A Spatial Search Framework for Executing Perceptions and Actions in Diagrammatic Reasoning*

Bonny Banerjee and B. Chandrasekaran

Lab for AI Research, Department of Computer Science & Engineering
The Ohio State University, Columbus, OH 43210, USA
Banerjee.28@osu.edu, Chandra@cse.ohio-state.edu

Abstract. Diagrammatic reasoning (DR) requires perceiving information from a diagram and modifying/creating objects in a diagram according to problem solving needs. The perceptions and actions in most DR systems are hand-coded for the specific application. The absence of a general framework for executing perceptions/actions poses as a major hindrance to using them opportunistically. Our goal is to develop a framework for executing a wide variety of specified perceptions and actions across tasks/domains without human intervention. We observe that the domain/task-specific perceptions/actions can be transformed into domain/task-independent spatial problems. In our framework, a human problem solver specifies a spatial problem as a quantified constraint satisfaction problem (QCSP) in the real domain using an open-ended vocabulary of properties, relations and actions involving three types of spatial objects – points, curves, regions. Traditional approaches solve such QCSPs by computing the equivalent quantifier-free algebraic expression, the complexity of which is inherently doubly exponential. In this paper, we investigate a domain-independent framework of spatial search for solving $2D$ spatial problems specified as QCSPs. The framework searches for the solution in the space of the diagram instead of in the space of algebraic equations/inequalities. We prove the correctness of our approach and show that it is more efficient than cylindrical algebraic decomposition, a well-known algebraic approach, by executing perceptions/actions in two army applications.

1 Introduction

Reasoning with diagrams requires the use of symbolic knowledge representations and inferences as well as visually perceiving information from a diagram and creating/modifying spatial objects satisfying certain constraints. Complex problems are solved opportunistically as different subtasks are solved by the component – symbolic or visual – that is more suitable for it. The focus in this

* This research was partially supported by participation in the Advanced Decision Architectures Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAD19-01-2-0009.

A.K. Goel, M. Jamnik, and N.H. Narayanan (Eds.): Diagrams 2010, LNAI 6170, pp. 144–159, 2010.

paper is on the visual component that extracts (or perceives) information from a diagram and creates/modifies (or acts on) a diagram to satisfy given informational constraints. Accomplishing these perceptions and actions require solving domain-independent spatial problems. For example, an army commander, while planning strategic operations, might need to know the portions of a path prone to ambush (a.k.a. $RiskyPortionsofPath$). The visual component has to solve appropriate spatial problems – compute the set of points q on a curve (the path) c such that q is within a specified distance (the firepower range) d from some point p and the line segment $\{p, q\}$ intersects a region r (i.e., p is behind r with respect to q) – to deliver the required information. A schematic diagrammatic reasoning (DR) architecture from [1] is shown in Fig. 1.

In the last couple of decades, numerous DR systems have been built for different applications in different domains, such as, SKETCHY [2] for graph understanding in thermodynamics and economics, REDRAW [3] for structural analysis in civil engineering, ARCHIMEDES [4] for proving theorems in Euclidean geometry, DIAMOND [5] for proving certain mathematical theorems by induction, MAGI [6] for symmetry detection, and so on. All of these DR systems require perceiving from and/or acting on diagrams, where each perception/action requires solving a domain-independent spatial problem. How were these spatial problems solved in the DR systems?

Typically, the human developing a DR system identifies a priori the problem solving steps including a set of perceptions and actions, and hand-codes algorithms for solving each of them. If the problem solving steps need to be altered in future and as a result, a new perception arises, the developer has to write a new algorithm for solving it. Clearly, this is inconvenient and does not allow fast and easy experimentation with different problem solving strategies for the same problem. These drawbacks are further magnified when the goal is to build a general-purpose DR system where a very large variety of perceptions and actions are possible which is not feasible to ascertain a priori, and develop and store algorithms for. Hence, our goal is to investigate:

1. A language for a human to specify a variety of spatial problems, and
2. A general domain-independent framework for efficiently solving spatial problems, specified in the above language, without human intervention and presenting the solution in visual form whenever possible.

Definition 1 ***Diagram.*** *(from [1]) A diagram $\mathcal{D}$ is a set of labeled 2D spatial objects $\{\mathcal{O}_1, \mathcal{O}_2, ...\mathcal{O}_l\}$. A spatial object $\mathcal{O}$ is a 3-tuple $< \mathcal{L}, \mathcal{T}, \mathcal{E} >$ where $\mathcal{L}$ is a label, $\mathcal{T}$ is a type (point, curve or region), and $\mathcal{E}$ is its spatial extent. The spatial extent of an object is the set of points constituent of the object. Further, diagrammatic image, $\mathcal{I}(\mathcal{D})$, is the set of points constituent of all objects in $\mathcal{D}$.*

We are interested only in monochromatic diagrams with no intensity variation. A perception is a mapping from a diagram $\mathcal{D}$ to a set of booleans $\{True, False\}$ or real numbers $\Re$. An action is a mapping from $\mathcal{D}$ to a new set of spatial objects or diagram $\mathcal{D}'$, where $\mathcal{I}(\mathcal{D}) \neq \mathcal{I}(\mathcal{D}')$.

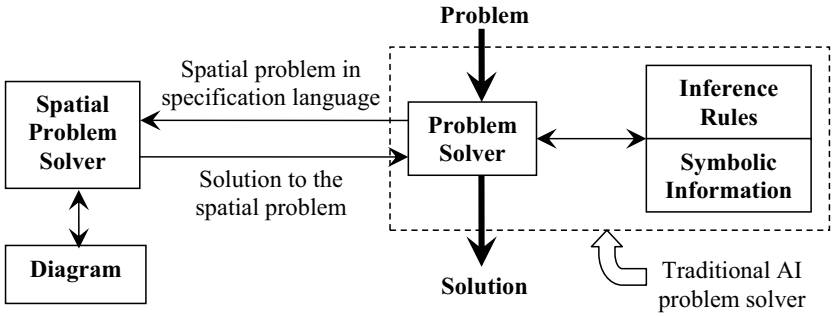


Fig. 1. A schematic diagrammatic reasoning architecture

In the next section, we briefly visit the specification language (borrowed from our earlier work) and discuss the challenges that arise in solving spatial problems from such specification. We have identified a small vocabulary of predicates which are hand-coded internally as procedures (as opposed to declarative sentences). A spatial problem is specified in the specification language using a fixed set of operators and the predicates in the vocabulary. In section 3, a general domain-independent framework of spatial search – the main focus of this paper – is described along with its correctness analysis. In this framework, unlike algebraic approaches, the solution is searched for in the space of the diagram instead of in the space of algebraic equations/inequalities. Enhancements are proposed to improve complexity. The enhanced algorithm is shown to be more efficient than cylindrical algebraic decomposition (CAD), a well-known algebraic approach, in executing perceptions/actions for two army applications.

