Abstract

Title of Dissertation: Spatial Probabilistic Temporal Databases

Austin Parker, Doctor of Philosophy, 2008

Dissertation directed by: Professor VS Subrahmanian
Department of Computer Science

Research in spatio-temporal probabilistic reasoning examines algorithms for handling data such as cell phone triangulation, GPS systems, movement prediction software, and other inexact but useful data sources. In this thesis I describe a probabilistic model theory for such data. The **S**patial **Pro**babilistic **T**emporal database framework (or SPOT database framework) provides methods for interpreting, checking consistency, automatically revising, and querying such databases. This thesis examines two different semantics within the SPOT framework and presents polynomial-time consistency checking algorithms for both. It introduces several revision techniques for repairing inconsistent databases and compares them to the AGM Axioms for belief state revision; finding an algorithm that, by only changing the probability bounds in the SPOT atoms, can repair a SPOT database in polynomial time while still satisfying the AGM axioms. Also included is an investigation into optimistic and cautious versions of a selection query that returns all objects in a given region with at least

(or at most) a certain probability. For these queries, I introduce an indexing structure akin to the R-tree called a SPOT tree, and show experiments where indexing speeds up selection with both artificial and real-world data. I also introduce query preprocessing techniques that bound the sets of solutions with both circumscribing and inscribing regions, and discover these to also provide query time improvements in practice. By covering semantics, consistency checking, database revision, indexing, and query preprocessing techniques for SPOT database, this thesis provides a significant step towards a SPOT database framework that may be applied to the sorts of real-world problems in the impressive amount of semi-accurate spatio-temporal data available today.

Spatial Probabilistic Temporal Databases

by

Austin Parker

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Advisory Committee:

Professor VS Subrahmanian, Chairman/Advisor
Professor Dana Nau
Professor John Grant
Professor Ashoke Agrawala
Associate Professor Subramanian Raghavan

Table of Contents

List of Figures

Chapter 1

Introduction

GIS, GPS, cell tower triangulation, sensor networks, movement prediction software, traffic cameras and satellite photos are just some of the recently developed technologies used to generate data about objects' locations. This data is spatio-temporal – it gives a region and a time for the object – but it is also inexact [29, 47]. All the named systems suffer from error due to everything from the dynamics of radio signals to the inexact technologies used in image recognition. Therefore there is need for a framework handling probabilistic spatio-temporal data from such systems. The framework should be general enough to use input from several disparate, probabilistic data sources, and flexible enough to incorporate conflicting data with minimal changes. It should incorporate and deal appropriately with the error inherent in the data collection technology; in particular, it should not make supurfluous independence assumptions. It should be accessible in the sense that it should efficiently answer queries about objects' potential locations. The SPOT database system will be my answer to this problem.

Motivation

In this section, I motivate the development of the SPOT framework by detailing several potential applications. These examples should be seen as target applications, whose existence motivates the creation of a framework for correctly and explicitly handling probabilistic spatial temporal data. All these applications have the potential to benefit from a system designed for large quantities of probabilistic spatial temporal data, and the SPOT framework is designed to do exactly that.

The first such application is to cell phone tracking by mobile phone companies. Mobile phone companies must store phones' locations in order to scale the routing algorithms associated with cell phone use: it is much more efficient to start a search for a given phone with the cell tower most likely to be serving that phone [10]. By leveraging the data available, cell phone companies can increase efficiency, however all available data is spatial temporal and probabilistic: cell phone localization based on cell tower triangulation and movement prediction algorithms are not exact. The SPOT framework can store and query exactly this kind of probabilistic data, answering such questions as: "what objects can we expect in the range of a given cell tower with probability better than 0.75 at some given time." Further, in such settings there is likely to be a wide variety of information sources. The SPOT framework provides correct revision mechanisms allowing for data from multiple sources to be combined into one database *even when such data is mutually inconsistent*.

A potential military application for such technology involves handling the data from probabilistic models of enemy movement. There was a joint project between BBN, Lockheed Martin, the US Navy, and the University of Maryland where past sensor readings are used to create predictive models for enemy submarine movement. These models specify when, where, and with what probability one can expect an en-

emy submarine to be in a given region [34]. Another similar system using a diverse set of ground-based military sensors for specifying vehicle location at some time with an associated probability, is described in [28]. The data for that system also provides some probabilistic information on vehicle location at a given time. For military planners there is clear utility to properly representing the probabilistic spatio-temporal data acquired from such sources. These applications demand queries asking the probability of a vehicle being in a given region, nearest-neighbor-type queries, and other sorts of consistency and distance queries, all of which exist in the SPOT framework.

In imperfect information games, systems similar to the SPOT framework (though differing in fundamental ways) have already been employed. There is a game called *kriegspiel*, where one plays chess without knowledge of the opponent's moves. Because one knowns what moves are available and what the eventual locations of the opponent pieces will be for each move, one can store all possible piece locations in a knowledgebase known as a *metaposition* [9]. If one were to apply the SPOT framework in place of the metaposition, one would have a system for storing the same information, with added capabilities of representing the information in a probabilistic fashion as well as storing the data in a temporally-dependent fashion. While there is no guarantee that a SPOT -like representation would be more useful than metapositions in *kriegspiel*, such imperfect information planning and decision making environments provide a rich potential application domain for SPOT -like frameworks.

In GIS mapping technologies there are many sorts of error [26], some of which can be mitigated through use of probabilistic data representation. By allowing objects to be at locations with associated probabilities, and by allowing those objects to change location over time, one can construct more accurate maps. The SPOT framework provides this functionality as well as the ability to integrate new *prima*

facie-incompatible information into the database with minimal change to the existing data. One may have a map in the SPOT format stating that a lake is in a given location with 90% probability. However, upon arrival at that location one may discover no lake. This new knowledge is incompatible with our map and we must somehow change our current knowledge to incorporate this change. The revision functionality of the SPOT framework described in Chapter 5 shows how one can revise the current knowledge to reflect this newly discovered, incompatible fact.

While there have been several approaches to different aspects of these problems, the SPOT database system is the first to do spatio-temporal probabilistic reasoning without depending on probability density functions. There have been approaches dealing with temporal reasoning [20] and approaches dealing with spatial reasoning [15, 14]. There have been approaches dealing with probability in databases [17, 6, 13, 48] in logic [37, 35] and using Bayesian Networks [42]. The other works apart from those supporting this thesis that deal with space, and probability is of Tao et al. [45, 46] and Ni et al. [36]. Ni et al. use a data model where the error associated with each point's location is correlated with the error of many other points' locations. The system was designed for applications where certain spatial relationships, such as the relationship between corners of a building, are exactly know regardless of the actual locations of the points. Their system cannot handle the probability intervals or multiple region uncertainty that is used in the SPOT framework. Tao el al.'s work, while an interesting and effective approach for its addressed problem, requires access to a raw probability distributions. This is a pretty severe requirement for the sorts of applications we envision; for instance, one is not guaranteed to know the proper probability distribution for locational data extracted by means of, for instance, image recognition algorithms. These algorithm may have a tendency to consistently report

the location left or right of truth due to systematic things such as imperfections in the camera lens or non-systematic things such as the wind that day. Without specific knowledge of the underlying distribution, the SPOT framework is the only formally studied general approach one can take to represent these kinds of probabilistic spatial temporal information.

Outline

The SPOT system stores atomic facts of the form "*this object* is in *this region* at *this time* with a probability in *this interval*". For instance, one atomic fact might be: "Socrates is in the forum at sundown with a probability between 0.5 and 0.75". This says that with at least a 50% probability and at most a 75% probability, Socrates is in the forum at sundown. By collecting sets of such facts, one has a SPOT database. There are different ways to interpret the meanings of such databases, depending on whether or not one knows how objects are allowed to move (i.e. a maximum speed or path-finding software such as web-based direction giving programs). As such I will introduce two separate semantics for interpreting this data. These semantics are given in Chapter 3.

