *Backward*. It uses a stack to store a search path as a set of pairs (S,T), where S and T are sets of nodes. Given all the region trees, it starts by calling *Forward* with initial stack (S_0,T_0), where S_0={*all children of one of the root nodes*} and T_0={*all the root nodes except for the parent of* S_0}.

Once a sub-problem is known to be inconsistent, HBT does not search sub-problems dominated by the inconsistent sub-problem. The performance of HBT depends on the distribution of possible solutions over constraint regions. The worst case happens when all non-terminal sub-problems are consistent.

Algorithm HBT

Forward
  do
    if S is not empty then select and delete s from S
    else Backward
  until SOLVE-SIMPLE($T\cup\{s\}$)
  $T\leftarrow$ {*all non-leaf nodes in* $T\cup\{s\}$}
  if T is empty then return
  else select and delete t from T
    $S\leftarrow$ {*all children of t*}
    pushdown (*S,T*) *and Forward*
Backward
  if stack is empty then exit
  else popup (*S,T*)
  if S is not empty then Forward
  else Backward ■

HBT is independent of other performance improvement schemes and can be used in combination with them; in the above algorithm description, a value ordering scheme can be applied to select s from S, and a variable ordering scheme can be applied to select t from T. HBT can also be used in combination with other algorithms such as path consistency algorithms, which will be discussed in the next section.

Path Consistency

Computing a path consistent network of the original $CSP(A,C,\varphi)$ is a powerful pre-processing technique for constraint satisfaction problems [15, 16]. Path consistency in our framework is defined as follows:

Definition 3 A path $i(1),\ldots, i(k)$ is consistent iff for any pair of $\omega_{i(1)}$, $\omega_{i(k)}$ such that $\omega_{i(k)}-\omega_{i(1)} \in \varphi_{i(1)i(k)}$ there exists configuration ω such that $\omega(a_{i(1)})=\omega_{i(1)}$, $\omega(a_{i(k)})=\omega_{i(k)}$, $\omega_{i(j+1)}-\omega_{i(j)} \in \varphi_{i(j)i(j+1)}$, $j=1,\ldots,k-1$, where φ_{ij} is the constraint region of constraint $c_{ij}=c(a_i,a_j)$. ■

$CSP(A,C,\varphi)$ is path-consistent iff every path is path consistent. Path consistency algorithms do not guarantee finding a solution; however, they are quite useful for pre-processing. These algorithms require two operations: intersection $\varphi_1 \oplus \varphi_2$ and composition $\varphi_1 \otimes \varphi_2$.

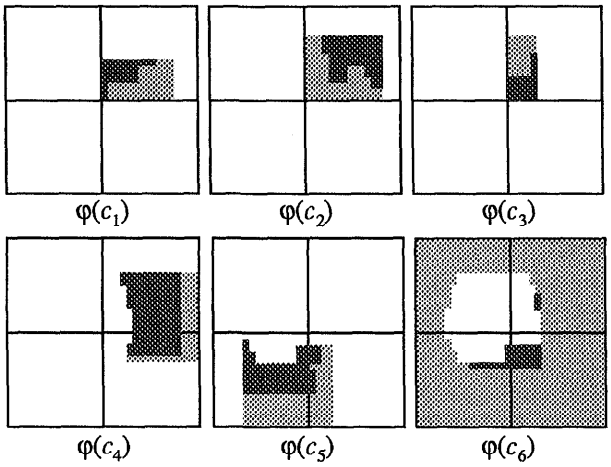


Figure 5. Path Consistent Regions

The two operations in our framework are defined as:

$\varphi_1 \oplus \varphi_2 = \{x \in \boldsymbol{R}^d | x \in \varphi_1, x \in \varphi_2\}$

$\varphi_1 \otimes \varphi_2 = \{x \in \boldsymbol{R}^d | x = x_1 + x_2, x_1 \in \varphi_1, x_2 \in \varphi_2\}$

The distributivity of $\otimes$ over $\oplus$ does not hold except in cases where both φ_1 and φ_2 are rectangles. Moreover, when performed between region trees, $\oplus$ does not preserve the simplicity of regions. With regard to the efficiency of the two operations, we obtain:

Lemma 2 When performed between region trees, $\varphi_1 \oplus \varphi_2$ and $\varphi_1 \otimes \varphi_2$ can be obtained in $O(dr^{2d})$ time. ■

Proof In general, φ_1 and φ_2 have $O(r^d)$ nodes (or corresponding rectangles). $\varphi_1 \oplus \varphi_2$ and $\varphi_1 \otimes \varphi_2$ require $O(r^{2d})$ operations between two rectangles and create $O(r^{2d})$ rectangles; the resulting region tree can be constructed from the rectangles in $O(dr^{2d})$ time. ■

Note that if quadtrees (or their d-dimensional equivalents) are used as region trees, $\varphi_1 \oplus \varphi_2$ can be obtained by comparing corresponding nodes in two trees, and the time complexity of $\varphi_1 \oplus \varphi_2$ can be reduced to $O(dr^d)$.

An efficient path-consistency algorithm called PC-2 is described in [15]. Relaxation operation *REVISE*$((i,k,j))$ is defined as $\varphi_{ij} \leftarrow \varphi_{ij} \oplus \varphi_{ik} \otimes \varphi_{kj}$, using intersection and composition defined above. *REVISE* returns *true* if φ_{ij} is modified, otherwise it returns *false*. *RELATED-PATHS* returns the set of 2-length paths relevant to the changed path (see [15] for details). PC-2 is formulated as:

Algorithm PC-2 [15]

$Q \leftarrow \{(i,k,j) | (i<j) \wedge (k \neq i,j)\}$
while Q is not empty do
  select and delete a path (i,k,j) *from Q, and*
  if REVISE$((i,k,j))$ *then*
    $Q \leftarrow Q \cup$ *RELATED-PATHS*$((i,k,j))$ ■

With regard to the efficiency of PC-2, we obtain:

Theorem 5 Algorithm PC-2 computes the path consistent network of $CSP(A,C,\varphi)$ in $O(dn^3r^{3d})$ time. ■

Proof Algorithm PC-2 needs $O(n^3r^d)$ steps to terminate, because in the worst case, for each constraint region, which can have $O(r^d)$ pixels, may be decreased by only a pixel in a single *REVISE* operation. According to Lemma 2, each *REVISE* operation consumes $O(dr^{2d})$ time. Total time is therefore $O(dn^3r^{3d})$. ■

Theorem 5 is the multi-dimensional counterpart of Theorem 5.7 in [4].

Example 4 PC-2 can be applied to the sample problem as follows:

1) construct region trees as in Figure 4 from the constraint regions in Figure 3, which are used as initial values for PC-2.
2) each time *REVISE* is called, $\oplus$ and $\otimes$ operations are performed between sets of rectangles in the region trees.

At the termination of PC-2, we obtain resulting path consistent constraint regions (constraint regions of a path consistent network) illustrated as in Figure 5 (darkly meshed area). ■

Discussions

Table 1 summarizes the above results about worst-case time complexity (the left cell of (H)BT is complexity when all constraint regions are simple polygons; the right is complexity in general). However, by storing solutions of non-terminal sub-problems (for HBT) or partial instantiations (for BT), we can reduce the worst-case time complexity of (H)BT to $O(dn^2r^{m(d-1)})$ and $O(dn^2r^{md})$ respectively.

Empirical results from 192 small-scale problems (d=2, range=16, the number of objects=4, quadtrees are used as region trees) are shown in Table 2. Performance is measured by the number of times *SOLVE-SIMPLE* is called to find a solution (denominated by 1,000 calls) and is averaged over 192 cases (the average number of rectangles corresponding to leaf nodes in a region tree was 33, and PC-2 could reduce it to 19). Although a full-scale experimental analysis should be done in the future, Table 2 well suggests

Table 1. Worst-Case Time Complexity

(H)BT		*REVISE*	PC-2
$O(dn^3r^{m(d-1)})$	$O(dn^3r^{md})$	$O(dr^{2d})$	$O(dn^3r^{3d})$

Table 2. Performance Comparisons

BT	PC-2+BT	HBT	PC-2+HBT
156	5	2	1

that HBT is more efficient than the combination of PC-2 and BT. In combination with PC-2, HBT works even better. Taking into consideration that PC-2 is not inexpensive, HBT without PC-2 would suffice for many practical problems.