2 Specification Language

The specification language [7] is a high-level language that is extensible, human-usable, and expressive enough to describe a wide variety of $2D$ spatial problems. The problems will be accepted as input by the spatial problem solver (SPS) and solved without human intervention. The language recognizes boolean operators $\{\wedge, \vee, \neg\}$, arithmetic operators $\{+, -, \times, \div\}$, relational operators $\{<, >, =, \neq\}$, and quantifiers $\{\exists, \forall\}$. The brackets () are used to express precedence while the brackets $\{\}$ are used for a set. The domain is real plane, $\Re^2$.

2.1 Vocabulary

From the literature [1,7,8,9,10], we have extracted a minimal vocabulary of properties, relations and actions based on their wide usage in expressing spatial problems in different domains. The vocabulary is not claimed to be complete and addition of new properties/relations/actions is allowed when a problem cannot be easily expressed using the existing ones. Currently, the properties, relations and actions in our vocabulary are as follows.

Properties. Associated with each type of object are a few properties – location $\mathcal{E}(p)$ of a point p, locations $\mathcal{E}_x(p)$ and $\mathcal{E}_y(p)$ of p along x- and y-axis respectively; spatial extent $\mathcal{E}(c)$ and length $Length(c)$ of a curve c; and spatial extent $\mathcal{E}(r)$, area $Area(r)$ and periphery $Periphery(r)$ of a region r, where the periphery of a region refers to its boundary curve.

Relations. The vocabulary contains the relations: $Distance(p_1, p_2)$, $Angle(p_1, p_2, p_3)$, $Leftof(p_1, p_2)$, $Rightof(p_1, p_2)$, $Above(p_1, p_2)$, $Below(p_1, p_2)$, $On(p, c)$ and $Inside(p, r)$, where p, p_1, p_2, p_3 are points, c a curve and r a region.

Actions. There are four actions for modifying objects. $Translate(\mathcal{O}, t_x, t_y)$ returns a translation of object $\mathcal{O}$ for t_x units along x-axis and t_y units along y-axis, $Rotate(\mathcal{O}, c, \theta)$ returns a rotation of the object $\mathcal{O}$ with respect to point c as center for θ degrees in the anti-clockwise direction, $Reflect(\mathcal{O}, \{a, b\})$ returns a reflection of object $\mathcal{O}$ with respect to the line segment $\{a, b\}$, $Scale(\mathcal{O}, c, s_x, s_y)$ returns a scaling of the object $\mathcal{O}$ with respect to point c for s_x units along x-axis and s_y units along y-axis.

2.2 The Language

We will use a functional constraint logic programming language (first-order predicate logic) as the specification language [7].

Quantified Constraint Satisfaction Problem. An instance of a constraint satisfaction problem (CSP) consists of a tuple $< \mathcal{V}, D, \mathcal{C} >$ where $\mathcal{V}$ is a finite set of variables, D is a domain, and $\mathcal{C}= \{\mathcal{C}_1, ...\mathcal{C}_k\}$ is a set of constraints. A constraint $\mathcal{C}_i$ consists of a pair $< \mathcal{S}_i, \mathcal{R}_i >$ where $\mathcal{S}_i$ is a list of m_i variables and $\mathcal{R}_i$ is a m_i-ary relation over the domain D. The question is to decide whether or not there is an assignment mapping each variable to a domain element such that all the constraints are satisfied. All of the variables in a CSP can be thought of as being implicitly existentially quantified.

A useful generalization of the CSP is the quantified constraint satisfaction problem (QCSP), where variables may be both existentially and universally quantified. An instance of the QCSP consists of a quantified formula in first-order logic, which consists of an ordered list of variables with associated quantifiers along with a set of constraints. A QCSP can be expressed as follows:

$$\phi(v_1, ...v_m) \equiv Q(x_n, ...x_1)\phi'(v_1, ...v_m, x_1, ...x_n)$$
$$Q(x_n, ...x_1) \equiv Q_n x_n, ...Q_1 x_1$$

where $Q_i \in \{\forall, \exists\}$, $\{x_1, ...x_n\}$ is the set of quantified variables, $\{v_1, ...v_m\}$ is the set of free variables, $\mathcal{V}= \{v_1, ...v_m, x_1, ...x_n\}$, and ϕ' is a quantifier-free expression called the matrix. Such representation of a quantified expression ϕ, where it is written as a sequence of quantifiers followed by the matrix, is referred to as prenex form.

Definition 2 ***Spatial Problem.*** *A spatial problem is a QCSP where a variable (quantified or free) can only be of type point, and the domain is* $\Re^2$.

Thus, a spatial problem ϕ is a mapping from a diagram $\mathcal{D}$ satisfying a logical combination of constraints $\mathcal{C}$ to a set of booleans $\{True, False\}$ or real numbers $\Re$ or spatial objects $\mathcal{D}'$, i.e.,

$$\phi : \mathcal{D} \xrightarrow{\mathcal{C}} \{True, False\} \cup \Re \cup \mathcal{D}'$$

Solving a spatial problem requires solving a QCSP. Algebraic approaches to solving a QCSP eliminate quantifiers and solve algebraic equations/inequalities to arrive at the most simplified expression. The computational bottleneck is real quantifier elimination (QE) which is inherently doubly exponential in time complexity even when there is only one free variable and all polynomials in the quantified input are linear [11]. The most general and elaborately implemented method for real QE is CAD [12], complexity of which is $(sd)^{O(1)^{k-1}}$ where s is the number of polynomials, their maximum degree is d and coefficients are real, and k is the number of blocks of quantified variables. For large real-world problems, it soon becomes too time consuming to solve a QCSP.