One essential algorithm needed for a complete SPOT system is the consistency checking procedure. It is possible for a set of atomic facts to force inconsistency by putting an object in two distinct regions with a probability greater than one. Inconsistent data is undesirable: one cannot rely on an inconsistent database. Fortunately, I was able to find efficient, polynomial-time consistency checking algorithms for all the semantics examined in this thesis. Chapter 4 introduces these algorithms and several related techniques to further decrease their computation time.

When the database is inconsistent, one might want it to automatically repair itself.

As such, I introduce in Chapter 5 several methods for revising an updated database. These methods can be run on update to guarantee the database's consistentency. These methods include traditional methods of considering consistent subdatabases [5, 24], as well as methods for changing each of the various aspects of a SPOT atom. These update methods are compared to the AGM axioms [1], a group of axioms that knowledge revision operations should satisfy. I find some of these procedures to satisfy the AGM axioms, while others to not. Of greatest interest is the probability-revision technique, which admits a polynomial time algorithm while satisfying the AGM axioms. This will allow a user of the SPOT system to be able to efficiently fix inconsistent databases.

Once one has a consistent collection of SPOT atoms, we can use it to answer questions about which objects will be where, and when. I develop several methods for efficiently answering these sorts of selection queries. As a canonical application, imagine such a query telling a cell phone company who they should expect to be in the region served by a given cell tower, as well as when they will be there. In Chapter 7 I examine several different pruning methods that can substantially increase the efficiency with which these queries will be answered. One method interfaces with a "probabilistic region" implicitly specified by a SPOT database. By storing both circumscribing and inscribed regions of the SPOT database's probabilistic region, we are able to quickly prune potential query answers without appealing to the database itself. Another method appeals to the intuitions behind the R-tree-like indexing data structures [7, 8, 44]. Called a SPOT tree, this index uses special composite atoms instead of minimum bounding rectangles to bound the sets of interpretations possible at a given level of the tree. By indexing in this way, we achieve speedup in experimentation with real-world data.

In summary, the contributions in this thesis are:

- The syntax for the SPOT framework and two different semantics (Chapter 3)
  - The point-based semantics for efficiently handling spatial probabilistic temporal data (Section 3.3).
  - The world-based semantics for correctly handling object movement constraints (Section 3.4).
- Polynomial-time consistency checking algorithms for both semantics (Chapter 4).
- An examination of potential revision techniques for repairing inconsistent databases (Chapter 5).
  - Proof that AGM-compliant revision techniques do not exist for spatial revision (Section 5.6).
  - A polynomial-time AGM-compliant probabilistic revision technique (Section 5.8).
- A basic algebra and set of queries for SPOT databases (Chapter 6).
- Several algorithms and data structures for speeding up selection queries (Chapter 7).
  - SPOT -trees, an R-tree variant for the SPOT framework (Section 7.3).
  - A data structure using enclosed regions (rather than bounding regions) for pruning (Section 7.4).

The SPOT database framework represents a new method for handling probabilistic spatio-temporal information. This thesis introduces the framework and addresses some of the major algorithms necessary for its successful deployment.

Chapter 2

Related Work

2.1 Probabilistic Spatial Temporal Databases

While there is much work on probabilistic, spatial, and temporal databases individually, there is little work covering all these areas. The works that do exist differ from the SPOT framework in that their representations, while effective for the specific problems they're addressing, lack certain kinds of generality.

The most significant related work is that of Tao et al. in [45, 46]. While both of these works deal with query answering schemes over probabilistic spatial temporal databases, they differ in that they assume access to the underlying probability distribution function. Indeed, these works provide an interesting, coherent, and efficient method for storing *one* probability distribution function (PDF) so as to avoid the computational costs of integration required by the query operations they consider. In contrast, SPOT framework queries never integrate. The work of Tao is fundamentally distinct from work on SPOT databases: while we can handle exactly one PDF as input, our framework is geared towards situations where the actual underlying PDF is unknown and one must reason with data limiting the set of possible PDFs. As such,

the work of Tao et al. is related but not comparable to SPOT databases.

In [45], the authors introduce the concept of an "uncertain object" o, which has a PDF: $o.pdf$ and an "uncertainty region" $o.ur$. The fundamental query of interest is a prob-range query, where given a region r_q and a threshold p_q, one must return all uncertain objects o such that

$$\int_{o.ur \cap r_q} o.pdf(x)dx \geq p_q.$$

For arbitrary PDFs, these are expensive operations due particularly to the fact that the space covered by $o.ur \cap r_q$ is not limited to some granular grid, but is instead a continuous subregion of space. The authors of [45] assert that when the PDFs are not known to be of a certain type (i.e. uniform, gaussian, etc), the best method for computing these integrals is a *Monte Carlo* computation. In such a computation, one draws a large number of points from $o.ur$ according to a uniform distribution and relates the probabilities of those points in r_q to the total probability sampled. With this method, the authors are able to achieve query answers for each object with about 0.1% error in 130 milliseconds with 10^8 samples. While this is fairly efficient for one object, when dealing with a sizeable dataset such numbers are unacceptable.

As an improvement upon the Monte Carlo computation, the authors introduce several structures leading up to the U-tree, an R-tree-esque indexing structure for individual spatial PDFs. The first of these structures is the probabilistically constrained region (PCR). A PCR for an object o is parameterized by a probability $p \in [0, 0.5]$. $o.pcr(p)$ is a set of $2d$ hyperplanes, two for each dimension. The hyperplanes divide space such that exactly p of the probability mass of o lies on one side of the line. There are two for each dimension so that there can be one hyperplane l_i^+ such that the probability p is to the "right" of the hyperplane, and another l_i^- such that the probability p is to the "left" of the hyperplane. The PCRs aid query computation by

providing pruning: if the query threshold is p_q, and the query region is on the wrong side of a PCR's hyperplane for some $o.pcr(p')$ (where $p' < p_q$), then we may prune object o without any integration. By pre-computing and storing PCRs for each object o and applying them in this way, one can hope to decrease the computation necessary for many probabilistic range queries, though there will always be many cases where integration is necessary.

A potential issue with the PCR, which the authors bring up and address, is that the PCRs represent a large amount of additional storage space. Therefore, the authors introduce the conservative functional box (CFB), a structure that linearly approximates the $o.pcr(p)$ (where $o.pcr(p)$ is considered to be a function with input p). The approximations are generated such that there are two conservative function boxes, one to bound the PCRs from the "inside" and one to bound the PCRs from the "outside". Thus the CFBs can still be used for pruning in ways similar to the PCRs. Further, since the CFBs are linear approximations, they can be represented with only a coefficient and a y-intercept, using much less storage space than the PCRs. The CFBs can then be placed into a U-tree, which is like an R-tree except the minimum bounding rectangles bound the CFBs (in this case, they might more appropriately be called minimum bounding functions). The authors provide a methodology for constructing such MBRs from CFBs, and can then apply the standard R-tree insertion and query methodology wholesale.

In [46], a subset of the authors of [45] examine a simplified version of the above techniques. They remove the CFBs from the U-tree and simply using PCRs as the region indexed by in an R-tree structure also called a U-tree. Again, standard R-tree procedures are applied with sensitivity to the new nature of the MBRs. In both papers, the experimental evaluation shows these techniques to provide efficient running times

for the associated probabilistic queries.

Another work on probabilistic spatial databases is that of Ni, Ravishankar and Bhanu [36]. They use a probabilistic spatial database framework with a complicated data model designed to handle correlation between data error. In their model, each spatial point is part of a "chunk", and the probabilistic error is associated with entire chunks. That is, if one point in a given chunk is actually two meters east from the location specified in the database, then all points in that chunk are actually two meters east from their specified locations. Such a data model requires careful mathematical definitions for range, distance, and join operations, and an even more carefully constructed index structure. The index structure they develop, called a PrR-tree, addresses the region that a point *may* occur in rather than the simple stored point. Also based on the R-tree, the PrR-tree uses a probability distribution to describe each node's minimum bounding rectangles. The author's PrR-tree algorithms can correctly handle their target operations. This contrasts with this thesis in representation and approach: the SPOT model theory allows for the representation of many separate probability distributions and handles many different sorts of correlation.