Conclusion

In this paper, we have proposed a new method to solve constraint-based spatio-temporal configuration problems; the method centers around the interpretation of constraints and the decomposition of constraint regions into region trees. In contrast to other approaches, our method is more flexible and efficient; the flexibility lies in defining interpretation procedures most suitable for given problems; the efficiency results from employing region trees as the basis for the hierarchical backtracking algorithm HBT. It is suggested that HBT is more efficient than the combination of classical backtracking and path consistency algorithms. However, further research in a couple of different directions will be necessary to refine the proposed framework and algorithm: the capability of handling a wider range of geometric operations such as rotation and scaling; and the empirical evaluation of performance when HBT is used in combination with other techniques including network-based heuristics.

References

[1] Allen, J.F. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26 (1983), 832-843.

[2] Chiang, Y.-J. and Tamassia, R. Dynamic Algorithms in Computational Geometry. *Proc. IEEE*, 80 (1992), 1412-1434.

[3] Dechter, R. and Pearl, J. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34 (1988), 1-38.

[4] Dechter, R., Meiri, I. and Pearl, J. Temporal Constraint Networks. *Artificial Intelligence*, 49 (1991), 61-95.

[5] Edelsbrunner, H. A New Approach to Rectangle Intersections Part I. *Intern. J. Computer Math.*, 13 (1983), 209-219.

[6] Edelsbrunner, H. A New Approach to Rectangle Intersections Part II. *Intern. J. Computer Math.*, 13 (1983), 221-229.

[7] Egenhofer, M.J. and Al-Taha, K.K. Reasoning about Gradual Changes of Topological Relationships. In Frank, A.U., Campari, I. and Formentini, U. (eds.) *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, Springer-Verlag, Berlin, Germany, 1992.

[8] Freeman-Benson, B.N., Maloney, J. and Borning, A. An Incremental Constraint Solver. *Commun. ACM*, 33 (1990), 54-63.

[9] Guesgen, H.W. and Hertzberg, J. *A Perspective of Constraint-Based Reasoning*. Springer-Verlag, Berlin, Germany, 1992.

[10] Haralick, R.M. and Elliott, G.L. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14 (1980), 263-313.

[11] Hunter, G.M. and Steiglitz, K. Operations on Images Using Quad Trees. *IEEE Trans. Pattern Anal. and Machine Intell.*, 1 (1979), 145-153.

[12] Kautz, H.A. and Ladkin, P.B. Integrating Metric and Qualitative Temporal Reasoning. In *Proc. AAAI '91* (Anaheim, CA, 1991), 241-246.

[13] Kin, N., Takai, Y. and Kunii, T.L. PictureEditor II: A Conversational Graphical Editing System Considering the Degree of Constraint. In Kunii, T.L. (ed.) *Visual Computing*. Springer-Verlag, Tokyo, Japan, 1992.

[14] Kramer, G.A. A Geometric Constraint Engine. *Artificial Intelligence*, 58 (1992), 327-360.

[15] Mackworth, A.K. Consistency in Networks of Relations. *Artificial Intelligence*, 8 (1977), 99-118.

[16] Mackworth, A.K. and Freuder, E.C. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25 (1985), 65-74.

[17] Malik, J. and Binford, T.O. Reasoning in Time and Space. In *Proc. IJCAI '83* (Karlsruhe, Germany, 1983), 343-345.

[18] Meiri, I. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Proc. AAAI '91* (Anaheim, CA, 1991), 260-267.

[19] Mukerjee, A. and Joe, G. A Qualitative Model for Space. In *Proc. AAAI '90* (Boston, MA, 1990), 721-727.

[20] Samet, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[21] Shahookar, K. and Mazumder, P. VLSI Cell Placement Techniques. *ACM Computing Surveys*, 23 (1991), 143-220.

[22] Tanimoto, T. Configuring Multimedia Presentations Using Default Constraints. In *Proc. PRICAI '92* (Seoul, Korea, 1992), 1086-1092.

[23] Tokuyama, T., Asano, T. and Tsukiyama, S. A Dynamic Algorithm for Placing Rectangles without Overlapping. *J. of Information Processing*, 14 (1991), 30-35.