3 Spatial Problem Solver (SPS)

We propose a general framework of spatial search for efficiently solving, without human intervention, a wide variety of spatial problems expressed as QCSPs and presenting the solution in visual form. The solution is searched for in the space of the diagram instead of in the space of algebraic equations/inequalities.

Decision and Function problems. In the specification language, a spatial problem ϕ is expressed as a QCSP where $\mathcal{V}$ consists of variables of type point, $D = \Re^2$, and $\mathcal{C}$ is the set of constraints to be staisfied. Solving a spatial problem involves:

1. When there are no free variables in $\mathcal{V}$, deciding whether there exists a mapping from $\mathcal{V}$ to D satisfying $\mathcal{C}$.
2. When there are free variables in $\mathcal{V}$, computing the mapping from $\mathcal{V}$ to D satisfying $\mathcal{C}$.

The first case constitutes a decision problem that yields a $True/False$ solution. The second case constitutes a function problem that yields a spatial object described by the free variables as the solution. For example, given a curve c and two points p, q, the spatial problem $BehindCurve(q, c, p)$ is defined as deciding whether or not q is behind c with respect to p. This might be specified as deciding whether or not the curve c and line segment $\{p, q\}$ intersect. Thus,

$$BehindCurve(q, c, p) \equiv Intersect(c, \{p, q\})$$

For particular instances of q, p, c, the solution to this problem is $True/False$ – a decision problem. For particular instances of p, c, and generalized coordinates of q i.e., $q \leftarrow (x, y)$, the solution to the same problem is a region object – a function problem. While a decision problem merely requires checking whether or not a given instance of an object satisfies the constraints or not, a function problem requires computing all instances that satisfy the constraints.

Since there are only point variables, the solution to a function problem can be obtained by solving the corresponding decision problem for every point in the diagram. Only the points for which the decision problem evaluates to $True$ are included in the solution set. Thus, theoretically, if a decision problem can be solved, function problems involving that decision problem can also be solved.

Since the number of points that constitute any diagram is infinite, checking whether or not a decision problem outputs $True$ for each one of them cannot be accomplished in finite time. However, if the space in a diagram is discretized into a finite number of pixel-like elements or *pels* where each pel corresponds to a point, the task can be accomplished in finite time. This reduction in time complexity comes at the cost of precision. The precision of the solution will be dependent on the maximum resolution of the diagram.[1] However, if the resolution can be varied in a problem-dependent manner such that only relevant parts of the diagram are viewed at high resolution, a significant amount of computation time can be saved without compromising precision significantly.

3.1 Underlying Representation and Operators

Pel-object data structure. The space in a diagram is discretized by imposing an array of square pels at the image resolution. Each pel corresponds to a point and is indexed and maintains a list of all spatial objects that it is occupied by. Each object maintains a list of all pels that it occupies. This can be conceptualized as a $2D$ table with the two dimensions corresponding to the objects and the pels, and each entry in the table containing a 1 or 0 depending on whether that pel and object belong to each other or not. Thus, given a pel and an object, their relation can be retrieved in constant time. Size of this table is $N \times l$ where N is the number of pels at image resolution and l the number of objects.

Implementation of predicates. $\mathcal{E}(p)$ returns the x- and y-indices of the pel occupied by point p. $\mathcal{E}(c)$ returns the sequence of indices of pels occupied by curve c from one end to another. $Length(c)$ returns the number of pels in $\mathcal{E}(c)$. $\mathcal{E}(r)$ returns indices of all pels occupied by region r. $Area(r)$ returns the number of pels in $\mathcal{E}(r)$. $Periphery(r)$ returns the sequence of indices of all pels occupied by the boundary curve of r starting from some pel on the boundary. $Distance(p_1, p_2)$ returns the Euclidean distance between $\mathcal{E}(p_1)$ and $\mathcal{E}(p_2)$. $Angle(p_1, p_2, p_3)$ returns the angle at $\mathcal{E}(p_2)$ from $\mathcal{E}(p_1)$ and to $\mathcal{E}(p_3)$. $Leftof(p_1, p_2)$ returns $True$ if $\mathcal{E}_x(p_1) \leq \mathcal{E}_x(p_2)$, otherwise $False$. $Rightof(p_1, p_2)$ returns $True$ if $\mathcal{E}_x(p_1) \geq \mathcal{E}_x(p_2)$, otherwise $False$. $Above(p_1, p_2)$ returns $True$ if $\mathcal{E}_y(p_1) \geq \mathcal{E}_y(p_2)$, otherwise $False$. $Below(p_1, p_2)$ returns $True$ if $\mathcal{E}_y(p_1) \leq \mathcal{E}_y(p_2)$, otherwise $False$. $On(p, c)$ returns $True$ if the data structure contains 1 for $\mathcal{E}(p)$ and c, otherwise $False$. $Inside(p, r)$ returns $True$ if the data structure contains 1 for $\mathcal{E}(p)$ and r, otherwise $False$. $Translate(\mathcal{O}, t_x, t_y)$ returns the set of pels in $\mathcal{E}(\mathcal{O})$ after adding t_x and t_y to the x- and y-indices of each pel. $Rotate(\mathcal{O}, p, \theta)$

[1] This is true for algebraic approaches as well where the complexity depends on s and d (see section 2.2). To keep d low, spatial objects are approximated piecewise-linearly. To keep s low, objects cannot be approximated too closely, thereby losing precision.

returns the set of pels in $\mathcal{E}(\mathcal{O})$ after rotating each pel's indices with respect to p for θ degrees in anticlockwise direction. $Reflect(\mathcal{O}, \{a, b\})$ returns the set of pels in $\mathcal{E}(\mathcal{O})$ after reflecting each pel's indices with respect to line segment $\{a, b\}$. $Scale(\mathcal{O}, p, s_x, s_y)$ returns the set of pels in $\mathcal{E}(\mathcal{O})$ after scaling each pel's indices with respect to p for s_x and s_y units along x- and y-axis. Each time a new object is introduced/modified/deleted, the data structure is updated.