Work by Dai et al. [16] deals with a different kind of probabilistic spatial databases: instead of assigning a probability to an object's location, they assign a probability to an object's *existence*. They call this *existential uncertainty*. Consider a sensor with a high rate of false positives. Such a sensor reports that an object is at location (x, y) and due to the potential for false positives one cannot assume an object exists at that point. There is no probability distribution over the object's potential locations, just over the object's potential to exist. These authors use R-trees to store the spatial locations and base their algorithms on the nearest neighbor algorithms from the R-tree literature. The general approach of their algorithms is to extract the neighbors of a location from

an R-tree, and then to augment the R-tree results with the associated probabilities of existence before returning, reordering the R-tree results when appropriate.

Of the works that may be called probabilistic spatial temporal, the SPOT framework is the most general, being the only one capable of representing large classes of probability distributions.

2.2 Work on Probabilistic Databases

There are several approaches to representing probabilistic data in the database literature. In this short survey, I first examine Cavallo and Pittarelli's [13] canonical approach of specifying mutually exclusive facts in one database table. We then look at an approach by Zhao, Dekhtyar and Goldsmith [48], which also maintains mutually exclusive facts but uses probability intervals and a more complex query language. From there, we look at the attribute-based probabilities present in the databases of Barbara et al. [6] and the system ProbView [31], which details an implementation combining some of the best techniques in the literature as well as some new techniques.

2.2.1 Table as the Distribution

Cavallo and Pittarelli present one of the first database approaches integrating probabilities. In their approach, a probability distribution is assumed over the tuples in a single relation. Each tuple is given a probability and it is required that in a single relation, the tuples' probabilities add to 1. For instance, there may be the relations R_1

and R_2 (example is from [13]):

$$R_1 = \begin{array}{ccc} v_1 & v_2 & p(\cdot) \\ 0 & 0 & 0.3 \\ 0 & 1 & 0.3 \\ 1 & 0 & 0.3 \\ 1 & 1 & 0.1 \end{array}, R_2 = \begin{array}{ccc} v_2 & v_3 & p(\cdot) \\ 0 & 0 & 0.2 \\ 0 & 1 & 0.4 \\ 1 & 0 & 0.2 \\ 1 & 1 & 0.2 \end{array}.$$

On the operations of join and projection, the probabilities are updated. For projection, this is straightforward: a projected tuple's probability is the sum of the probabilities of the tuples it is derived from. For instance:

$$\Pi_{v_1}(R_1) = \begin{array}{cc} v_1 & p(\cdot) \\ 0 & 0.6 \\ 1 & 0.4 \end{array}.$$

For join, this is less straightforward. First we need to consider the *entropy* of a given probabilistic relation. This is defined as is normal: $H(q) = -\sum_{t \in T} q(t) \log(t)$ (q is a probability distribution over T, where T is a given relation). The *information content* of a probabilistic relation is defined to be the difference between the uniform distribution's entropy ($H(u)$) and the relation's entropy. The authors make substantial use of entropy and information content to define such things as functional dependencies, project-joins, and multivalued dependencies in the probabilistic database context. For probabilistic join, the process is simple: join occurs as normal – the tuples are combined according to the conditions. Then the probabilities are assigned in such a way that when one projects back from the join relation to the base relation, the probabilities are the same as originally in the base relation. Since there are in general many ways of assigning probabilities to satisfy this condition, the authors avoid ambiguity by requiring the assignment with maximal entropy is used. This paper is an interest-

ing and complete first look at how one might approach probabilistic databases, and surprisingly, one of the few papers to appeal to entropy in the probabilistic database literature.

2.2.2 Probability Intervals in Probabilistic DBs

Zhao, Dekhtyar and Goldsmith have produced a methodology by which probability intervals can be maintained for facts in databases [48]. Their formalism clearly intends to capture the sort of data one acquires through exit polls, surveys, and sampling techniques whereby percentages can be calculated to be within a given confidence interval. They start their formalization with a simple semistructured probabilistic object (SPO), which contains[1]:

- Data specifying the *context* of the distribution.
- A set of random variables V with possible values $dom(V)$.
- A probability distribution over $dom(V)$.

For instance, one may take an exit poll outside a particular polling station asking if the voters voted for candidate A or candidate B. In this case, the context would state the polling station where the survey occurs and the set of random variables would simply be the candidate the voter voted for (a singleton). $dom(V)$ then just contains the two candidates: $\{A, B\}$, and the probability distribution tells what percentage voted for each.

To incorporate probability intervals, the authors introduce extended semistructured probabilistic objects (ESPOs). These objects contain extra contextual informa-

[1]Paraphrased for presentation purposes.

tion, as well as probability *bounds* for the attributes instead of the bare probabilities stored in SPOs.

Altogether, the contexts and the ability to store probability bounds combine to form a very powerful logic and query language capable of asking exactly the kinds of statistical queries that are both useful and mathematically possible. For instance, there are projection queries that ask what the probabilities are for one random variable independent of the value of the others. Here the probability interval is of clear utility when one does not know if a conditional dependence exists. The authors also introduce a new query: the conditionalization query. Here one asks the what the probability distribution is for one variable when certain random variables have particular values. These naturally return ESPOs with the query's condition as a part of the ESPOs' context, and the (new) lower and upper probability distributions. Overall, the framework provided is coherent and consistent and of clear utility to those wanting to represent statistical information.

2.2.3 Probabilities in Attributes

Barbara et al. [6] examine a method for handling probabilistic data by associating probabilities with the values of attributes, including partially specified probability distributions. One might say their framework introduces the probability distribution as a new attribute type to be considered on par with traditional types such as "integer" and "string". For example, a given schema may be used in a business application for attempting to determine bonuses (example derived from [6]). In this example, coworkers of John Smith were asked to estimate what they thought the size of John's bonus should be as well as what they thought his contribution to sales had been that year.

Employee	Bonus Estimate	Sales Estimate
	0.3 [Great]	0.4 [$30-$34K]
John Smith	0.4 [Ok]	0.5 [$35-$39K]
	0.3 [*]	0.1 [*]

This representation is more compact than previous probabilistic database approaches requiring one probability per tuple. The above table would require 9 tuples to represent in Cavallo and Pittarelli's framework. However, it is also more difficult to represent conditioned probabilities: the distributions across tuples are generally assumed to be independent.

The authors produce a probabilistic relational algebra based on this framework. They include the standards of union, intersection, projection, selection, join, etc, as well as some new operators like ϵ-select and ϵ-join. The ϵ operators differ from their non-ϵ counterparts in that they do not require an exact match to their condition, but instead require only that the distance according to some metric is below the threshold ϵ. For instance, one might want to select from the above table all employees with a bonus estimate of "0.33 [Great], 0.42 [Ok], 0.25 [*]". This distribution is close to the example one, but because it is not exactly the same, the tuple in the above database would not match. In an ϵ-select operation, the distance between the two distributions determines if the tuple is returned.

The work Barbara et al. is a canonical example of an attribute probabilities approach to probabilistic databases, where the probability distribution is stored at the attribute level. This is to be compared with storing the probabilities at the tuple level, as in the previously described works and the SPOT database framework.

2.2.4 ProbView

In [31], Lakshmanan et al. detail their probabilistic database system "ProbView". ProbView is unique in that it uses both attribute and tuple probabilities, one for representation and one for presentation.

The basic probabilistic tuple for relation scheme $R = \{A_1, \cdots, A_n\}$ is defined to be $(\langle V_1, h_1 \rangle, \cdots, \langle V_n, h_n \rangle)$ where $V_i \subseteq A_i$ and h_i is a function from V_i to probability intervals (all $a \in A_i, a \notin V_i$ are assigned probability 0). These probabilistic tuples are expressed in the framework as annotated probabilistic tuples, which are tuples from the base relation with an associated probability interval and some extra data justifying the annotation. They define many operations such as selection, join, etc, and also introduce the new operation of "compaction". Compaction is a probabilistic version of duplicate elimination, where the probability bounds of data-identical tuples are combined as is appropriate.