The three spatial search operators. Whether a spatial problem is decision or function is interpreted from the problem specification by the SPS – if the specification contains free variables, it is a function problem, else a decision problem. The SPS also knows how to search when the different kinds of quantifiers occur – using three operators $\{F_\exists, F_\forall, F\}$.

In a decision problem, when an existential quantifier occurs in the specification, i.e. $\phi \equiv \exists x, \phi'(x)$, the SPS searches all pels in the diagram until it finds one that evaluates ϕ' to $True$, when it halts and outputs $True$. If no such pel is found, the SPS outputs $False$. The pels being searched correspond to the existentially quantified point variable. Given a diagram D, a quantifier free expression $\phi'(x)$ such that $\phi \equiv \exists x, \phi'(x)$, and the existentially quantified variable x, the operator $F_\exists$ computes the solution to ϕ, as follows:

$F_\exists$: 1. for $i \leftarrow$ each pel in D
2. if $\phi'(i) = True$,
3. return $True$;
4. return $False$;

The application of $F_\exists$ is written as $F_\exists\ o\ (D, \phi'(x), \{\}, \exists x)$ where the empty set $\{\}$ indicates no free variables. A function problem requires two searches – for each pel in the diagram, the SPS needs to solve the decision problem. The pels for which the decision problem evaluates to $True$ constitute the solution. The operator F collects all those pels from a diagram that satisfy the decision problem. If $\phi(v) \equiv \exists x, \phi'(v, x)$ is a function problem and v is the free variable, F computes the solution to ϕ, as follows:

F : 1. $S \leftarrow \{\}$;
2. for $i \leftarrow$ each pel in D
3. if $F_\exists\ o\ (D, \phi'(i, x), \{\}, \exists x) = True$,
4. $S \leftarrow S \cup \{i\}$;
5. return S;

This is written as $F\ o\ F_\exists\ o\ (D, \phi'(v, x), \{v\}, \exists x)$.

Similarly, for a decision problem with a universal quantifier, i.e. $\phi \equiv \forall x, \phi'(x)$, $F_\forall$ computes the solution to ϕ as follows:

$F_\forall$: 1. for $i \leftarrow$ each pel in D
2. if $\phi'(i) = False$,
3. return $False$;
4. return $True$;

This is written as $F_\forall\ o\ (D, \phi'(x), \{\}, \forall x)$.

When there is more than one quantified variable in a decision problem $\phi \equiv Q(x_n, ...x_1)\phi'(x_1, ...x_n)$, it is first expressed in prenex form to extract $\phi'(x_1, ...x_n)$. Then the problem is solved by the successive application of the operators, $F_\exists$ or $F_\forall$, corresponding to the quantifiers, $\exists$ or $\forall$. For example, a decision problem $\phi \equiv \exists x_2 \forall x_1, \phi'(x_1, x_2)$ is solved as $F_\exists\ o\ F_\forall\ o\ (D, \phi'(x_1, x_2), \{\}, \exists x_2 \forall x_1)$. The corresponding function problem is solved as $F\ o\ F_\exists\ o\ F_\forall\ o\ (D, \phi'(v, x_1, x_2), \{v\}, \exists x_2 \forall x_1)$.

In a $2D$ diagram, there can be at most one free variable in any problem, but a number of quantified variables in arbitrary order. In general, a decision problem is of the form $\phi \equiv Q_n x_n, ...Q_1 x_1 \phi'(x_1 ... x_n)$, and is solved as:

$$F_{Q_n}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(x_1, ...x_{n-1}, x_n), \{\}, Q(x_n, ...x_1))$$

Similarly, a function problem in DR is in general of the form $\phi(v) \equiv Q_n x_n, ...Q_1 x_1$ $\phi'(v, x_1 ... x_n)$ where v is the free variable, and is solved as:

$$F\ o\ F_{Q_n}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(v, x_1, ...x_n), \{v\}, Q(x_n, ...x_1))$$

The three operators automate the process of spatial problem solving. The time complexity of this naive approach is $O(N^k)$ where N is the number of pels in the diagram and k is the total number of free and quantified variables.

3.2 Correctness Analysis

By induction, we will show that the decision problem $\phi \equiv Q(x_n, ...x_1)\phi'(x_1, ...x_n)$, where $Q(x_n, ...x_1) \equiv Q_n x_n ... Q_1 x_1$, $Q_i \in \{\exists, \forall\}$, can be solved by the application of the two operators, $F_\exists$ and $F_\forall$, in the following sequence:

$$F_{Q_n}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(x_1, ...x_n), \{\}, Q(x_n, ...x_1))$$

Using that, we will show that the function problem $\phi \equiv Q(x_n, ...x_1)\phi'(v, x_1, ...x_n)$, where v is a free variable, can be solved by the application of the three operators, F, $F_\exists$ and $F_\forall$, in the following sequence:

$$F\ o\ F_{Q_n}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(v, x_1, ...x_n), \{v\}, Q(x_n, ...x_1))$$

For $n = 1$, the decision problem $\phi \equiv \exists x_1, \phi'(x_1)$ or $\phi \equiv \forall x_1, \phi'(x_1)$. We have shown that these problems can be solved by $F_\exists\ o\ (D, \phi'(x_1), \{\}, \exists x_1)$ and $F_\forall\ o\ (D, \phi'(x_1), \{\}, \forall x_1)$ respectively.

For $n = 2$, the decision problem ϕ can have four cases: $\phi \equiv \exists x_2 \exists x_1, \phi'(x_1, x_2)$, $\phi \equiv \forall x_2 \exists x_1, \phi'(x_1, x_2)$, etc. The first case is solved by $F_\exists\ o\ F_\exists\ o\ (D, \phi'(x_1, x_2), \{\}$, $\exists x_2 \exists x_1)$ and similarly for the other three cases.

Let us assume that the proposition holds for $n = m$. Therefore, the decision problem $\phi \equiv Q(x_m, ...x_1)\phi'(x_1, ...x_m)$ can be solved as:

$$F_{Q_m}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(x_1, ...x_m), \{\}, Q(x_m, ...x_1))$$

Consider the proposition for $n = m + 1$. The decision problem

$\phi \equiv Q(x_{m+1}, ...x_1)\phi'(x_1, ...x_{m+1})$ can have two cases:
$\phi \equiv \exists x_{m+1} Q(x_m, ...x_1)\phi'(x_1, ...x_{m+1})$ or $\phi \equiv \forall x_{m+1} Q(x_m, ...x_1)\phi'(x_1, ...x_{m+1})$.

The first case is solved as $F_{\exists}\ o\ F_{Q_m}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(x_1, ...x_{m+1}), \{\}, \exists x_{m+1}\ Q(x_m, ...x_1))$. Similarly for the other case. Thus, given that the proposition holds for $n = m$, it also holds for $n = m + 1$. But it holds for $n = 1, 2$. Hence it holds for $n = 3, 4, ...$ The proof follows.

Trivially, a function problem $\phi(v) \equiv Q(x_n, ...x_1)\phi'(v, x_1, ...x_n)$ is solved as $F\ o\ F_{Q_n}\ o\ ...\ F_{Q_1}\ o\ (D, \phi'(v, x_1, ...x_n), \{v\}, Q(x_n, ...x_1))$. Thus, the proposition holds for function problems as well.

3.3 Enhancing the Efficiency of Spatial Search

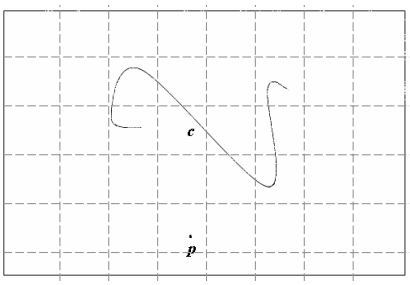
It is impressive, to say the least, how the human visual system executes spatial search so efficiently for such a wide variety of spatial problems. We believe, in addition to parallel computation, a problem guides the visual system to facilitate search by restricting computation to relevant parts of the space. For example, when computing the region behind a curve c with respect to a point p (see Fig. 2), certain areas require more computation than others due to constraints in the problem and not merely due to the configuration of objects in the diagram. In this problem, the space that contain the boundaries of the behind region receives more computation time. The rest of the space, either clearly behind or clearly not, receives less time. In general, a vast majority of time is restricted to computing the precise boundary of the solution.

To implement this strategy, we maintain the pel-object data structure at multiple resolutions. Let $\mathcal{P}^{(i)}$ denote a pel at the i^{th} resolution. Let there be s resolutions, $i = 0$ being the highest resolution, $d \times d$ be the number of $\mathcal{P}^{(i)}$s in one $\mathcal{P}^{(i+1)}$ (i.e. factor of change in resolution). Then, $\mathcal{P}^{(i)}$ contains $d^i \times d^i$ $\mathcal{P}^{(0)}$s. Each $\mathcal{P}^{(i)}$ is indexed by the indices of the four $\mathcal{P}^{(0)}$s at its corners.

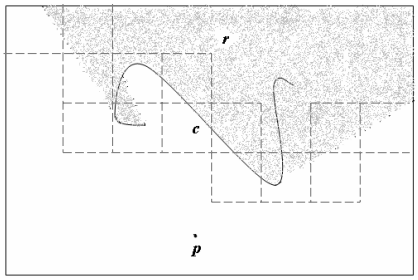
Given a spatial problem, the diagram is processed starting from the lowest resolution. $\mathcal{P}^{(i)}$ is said to satisfy constraints $\mathcal{C}$ when all four of its corner $\mathcal{P}^{(0)}$s satisfy $\mathcal{C}$. $\mathcal{P}^{(i)}$ is said to not satisfy $\mathcal{C}$ when all four of its corner $\mathcal{P}^{(0)}$s do not satisfy $\mathcal{C}$. $\mathcal{P}^{(i)}$ is said to partially satisfy $\mathcal{C}$ when at least one of its corner $\mathcal{P}^{(0)}$s satisfies $\mathcal{C}$ and at least one does not. Thus, $\mathcal{P}^{(i)}$, when checked for satisfaction of constraints, belongs to one of three classes – satisfies, does not satisfy, or partially satisfies. Pels belonging to the last class are further investigated at the next higher resolution and checked for satisfaction of constraints thereby classifying them into three classes. This procedure continues until the maximum resolution is reached. It is trivial to see that, in this implementation, computation time is restricted to space in the diagram that requires it and not wasted by processing the entire diagram uniformly.[2] See Fig. 2 for example.

Computational complexity. Since the amount of space explored is problem-dependent, it is difficult to ascertain the absolute reduction in computational costs. However, an average case analysis will provide an idea of the efficiency

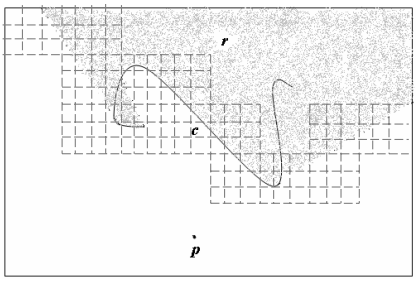
[2] A data-structure called *quadtree* is used to sample space hierarchically using pels of varying sizes based on the density of objects – higher the density, smaller the pel. Efficient spatial search requires a problem-dependent exploration of space as implemented by our strategy and not by quadtree.



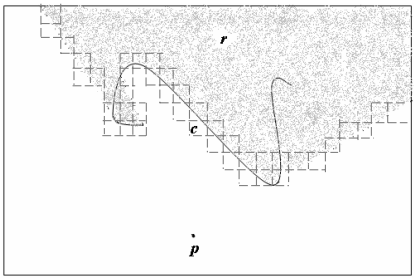
(a) An array is imposed on the diagram.



(b) The partially behind array elements.



(c) Subdividing partially behind array elements into smaller elements.



(d) The partially behind array elements at the higher resolution.