Central to the techniques in this paper is the concept of a "path". Paths are stored for each annotated tuple as part of the data justifying the annotation. A path is a boolean formula expressing what must be true for a given tuple to be possible. For instance, suppose we have tuples a_1 and a_2 with probability intervals $[\ell_1, u_1], [\ell_2, u_2]$. Now consider the concatenated tuple $a_c = a_1 \cdot a_2$. Since a_1 and a_2 only occur with some probability, a_c is only possible when both a_1 and a_2 occur. Let w_1 be true when a_1 occurs and w_2 be true when a_2 occurs (w_1 and w_2 are the paths for a_1 and a_2), then a_c can only occur if $a_1 \wedge a_2$. Thus $a_1 \wedge a_2$ will be the *path* for a_c.

In their implementation, data is communicated to users via probabilistic tuples while stored internally as annotated tuples. They present a host of experiments showing adequate performance by their implementation.

2.2.5 Efficient Probabilistic Query Evaluation

In [17], Dalvi and Suciu present an interesting and effective way of determining when it is possible to do extensional queries. An extensional system tags each bit of data with a probability and proceeds to do queries that are algebraic combinations of those probabilities, while an intensional system defines a probability distribution on a set of worlds and computes queries in that realm. Extensional semantics have difficulties dealing with correlated facts – queries depending on such correlation are generally incorrect when computed extensionally. However, when extensional semantics can be used, they are substantially more efficient than intensional systems, which generally must consider each of the many of possible worlds individually.

Dalvi and Suciu show how to create "safe" extensional query plans when such plans exist. That is, they give an algorithm that takes a query plan and rewrites it so that all operations can be performed extensionally. Their technique relies on finding dependent variables in the database and ensuring that the extensional operations are correctly distributed over the dependencies. However in some cases, an extensional query plan does not exit. They provide an optimizer that finds extensional query plans in all cases where such plans exist, and they prove their optimizer to run in polynomial time. Finally, they give some hints as to how one might answer queries that cannot be computed extensionally.

2.2.6 Probabilistic Temporal Databases

Dekhtyar, Ross, and Subrahmanian introduce a formalism for handling probabilistic temporal databases in [19]. A TP-case statement is one basic construct in this formalism, consisting of: $\{(C_1, D_1, L_1, U_1, \delta_1), \cdots, (C_n, D_n, L_n, U_n, \delta_n)\}$, where each C_1 and D_i are time intervals, L_i, U_i are probabilities, and δ_i is a pdf over D_i. δ_i is only

considered over the time points specified in C_i: every point $x \in D_i, x \notin C_i$ is given a probability of zero (after normalization). A tuple associated with a TP-case statement is true at a given time with a probability derivable from the data in a TP-case statement. A TP-tuple provides this association, containing a standard database tuple and a TP-case statement specifying when and with what probability the tuple holds.

For theoretical analysis, TP-tuples are divided into *annotated relations*, assigning each time point its own probability interval. Formally, if tp is a TP-tuple, then $ANN(tp) = \{(d, t, L_t, U_t) | t \in C_i, d = d_i\}$ where d_i is the database tuple in the TP-tuple, t is a time point, and L_t and U_t are the appropriate probability bounds for that time point. The semantics of the system are defined in terms of annotated relations. An interpretation specifies a probability for each tuple at each time point, and is consistent with an annotated relation only if the interpretation's specified probability is within the annotated relation's probability interval.

Definitions for union, difference, intersection, Cartesian product, selection, projection and join are given in terms of these annotated relations. Together with a mapping from TP-tuples to annotated relations, these definitions define corresponding operations for TP-tuples. Correctness of algorithms for those operations on TP-tuples is then proven in terms of the semantics for annotated relations. The paper provides correct algorithms for the TP-tuple operations that *do not* require the instantiation of annotated relations while at the same time being proven correct through appeal to annotated relations..

In a followup paper, these authors examine the query processing via a temporal probabilistic calculus (TP-calculus) [18]. Operators such as join, selection and projection are defined and rules for query rewriting based on the algebraic properties of those operators are examined. The authors experiment with their queries and query

rewriting system with a prototype implementation.

2.3 Probabilistic Logics

The AI Logic community has also seen substantial work on probabilistic reasoning. I will discuss a few of such works here.

In *A Logic for Reasoning about Probabilities* [23], Fagin, Halpern and Megiddo develop a logic that works over arbitrary (not necessarily discrete) probability distributions. The basic formulation allows primitive weight terms $w(\phi_i)$ where ϕ_i is a propositional formula. An inequality term is an inequality:

$$a_1 x_1 + \cdots + a_k x_k \geq c$$

An inequality formula is a boolean combination of inequality terms. The authors prove soundness and completeness of an axiomatization for logics based on such inequality formula, and show satisfiability to be NP-complete.

In another paper, Ng and Subrahmanian provide one of the first treatments of probabilities in logic programming [35]. Let B_L be the Herbrand base associated with a first order logic and let *basic formula* be any conjunction and disjunction over B_L. The basis of their probabilistic logic is a Horn clause equivalent, the p-clause, with the form:

$$A : [\ell, u] \leftarrow F_1 : [\ell_1, u_1] \wedge \cdots \wedge F_n : [\ell_n, u_n]$$

where $F_1, \ldots, F_n$ are basic formula. A p-program is a set of p-clauses. This structure is an instance of an *annotated* approach, due to the fact that the probability intervals annotate the associated formula.

To define the semantics of such programs, Ng and Subrahmanian use an atomic function f that assigns a probability interval to each member of B_L. Then, to extract

probability intervals for the basic formula, conjunction or disjunction strategies are used (as appropriate). In this way, we can tell if f satisfies each basic formula *and its associated probability interval* $F_i : [\ell_i, u_i]$ in the p-caluse. As is standard, to satisfy a p-clause f must satisfy the head or not satisfy at least one element of the body. The authors define algorithms for computing satisfaction, consistency, and other operations relating to this probabilistic logic.

In a latter, and somewhat more general paper, Lakshmanan and Shiri propose a probabilistic logic that works on certainty lattices satisfying certain properties (i.e. probabilities, probability intervals, some fuzzy logics, binary logics, etc). Using a first order logic, they allow rules of the form

$$r : A \overset{\alpha}{\leftarrow} B_1, \cdots, B_n; \langle f_d, f_p, f_c \rangle.$$

Where $A, B_1, \cdots, B_n$ are (ground or non-ground) atoms, and f_d, f_q, f_c are functions describing disjunction, propagation, and conjunction strategies respectively in the given certainty lattice. An example rule, using probability intervals as the certainty lattice, might say that an area is "affected" (by a disease) if there is an outbreak (of the disease) with a probability in the interval $[0.8, 0.9]$:

$$r_a : \textit{affected}(D) \overset{[0.8,0.9]}{\longleftarrow} outbreak(D); \langle \times, \times, \times \rangle$$

Where $\times$ is the function: $[a, b] \times [c, d] = [a \times c, b \times d]$ (this assumes all variables involved are conditionally independent). The authors fully develop several equivalent semantics (declarative, fixed point, proof theoretic) and study conjunctive queries for such statements, finding several classes of conjunctive queries with differing properties. These results will be useful for further work optimizing such queries.

2.4 Repair, Revision, and Update

2.4.1 AGM Contraction and Revision

In a canonical paper, Alchourrón, Gärdenfors and Makinson flush out what have been come to be known as the AGM axioms. These axioms pertain to theory change [1]. A theory is a set of propositions closed under the consequence operation, which is to say that should proposition p in theory A imply a proposition q, then q must also be in A. $Cn(A)$ denotes the closure of an arbitrary set of propositions A under the consequence operation.

A *contraction* operator ($A \dot{-} p$) removes some proposition p from theory A. The authors introduce the following postulates for contraction.

- $A \dot{-} x$ is a theory whenever A is a theory (closure).
- $A \dot{-} x \subseteq A$ (inclusion).
- If $x \notin Cn(A)$, then $A \dot{+} x = A$ (vacuity).
- If $x \notin Cn(\emptyset)$, then $x \notin Cn(A \dot{-} x)$ (success).
- $A \subseteq Cn((A \dot{-} x) \cup \{x\})$ (recovery).

As an example of contraction, the authors introduce full meet contraction, First they define $A \perp x$ as the set of subsets of A such that for $A' \in A \perp x$, (i) $A' \subseteq A$, (ii) No $A'' \subsetneq A'$ is in $A \perp x$, and (iii) $x \notin Cn(A')$. *Full meet contraction* is then the intersection of all members of $\cap A \perp x$. *Partial meet contraction* is defined as $\cap\gamma(A \perp x)$ where the operator γ satisfies $\gamma(A \perp x) \subseteq A \perp x$. Partial meet contraction uses only the "most important" members of $A \perp x$ (according to γ) to accomplish the contraction. The authors show full meet contraction to satisfy the revision axioms

above, then go further to show that *any* operator $\dot{-}$ satisfying the above axioms is a partial meet contraction.

Further axioms are introduced for *revision* of a theory, whereby a proposition p is added to a theory, and if there is a contradiction, the theory is modified to preserve consistency. These are:

- $A \dot{+} x$ is always a theory.
- $x \in A \dot{+} x$.
- If $\neg x \notin Cn(A)$ then $A \dot{+} x = Cn(A \cup \{x\})$.
- If $\neg x \notin Cn(\emptyset)$ then $A \dot{+} x$ is consistent under Cn.
- If $Cn(x) = Cn(y)$ then $A \dot{+} x = A \dot{+} y$.
- $(A \dot{+} x) \cap A = A \dot{-} \neg x$.

It turns out that for any valid contraction operation $\dot{-}$ (such as partial meet contraction, above), the Levi identity implies a revision operation: $A \dot{+} x = Cn((A \dot{-} \neg x) \cup \{x\})$. Because all contraction operations have a partial meet instantiation, this means that all revision operations can be traced back to a partial meet contraction operation.

The authors further discuss many supplementary postulates for contraction and revision, and show large classes of such operations to be equivalent.

2.4.2 Inconsistent Database Repair and Querying

There is substantial work on repair in inconsistent databases. Much of it differs in philosophy from the work presented in this thesis by focusing on answering queries with inconsistent databases rather than by repairing or revising an inconsistent database on update.

In [3], Arenas Bertossi and Chomicki produce a method for rewriting queries such that the resulting rewritten query returns answers even when the queried database does not satisfy integrity constraints. The integrity constraints that may be violated are part of the rewritten query. The paper offers proof of the technique's correctness.

The work in [4] introduces a logic programming technique for computing consistent query answers (i.e. answers that are true for every repair of the database). The technique is only given for binary integrity constraints, but may be extended.

In [25] Fazzinga *et al* examine how to use mixed integer linear programming to repair inconsistent databases. They allow constraint specifications that include SQL statements and produce mixed integer linear programs from those specifications. By allowing individual entries in the database to be variables in the linear program, the authors allow for solutions to the linear program to be repairs to inconsistent databases. Using clever optimization techniques, the authors are able to solve the system of constraints with a minimal number of changed variables. Thus they are able to repair the database by changing a minimal number database entries. Since they deal with mixed integer linear programming, their algorithms are NP-hard. In particular, the technique used to minimize the number of changed variables is only available in mixed integer linear programming and is thus also NP-hard.

2.4.3 Combining Multiple Knowledgebases

In [5], Baral et al detail a method for combining knowledge bases even when the combination violates certain integrity constraints. Their procedure uses a heuristic whereby if $A \leftarrow B$ is an integrity constraint, then repairs that include A are preferable to repairs that include $\neg B$. They define certain axioms to be followed when combining multiple knowledgebases. These are similar to the AGM axioms, but in-

clude such things as commutativity (K_1 combined with K_2 should result in the same knowledgebase as K_2 combined with K_1). The authors study several algorithms for various special cases of knowledge base combination, including combining theories consisting only of facts, combining theories with rules without negation in the bodies, syntactic approaches, and incremental theory combination. All algorithms use a maximal subset approach to address inconsistencies arising from the combination.

2.5 Spatial Databases

While there are very few works dealing with space, time, and probability, there are many works dealing with space individually. In this section I summarize a few such works.

2.5.1 Spatial Reasoning

Cohn and Hazarika survey many forms of qualitative spatial representation and reasoning in [15]. These include pointless geometries such as the RCC system [14], in which the basic object is a region and there are predicates specifying the sort of connectivity ("part of", "disconnected from", "connected to", etc).

In [33], Malek describes a way of using RCC in a GIS environment. He uses a four dimensional space to represent space time, and then, for a given event, he draws a cone shape containing the locations (and times) that a given object can *possibly* affect. He describes these regions with an RCC-style logical calculus.

In [12], Cao et al. study the application of a computer graphics line simplification algorithm to spatial temporal databases. The line simplification algorithm is shown to introduce bounded error in the representation of objects' locations over time. The

algorithm can be used to create more compact temporal databases which store simplified lines rather than per-time-point data.

2.5.2 Indexing Spatial Databases

In spatial databases there are two major indexing structures, R-trees and Quad-trees.

R-trees have many variants [7, 8, 44], though the basic concept is a tree structure with spatial data in the leaf nodes. Internal nodes contain minimal bounding rectangles (MBRs) and generally satisfy the constraint that all data beneath an internal node must be contained in the internal node's MBR. The many variants to the R-tree each have their own heuristic method for deciding how to insert new data in order to minimize some objective (number of I/Os or runtime). When one wants to discover the data existing in a particular spatial region, one need only search from the root visiting only those children whose MBR intersects the query region. The issue with this is that while it accomplishes some pruning, there is no guarantee that the spatial data will be found down any given path, and thus the performance bounds for R-trees are no better than when R-trees are not used. Despite these theoretical difficulties, R-trees perform quite well in practice.

Quad-trees, however, do have theoretical bounds on their running time [43]. At the most general level, a quad-tree node divides the space it represents into regions, and stores a pointer to another node responsible for each region. In some quadtrees, such as point quadtrees, the internal nodes store a data item and use that item to divide the space they represent. Other quadtrees simply divide space in half and store the data at the leaves. Still other structures, known as adaptive k-d trees, adaptively decide where best to divide space based on the input they represent.

Chapter 3

SPOT Databases: Syntax and Semantics

3.1 SPOT Semantics

In this chapter, I introduce two different semantics for the SPOT framework. The different semantics have differing advantages, allowing for us to make a tradeoff between the database's efficiency and the information implied by the database. The first semantics achieves faster execution time because it does not consider the possibility of there being movement constraints, instead assuming that interactions between time points are handled by the data source. The second semantics will be less efficient (but will still admit polynomial-time algorithms) and will ensure reachability constraints are maintained in consistent databases.

Both semantics rely on the use of probability intervals. The inclusion of probability intervals in the SPOT framework allows for the data producer to more delicately handle potential correlations between objects. As an example, consider two facts reported by the same sensor. Independently, each fact is true with a 80% probability, however, if one fact is false, the sensor is likely faulty and the second fact is only true with a 60% probability. Thus the appropriate probability to assign to both facts is

the interval of 60% to 80%. I introduce probability intervals to handle exactly these sorts of representational issues, following in the tradition established by Boole [11] and continued in modern computer science [27, 23, 35, 48, 21].

The first semantics is used in several papers of mine [41, 38]. It allows a distinct probability distribution for each object and time point, and as such is called time point semantics. Because this semantics allows no particular correlation between these probability distributions, it cannot account for an object's potential movement. Databases using these semantics will instead be expected to encode any possible movement constraints in the data. In most cases the algorithms for these semantics outperform any semantics taking potential movement into account.