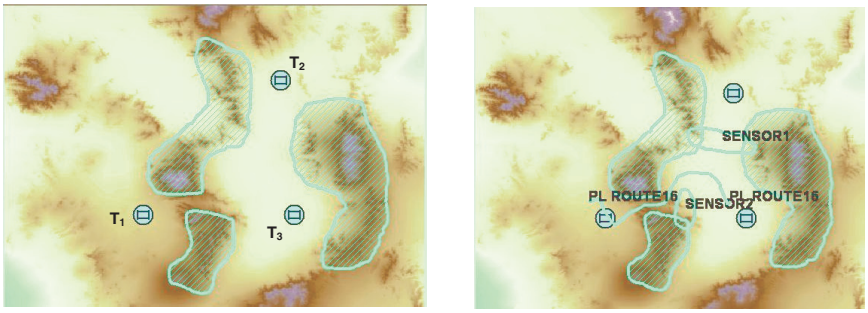
Fig. 2. Solving the $BehindCurve(p, c)$ problem by spatial search

achieved. Two kinds of computational costs are involved – a one time cost of filling in the multi-resolution data structure and a processing cost incurred due to checking every pel for satisfaction of the decision problem. The goal in this analysis is to count the total number of pels updated and processed by the naive spatial search strategy and compare it with the same for the resolution-based enhanced strategy. Let n be the number of pels at the lowest resolution, N be the number of pels at the highest resolution, m be the average fraction of pels that require further investigation at any step, and k be the total number of variables in the specification of the spatial problem.

Therefore, $N = nd^{2s}$. The multi-resolution data structure contains a total of $O(Nl)$ pels. In the naive spatial search, the SPS processes N^k pels. In the multi-resolution approach, at the i^{th} resolution, SPS processes $n^k(md^2)^{ik}$ pels. Thus, the total number of pels processed in s resolutions is $N^k m^{sk}$ – a reduction by m^{sk} $(m \leq 1, s, k \geq 1)$ compared to the naive approach. For example, in the *BehindCurve* problem (Fig. 2), using parameters $m \leftarrow 0.5$, $d \leftarrow 3$, $s \leftarrow 4$, $n \leftarrow 12 \times 12$, $N \leftarrow 1024 \times 1024$, $k \leftarrow 2$, processing time reduces by 99.61%.

4 Experimental Results

In this section, we illustrate how a human problem solver can exploit the SPS to execute perceptions and actions without human intervention as needed for



(a) Terrain, impassable regions, and sighted tanks.

(b) A path from the only plausible homotopy class.

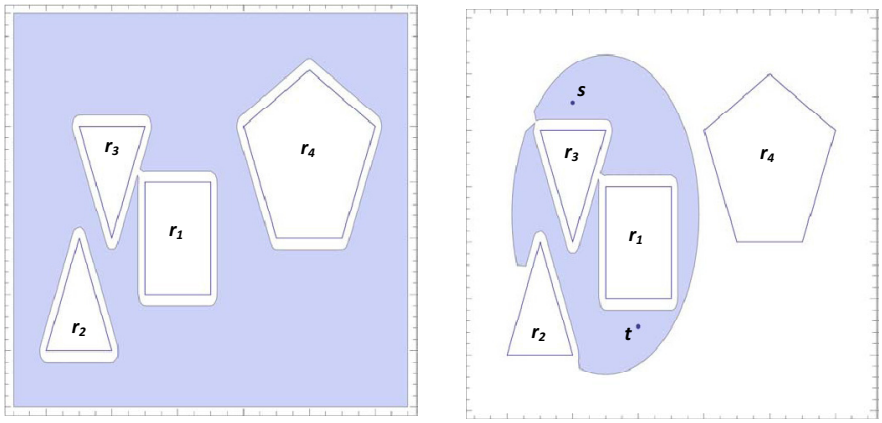
Fig. 3. A simplified entity re-identification scenario

DR. Two applications will be considered – entity re-identification and ambush analysis – that are deemed very important in the military domain. Problems in military domain involve a wide variety of objects with arbitrary properties and relations, and hence, help to illustrate the expressiveness of the specification language and the efficiency and generality of the spatial search strategy. Comparison of computation time between our SPS and the CAD algorithm [12] is provided for an instance of each spatial problem.

4.1 Entity Re-identification

The entity re-identification problem is a core task in the US Army's All-Source Analysis System (ASAS). ASAS receives a new report about sighting of an entity T_3 of type T (e.g. tanks). The task is to decide if the new sighting is the same as any of the entities in its database of earlier sightings, or an entirely new entity. Reasoning has to dynamically integrate information from different sources – database of sightings, mobility of vehicles, sensor reports, terrain and map information – to make the decision.

Fig. 3(a) shows the terrain of interest – mountainous with the closed regions marking impassable areas for entities of type T (e.g., tanks). Let T_3 be an entity newly sighted at time t_3 located at point p_3 while T_1, T_2 are the two entities that were located at points p_1, p_2 when last sighted at times t_1, t_2 respectively. T_1 and T_2 were retrieved from the database as having the potential to be T_3 based on their partial identity information. Also, in the area of interest, there are three enemy regions or obstacles $\{r_1, r_2, r_3\}$ with a given firepower/sight range d of the enemy. Reasoning proceeds as follows. If T_1 can reach p_3 within the time $t_3 - t_1$, then T_3 might be T_1. Similarly for T_2. Since each mountainous region (or obstacle) is a hiding place for enemies with a firepower range d, the existence of an entity shows that it most probably did not traverse through a territory within the firepower range. Further, there might be sensor fields that report to the database when they sense entities. If no entity was sensed between the times t_1 and t_3, then T_1 could not have followed a path that passed through



(a) The unshaded polygons are obstacles. The shaded region is the safe region.

(b) Paths lying in the safe region and less than a given length between two points.

Fig. 4. A simplified scenario to illustrate the performance of the proposed SPS

that sensor field. Such constraints have to be taken into account while reasoning. All information might not be available in the database at once. In what follows is a simple scenario and a discussion of the spatial problems as they occur.