The second semantics was developed to explicitly enforce object movement constraints and is used in two other papers of mine [41, 39]. It only allows objects to move between "reachable" locations by assigning probabilities only to "worlds" that satisfy specified reachability constraints. I therefore call this the world-based semantics. The world-based semantics generally requires more computation than the time-point semantics, but will still admit polynomially bounded running times.

While the two semantics are technically quite different, conceptually they can be thought of similarly. Many of the techniques in this thesis will be introduced only in one of the two semantics, and have its details flushed out with those particular assumptions. I would like to caution the reader against concluding that the technique is thus only applicable to the considered semantics: in many cases the fundamental idea of a given technique is also applicable to the other semantics, due at least partially to the fact that both semantics use the same syntax and similar probabilistic model theories.

I now introduce the syntax for the SPOT framework.

3.2 Syntax

3.2.1 Preliminary Definitions

SPOT databases are composed of SPOT atoms. In this section we will detail the basic definitions used in this paper.

Suppose ID is a set of object identifiers. Unless otherwise specified, we assume the set ID to be finite. Further suppose a *distance function* $d_{id} : ID \times ID \to \mathbb{R}$ on the set ID. The distance function specifies how similar two ids are, and is provided by the user.

Example 1. *Suppose the set ID contains car license plate numbers. Then $d_{id}(id_1, id_2)$ can be the edit distance between the license plate numbers id_1 and id_2.*

Time points are represented via a finite subset of the integers T, as is standard. There are integers T^- and T^+ such that $T = \{i \in \mathbb{Z} \,|\, T^- \leq i \leq T^+\}$. This assumption of discrete finite time is one commonly made in database and logical systems [19, 36, 20]. It is important for the database designer to ensure the granularity of time is fine enough to account for the phenomenon of interest (i.e. for studying plate tectonics, one might allow each time point to represent 1000 years, while for studying vehicle movement, a time point representing 10 minutes might have better utility).

Space will be represented by a finite set of locations $\mathcal{L}$. Each location $L \in \mathcal{L}$ is a point in $\mathbb{R}^2$: $L = (x, y) \in \mathbb{R}^2$. The distance between points $L_1, L_2 \in \mathcal{L}$ will be the Euclidean distance:

$$ed(L_1 = (x_1, y_1), L_2 = (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

As I proceed, you will notice that the locations can just as easily be points in a higher

dimensional space $\mathbb{R}^k$ – they are specified as members of $\mathbb{R}^2$ simply for ease of presentation.

Subsets of $\mathcal{L}$ will be called *regions*. A *rectangular region* r is defined by lower and upper corners (x^-, y^-) and (x^+, y^+) and contains all $(x, y) \in \mathcal{L}$ such that $x^- \leq x \leq x^+$ and $y^- \leq y \leq y^+$. To denote the complement of a region r, I will use $\bar{r} = \mathcal{L} \setminus r$.

At times we will assume a *reachability predicate*. This predicate specifies if a given object can move between two location in one unit of time.

Definition 1 (Reachability Predicate)**.** *A reachability predicate is a function*

$$reachable : ID \times \mathcal{L} \times \mathcal{L} \rightarrow \{true, false\}$$

where $reachable(id, L_1, L_2)$ *tells if one can reach location* L_2 *from location* L_1 *in one unit of time.*

The reachability predicate is given *a priori*. We will say $reachable(id, k, L, L')$ iff L' is reachable from L in k time points, or more formally: there exists $L = L_0, \cdots, L_k = L'$ such that $reachable(id, L_i, L_{i+1})$ for all $i \geq 1, i < k$.

Example 2. *If we have maximal velocity* v_{id} *for each* $id \in ID$*, then we can use* $reachable(id, L_1, L_2) = (v_{id} \leq ed(L_1, L_2))$ *as a reachability predicate. In this case, if the object* id *is at location* L_1 *at time* 1*, then at time* 2 *it can only be at locations* L' *such that* L' *has distance less than* v_{id} *from* L_1*.*

Probability intervals will be specified as a closed set: $[\ell, u] \subset [0, 1]$.

3.2.2 SPOT atoms

SPOT atoms represent statements of the form: "object id was in region r at time t with a probability between ℓ and u". For instance, I could be between 80% and

90% certain that I saw my friend Larry in the park at noon, leading to the statement: "**Larry** was in **the park** at **noon** with a probability between **0.8** and **0.9**". To represent such a statement, I use a SPOT atom [38, 40].

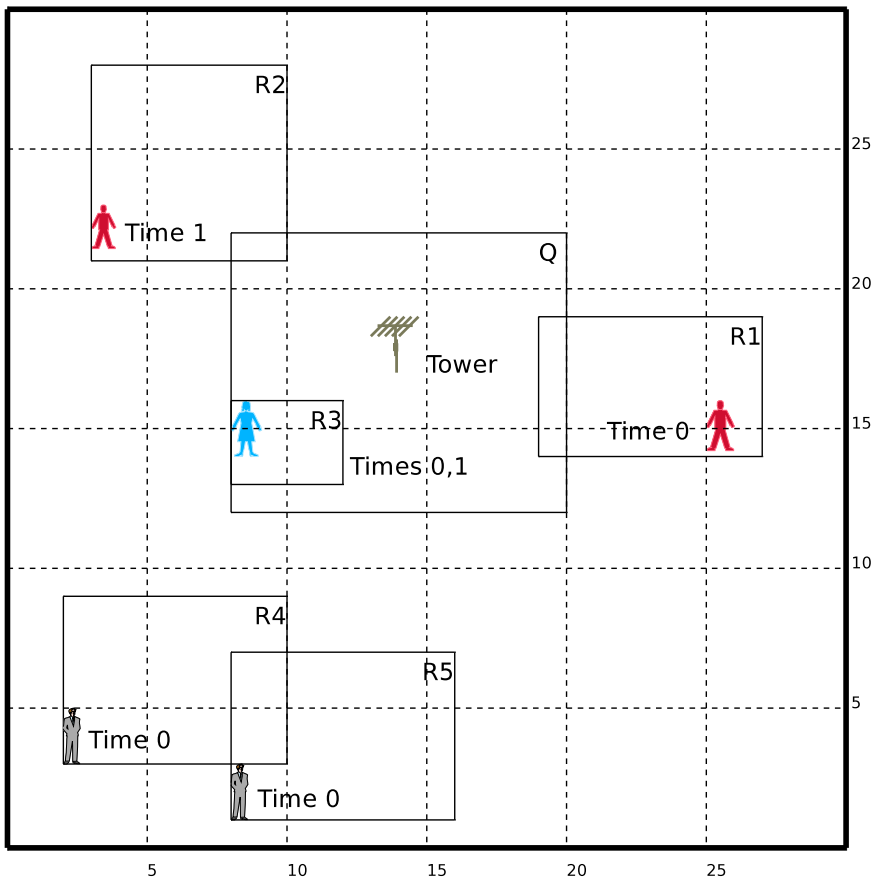
Definition 2 (SPOT Atom)**.** *For an object $id \in ID$, a time point $t \in T$, a region r and probability bounds $[\ell, u] \in [0, 1]$, $(id, r, t[\ell, u])$ is a* SPOT *atom. A set of* SPOT *atoms $\mathcal{S}$ is a* SPOT *database.*

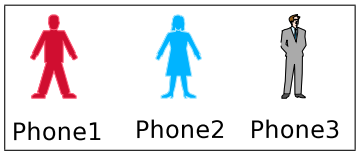
In this definition we do not require the region r to be closed or rectangular, but for storage efficiency reasons, rectangular regions may be preferred in practice.