The problem solver (e.g., a commander) wants to know whether there exists a contiguous safe region containing the points p_1 and p_3, and specifies:

$$SafeRegion(q, \{r_1, ...r_n\}, d) \equiv \forall a, \neg(\vee_{i\leftarrow 1}^{n} Inside(a, r_i)) \vee Distance(q, a) \geq d$$

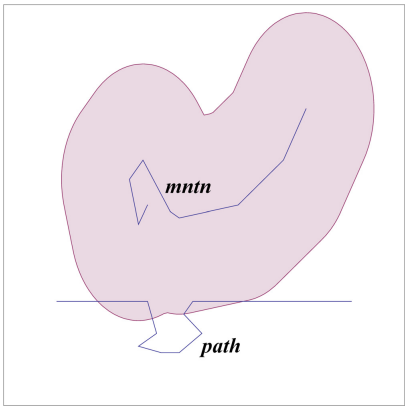
where $q \leftarrow (x, y)$. In order to compare the actual computation times, we constructed a very simple diagram consisting of four polygonal regions depicting obstacles (see Fig. 4(a)), where $r_1 \leftarrow \{(10, 10), (30, 10), (30, 30), (10, 30)\}$, $r_2 \leftarrow \{(-20, 0), (0, 0), (-10, 20)\}$, $r_3 \leftarrow \{(0, 20), (10, 40), (-10, 40)\}$, $r_4 \leftarrow \{(50, 20), (70, 20), (80, 40), (60, 50), (40, 40)\}$, and $d \leftarrow 2$. See Table 1.[3]

Next the problem solver wants to know whether there exists a path between points p_1 and p_3 safely avoiding the obstacles and enemy firepower range, and whether that path can be traversed in time $t_3 - t_1$. Let v be the velocity of the sighted entity – a piece of symbolic knowledge available from the database. Then, the maximum length of path traversable in the given time is $L = v \times (t_3 - t_1)$. Let $l \ll L$ be a rational number. Then, the problem of path existence between two points s and t such that the path lies inside a region r and is less than a given length l can be specified as:

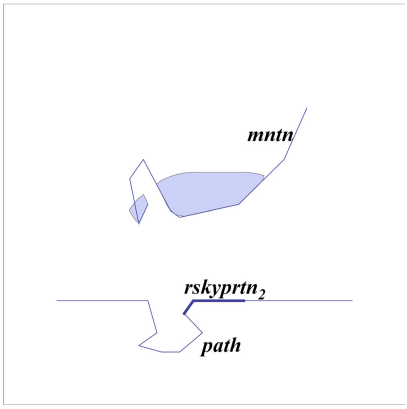
$$PathExists(s, t, r, l) \equiv \exists q, Inside(q, r) \wedge Distance(s, q) + Distance(q, t) \leq l$$

In Fig. 4, $s \leftarrow (0, 45)$, $t \leftarrow (20, 5)$, $r \leftarrow SafeRegion((x, y), \{r_1, r_2, r_3, r_4\}, 2)$, $l \leftarrow \sqrt{1010}$. The region consisting of all paths that satisfy the constraints is computed from the function problem

[3] SPS refers to objects by their labels and acquires their locations from the pels they occupy. But, for the CAD algorithm, numerical coordinates need to be provided.



(a) The shaded region is the risky region prone to ambush due to enemies hiding at *mntn*. The risky portions of *path* are those inside the risky region.



(b) Troops traveling on $rskyprtn_2$ (in bold) might be ambushed as they are within firepower range from enemies hiding in the shaded region.

Fig. 5. A simplified ambush analysis scenario to illustrate the performance of SPS

$$FindPath(q, s, t, r, l) \equiv Inside(q, r) \wedge Distance(s, q) + Distance(q, t) \leq l$$

where $q \leftarrow (x, y)$, and shown in Fig. 4(b).

From the results, the problem solver infers that T_3 might be T_1. Next he repeats the same for entities T_3 and T_2, and finds that T_3 might be T_2 as well. The sensor database informs that there are two sensor fields – `SENSOR1`, `SENSOR2` – in the area of interest but there has been no report from them of any passing vehicle. Problem solver wants to verify whether any of the paths passes through any of the sensor fields. He specifies the problem $IntersectRegions(r_1, r_2)$ to compute the intersection of two regions r_1, r_2 as:

$$IntersectRegions(r_1, r_2) \equiv \exists q, Inside(q, r_1) \wedge Inside(q, r_2)$$

He computes the problem $IntersectRegions(paths_{13}, s_1)$ where $paths_{13} \leftarrow FindPath(q, p_1, p_3, r, l)$ and s_1 is the region covered by `SENSOR1`. In our scenario, the solution is $True$. Next the problem solver wants to know whether there exists a path between points p_1 and p_3 safely avoiding the obstacles and enemy firepower range such that it can be traversed in time $t_3 - t_1$. He computes $PathExists(p_1, p_3, r_{13}, l)$, where $r_{13} \leftarrow paths_{13} - s_1$, which returns $True$. The inference follows that T_3 might be T_2. The same reasoning is repeated for T_3 and T_2; $Intersect(paths_{23}, s_2)$ returns $True$ while $PathExists(p_2, p_3, r_{23}, l)$ returns $False$. The inference follows that T_3 cannot be T_1. Hence, T_3 is T_2.

4.2 Ambush Analysis

There are two main factors – range of firepower and sight – that determine the area covered by a military unit. Presence of terrain features, such as mountains, limit these factors and allow units to hide from opponents. These hidden units

not only enjoy the advantage of concealing their resources and intentions from the opponents but can also attack the opponents catching them unawares if they are traveling along a path that is within the sight and firepower range of the hidden units, thereby ambushing them. Thus, it is of utmost importance for any military unit to determine the areas or portions of a path prone to ambush before traversing them. In this section, given a curve or region as a hiding place and the firepower and sight ranges, we show how the regions and portions of path prone to ambush is efficiently computed by the proposed SPS.

Given a curve c and the range d, the problem $RiskyRegion(q, c, d)$ is defined as the set of all points covered by that range from c, and specified as:

$$RiskyRegion(q, c, d) \equiv \exists a, On(a, c) \wedge Distance(a, q) \leq d$$

where $q \leftarrow (x, y)$. In order to compare the actual computation times required to solve the problem, we constructed a very simple diagram consisting of two curves, $path$ and $mntn$, where $path \leftarrow \{(-25, -10), (-5, -10), (-3, -15), (-7, -17), (-2, -18), (2, -18), (7, -15), (3, -12), (5, -10), (40, -10)\}, mntn \leftarrow \{(-5, 5), (-7, 2), (-9, 9), (-6, 12), (0, 4), (2, 3), (15, 5), (25, 12), (30, 20)\}$. The solution to $RiskyRegion(q, mntn, 15)$ is shown in Fig. 5(a) where $mntn$ is an obstacle for hiding (e.g., mountain range). $RiskyRegion(q, r, d)$ is specified by replacing $On(p, c)$ with $Inside(p, r)$.