Example 3. *The table shown in Fig. 3.1 is an example of* SPOT *database. The first row in this table specifies that $Phone1$ is in region $R1$ at time 0 with probability between 0.7 and 0.75.*

Probabilistic intervals are critical to this representation and serve the dual purpose of allowing for less exact data and providing a mechanism to avoid assumptions about conditional dependence. There are many cases where an exact prediction of a probabilistic occurrence is not possible. Even poll results on TV are given as intervals ($47\% \pm 2$ of voters in such-and-such district voted for so-and-so, giving the interval 45%-49%). Further, it is well known that even if we know point probabilities p_1, p_2 respectively for the occurrence of events e_1, e_2 respectively, the probability of $(e_1 \wedge e_2)$ and $(e_1 \vee e_2)$ cannot be determined exactly *unless additional assumptions (e.g. independence) are made.* In fact, this result has been known for over a century (Boole [11], Hailperin [27], Fagin et. al. [23], Ng and Subrahmanian[35]). Thus, unless one plans to make additional assumptions, it is best to use probability intervals.

For a given SPOT database $\mathcal{S}$, we say $\mathcal{S}^{id,t}$ for the subset of the database contain-





SPOT Database $\mathcal{S}_{exm}$
(Phone1, R1, 0, [0.7, 0.75])
(Phone1, R2, 1, [0.6, 0.9])
(Phone2, R3, 0, [0.9, 1])
(Phone2, R3, 1, [0.95, 1])
(Phone3, R4, 0, [0.8, 0.9])
(Phone3, R5, 0, [0.7, 0.9])

Figure 3.1: An example SPOT database with three cell phone carriers and two time points.

ing only atoms having to do with id ID and time point t. That is:

$$\mathcal{S}^{id,t} = \{(id, r, t, [\ell, u]) \in \mathcal{S}\}.$$

3.3 Time-Point Semantics

Time point semantics assign a probability distribution to each object id at each time t. All such distributions will be interpretations.

Definition 3 (Time-Point Interpretation)**.** *A function $I : ID \times T \times \mathcal{L} \rightarrow [0, 1]$ is a* time point interpretation *iff for all $id \in ID$ and $t \in T$:*

$$\sum_{L \in \mathcal{L}} I(id, t, L) = 1.$$

We let $\mathcal{I}$ be the set of all time-point interpretations.

Example 4. *For our running example in Figure 3.1, one possible interpretation which we call I_1 is the one which assigns probability 1 to each of {Phone1, Phone2, Phone3} being at location $(0, 0)$ at both times 0 and 1. This interpretation does not reflect the information in the database. One that does might be I_2, defined as:*

$$I_2(Phone1, 0, (20, 15)) = 0.75, \quad I_2(Phone1, 0, (25, 10)) = 0.25,$$
$$I_2(Phone1, 1, (5, 25)) = 0.7, \quad I_2(Phone1, 1, (15, 20)) = 0.3,$$
$$I_2(Phone2, 0, (10, 15)) = 1, \quad I_2(Phone2, 1, (10, 15)) = 1,$$
$$I_2(Phone3, 0, (9, 5)) = 0.9, \quad I_2(Phone3, 0, (20, 5)) = 0.1,$$
$$I_2(Phone3, 1, (15, 15)) = 0.5, \quad I_2(Phone3, 1, (15, 16)) = 0.5.$$

For all unmentioned parameters $I_2(\cdot, \cdot, \cdot) = 0$.

Such interpretations have the ability to satisfy the constraints imposed by SPOT atoms.

Definition 4 (Time-Point Satisfaction)**.** *For a* SPOT *atom* $sa = (id, r, t, [\ell, u])$ *and a time-point interpretation* I*,* I satisfies sa *iff*

$$\ell \leq \sum_{L \in r} I(id, t, L) \leq u.$$

I satisfies *a* SPOT *database* $\mathcal{S}$ *iff* I *satisfies every* $sa \in \mathcal{S}$*.*

Example 5. *For* I_1 *and* I_2 *in Example 4,* I_2 *satisfies the database* $\mathcal{S}_{exm}$ *in Fig. 3.1, while* I_1 *does not satisfy the database in Fig. 3.1.*

We let $\mathcal{I}(\mathcal{S})$ be the set of interpretations satisfying $\mathcal{S}$ and $\mathcal{I}(sa)$ be the set of interpretations satisfying sa. A database is consistent only if it has a satisfying interpretation.

Definition 5. SPOT *database* $\mathcal{S}$ *is consistent iff* $\mathcal{I}(\mathcal{S}) \neq \emptyset$*.*

Example 6. *The database* $\mathcal{S}_{exm}$ *in Figure 3.1 is consistent under time-point semantics. If we were to add the atom* $(Phone1, R4, 0, [0.45, 0.45])$ *to* $\mathcal{S}_{exm}$*, it would no longer be consistent as there is no interpretation which can assign probabilities of* 0.45 *and* 0.7 *to disjoint regions* $R4$ *and* $R1$*.*

We say an atom is *compatible* with a SPOT database iff adding the atom to the database results in a consistent database.

Definition 6. SPOT *atom* sa *is compatible with* SPOT *database* $\mathcal{S}$ *(denoted* $sa \Subset \mathcal{S}$*) iff* $\mathcal{S} \cup \{sa\}$ *is consistent.*

Example 7. *For database* $\mathcal{S}_{exm}$ *in Figure 3.1, atom* $(Phone1, R4, 0, [0.45, 0.45])$ *(from Example 6) is not compatible with* $\mathcal{S}_{exm}$*. However,* $(Phone1, R4, [0.3, 0.3])$ *is compatible with* $\mathcal{S}_{exm}$*.*

Entailment is defined in the usual logical manner.

Definition 7 (Entailment)**.** *For* SPOT *databases* $\mathcal{S}_1, \mathcal{S}_2$*, and* SPOT *atoms* sa_1, sa_2 *we say*

- $\mathcal{S}_1$ entails $\mathcal{S}_2$ *(denoted* $\mathcal{S}_1 \models \mathcal{S}_2$*) iff* $\mathcal{I}(\mathcal{S}_1) \subseteq \mathcal{I}(\mathcal{S}_2)$
- $\mathcal{S}_1$ entails sa_1 *(denoted* $\mathcal{S}_1 \models sa_1$*) iff* $\mathcal{I}(\mathcal{S}_1) \subseteq \mathcal{I}(sa_1)$
- sa_1 entails $\mathcal{S}_1$ *(denoted* $sa_1 \models \mathcal{S}_1$*) iff* $\mathcal{I}(sa_1) \subseteq \mathcal{I}(\mathcal{S}_1)$
- sa_1 entails sa_2 *(denoted* $sa_1 \models sa_2$*) iff* $\mathcal{I}(sa_1) \subseteq \mathcal{I}(sa_2)$.

Example 8. *Again considering* $\mathcal{S}_{exm}$ *from Figure 3.1 along with* SPOT *atom* $sa = (Phone1, R4, 0, [0, 0.3])$*, we have that* $\mathcal{S}_{exm} \models sa$ *but not that* $sa \models \mathcal{S}_{exm}$*. We do however have that* $sa \models (Phone1, R1, 0, [0, 0.7])$.

In the below example, we present two equivalent databases: they mean the same thing but say it in different ways.

Example 9. *Consider if* $L = \{L_1, L_2\}$*,* $\mathcal{S}_1 = \{(0, \{L_1\}, 0, [0.5, 0.5])\}$*, and* $\mathcal{S}_2 = \{(0, \{L_2\}, 0, [0.5, 0.5])\}$*. Both databases satisfy only the interpretation* I *where:*

$$I(0, 0, L_1) = 0.5 \quad I(0, 0, L_2) = 0.5$$

Thus $\mathcal{I}(\mathcal{S}_1) = \mathcal{I}(\mathcal{S}_2)$.

This is formalized in the definition of equivalence:

Definition 8. SPOT *databases* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are* equivalent *iff* $\mathcal{I}(\mathcal{S}_1) = \mathcal{I}(\mathcal{S}_2)$.