Again, given a curve c_1 as a path, a curve c_2 for hiding, and a firepower range d, the problem $RiskyPortionsofPath(q, c_1, c_2, d)$ is defined as parts of c_1 covered by that range from c_2. Thus,

$$RiskyPortionsofPath(q, c_1, c_2, d)$$
$$\equiv On(q, c_1) \wedge \exists p, On(p, c_2) \wedge Distance(p, q) \leq d$$

where $q \leftarrow (x, y)$. Solution to $RiskyPortionsofPath(q, path, mntn, 15)$ is shown in Fig. 5(a). If the hiding place is a region r instead of the curve c_2, the problem can be specified by replacing $On(p, c_2)$ with $Inside(p, r)$.

The region behind c_2 where the enemies might be hiding is the set of all points that are behind c_2 with respect to each point on the risky portions of curve c_1. However, enemies might be hiding not anywhere behind a mountain but within a distance from where they can ambush the friendly units. An important problem is $BehindCurvewrtRiskyPath(q, c, c_2, d)$ where d is the distance from where the enemies can ambush them.

$$BehindCurvewrtRiskyPath(q, c, c_2, d)$$
$$\equiv \exists a, On(a, c) \wedge BehindCurve(q, c_2, a) \wedge Distance(a, q) \leq d$$

where $q \leftarrow (x, y)$. Solution to $BehindCurvewrtRiskyPath(q, rskyprtn_2, mntn, 20)$ is shown in Fig. 5(b). If the hiding place is a region r instead of c_2, the problem can be specified by replacing $On(p, c_2)$ with $Inside(p, r)$. A comparison between the CAD algorithm and our proposed SPS of actual computation times for problems relevant to entity re-identification and ambush analysis is in Table 1.

Table 1. Comparison of computation times (in seconds) between the CAD algorithm and our proposed SPS for instances of spatial problems discussed. A 2.8 GHz PC with 4 GB RAM, 5356 MB virtual memory, 32-bit operating system was used, and implemented in *Mathematica*. SPS maximum resolution was 128×128, comparable to CAD's piecewise linear approximation. Time required for object approximation and filling data structures were not included. Below, $q \leftarrow (x, y)$ and "OOM" refers to out of memory.

Spatial Problem	**SPS**	**CAD**
$SafeRegion(q, \{r_1, r_2, r_3, r_4\}, 2)$	3.0	5.5
$PathExists(s, t, r, \sqrt{1010})$	1.2	OOM
$IntersectRegions(paths_{13}, s_1)$	1.5	OOM
$CurveInsideRegion(c, r)$	5.7	175.01
$RiskyRegion(q, mntn, 15)$	0.5	0.11
$RiskyPortionsofPath(q, path, mntn, 15)$	0.1	0.48
$BehindCurve(q, mntn, (5, -10))$	0.6	0.71
$BehindCurvewrtRiskyPath(q, rskyprtn_2, mntn, 20)$	7.4	15.33

5 Conclusion

Our goal was to build a general framework for executing perceptions/actions such that a human can use them opportunistically for DR. These perceptions/actions were transformed into domain/task-independent $2D$ spatial problems and specified as QCSPs in the real domain. Traditional algebraic approaches for solving such QCSPs are inherently doubly exponential in time complexity. We proposed a general framework of spatial problem solving where a small vocabulary of predicates are implemented as procedures, using which a spatial problem is specified in first-order logic. Our SPS searches for the solution in the space of the diagram instead of in the space of algebraic equations/inequalities. We proved the correctness of our approach and showed it to be more efficient than CAD in executing perceptions/actions for two army applications.

References

1. Chandrasekaran, B., Kurup, U., Banerjee, B., Josephson, J.R., Winkler, R.: An architecture for problem solving with diagrams. In: Blackwell, A.F., Marriott, K., Shimojima, A. (eds.) Diagrams 2004. LNCS (LNAI), vol. 2980, pp. 151–165. Springer, Heidelberg (2004)
2. Pisan, Y.: A visual routines based model of graph understanding. In: Proc. 17th Annual Conf. Cognitive Science Society. Erlbaum, Pittsburgh (1995)
3. Tessler, S., Iwasaki, Y., Law, K.: Qualitative structural analysis using diagrammatic reasoning. In: Proc. 14th Intl. Joint Conf. AI, Montreal, pp. 885–893 (1995)
4. Lindsay, R.K.: Using diagrams to understand geometry. Computational Intelligence 14(2), 238–272 (1998)
5. Jamnik, M.: Mathematical Reasoning with Diagrams: From Intuition to Automation. CSLI Press/Stanford University, CA (2001)

6. Ferguson, R.W.: Magi: Analogy-based encoding using symmetry and regularity. In: Proc. 16th Annual Conf. Cognitive Science Society, Atlanta, GA, pp. 283–288 (1994)
7. Banerjee, B.: Spatial problem solving for diagrammatic reasoning. PhD thesis, Dept. of Computer Science & Engineering, The Ohio State University, Columbus (2007)
8. Egenhofer, M.J., Franzosa, R.D.: Point set topological relations. Intl. Journal of Geographical Information Systems 5, 161–174 (1991)
9. Frank, A.U.: Qualitative spatial reasoning about distances and directions in geographic space. Journal of Visual Languages and Computing 3, 343–371 (1992)
10. Cohn, A.G., Bennett, B., Gooday, J.M., Gotts, N.: RCC: A calculus for region based qualitative spatial reasoning. GeoInformatica 1, 275–316 (1997)
11. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proc. Intl. Symp. Symbolic and Algebraic Computation, pp. 54–60. ACM, New York (2007)
12. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. Journal of Symbolic Computation 12(3), 299–328 (1991)