Equivalence between $\mathcal{S}_1$ and $\mathcal{S}_2$ is denoted $\mathcal{S}_1 \equiv \mathcal{S}_2$. I occasionally abuse this notation slightly and say that SPOT atoms sa_1 and sa_2 are equivalent ($sa_1 \equiv sa_2$) when what is meant is that the associated singleton SPOT databases are equivalent ($\{sa_1\} \equiv \{sa_2\}$).

3.4 World-based Semantics

One major issue with time-point based semantics is that, for any object id, even if the object cannot move from point L_1 to point L_k in one time point, the database:

$$(id, \{L_1\}, 0, [1, 1]), (id, \{L_k\}, 1, [1, 1])$$

will still be considered consistent despite the fact that it *requires* the object to move from L_1 to L_k in one time point. In some applications, this is not an issue – movement constraints of this form can sometimes be assumed to have been coded into the SPOT database. However, there are cases when more detailed reasoning techniques will be needed to incorporate information about where an object can travel. In these cases, one should use world based semantics.

In world based semantics, we define the set of worlds, $\mathcal{W}$, to be the set of all possible paths all objects may take.

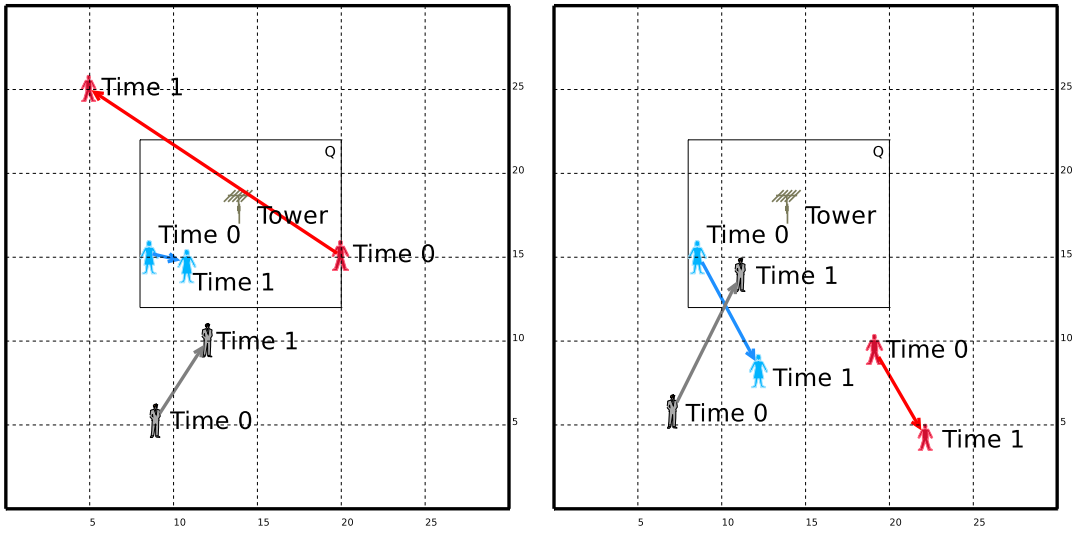
Definition 9. *A* world *is a function $w : ID \times T \rightarrow \mathcal{L}$ that specifies where object id is at time t. The set $\mathcal{W}$ is the set of worlds that satisfy the reachability condition:*

$$\mathcal{W} = \{w \,|\, \forall id \in ID, \forall t, t+1 \in T, reachable(id, w(id, t), w(id, t+1))\}.$$

Example 10. *World $w1$ in Figure 3.2(a) places Phone1 at $(20, 15)$ at time 0 and at $(5, 25)$ at time 1. It places Phone2 at $(8, 15)$ at time 0 and at $(11, 14)$ at time 1. Phone3 is placed at $(9, 6)$ at time 0 and at $(12, 10)$ at time 1. Other worlds are displayed in Figures 3.2(b)-3.2(d).*

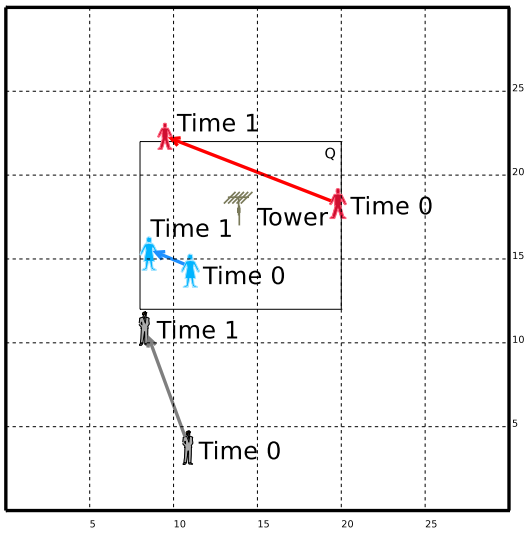
Interpretations in world based semantics are probability distributions over $\mathcal{W}$.

Definition 10 (World-based Interpretation)**.** *A* world-based interpretation *I is a function $I : \mathcal{W} \rightarrow [0, 1]$ such that $\sum_{w \in \mathcal{W}} I(w) = 1$.*

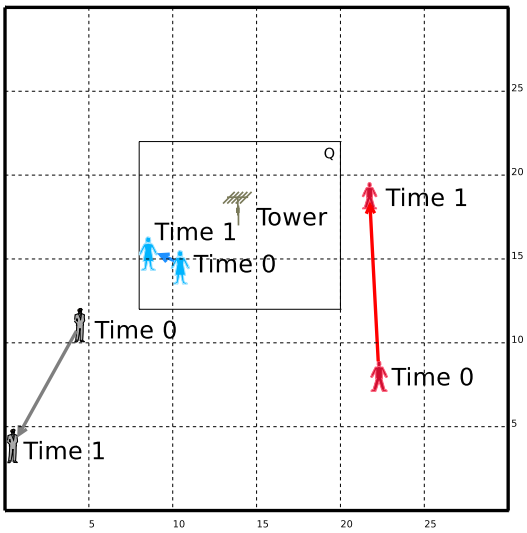


(a) World $w1$

(b) World $w2$



(c) World $w3$



(d) World $w4$

Figure 3.2: Several example worlds.

We let $\mathcal{I}^{\mathcal{W}}$ be the set of all world-based interpretations.

Example 11. *Using the worlds in Figures 3.2(a)-3.2(d), an example interpretation will be* I_1*, which assigns a probability of* 0.70 *to world* $w1$*,* 0.05 *to world* $w2$*,* 0.05 *to world* $w3$*,* 0.20 *to world* $w4$*, and* 0 *to every other possible world.*

One major issue in world-based semantics is the size of the representation of an interpretation I. $|\mathcal{W}|$ is bounded by $|\mathcal{L}|^{|T|\cdot|ID|}$, and for most reasonable reachability definitions will be exponential in $|T|$ and $|ID|$. Since a world-based interpretation I may need to store a separate value of each world, this makes the naïve storage of I intractable in the general case. However, in the section on consistency checking in world based semantics (Section 4.2 on page 60) I will detail an alternative efficient representation.

In these new semantics, we have new definitions for satisfaction, entailment, and so forth.

Definition 11 (World-based Satisfaction)**.** *A world-based interpretation* I *satisfies a* SPOT *atom* $sa = (id, r, t, [\ell, u])$ *iff*

$$\ell \leq \sum_{w(id,t)\in r} I(w) \leq u.$$

A I *satisfies* SPOT *database* $\mathcal{S}$ *iff* I *satisfies all* $sa \in \mathcal{S}$*.*

Example 12. *The interpretation* I_1 *from Example 11 satisfies the database* $\mathcal{S}_{exm}$ *from Figure 3.1.*

To distinguish from time-point semantics, we say $\mathcal{I}^{\mathcal{W}}(sa)$ is the set of world-based interpretations which satisfy sa. Similarly $\mathcal{I}^{\mathcal{W}}(\mathcal{S})$ is the set of world-based interpretations which satisfy $\mathcal{S}$. When it is clear from context that only the world-based semantics are possible, $\mathcal{I}(\mathcal{S})$ will be used instead of $\mathcal{I}^{\mathcal{W}}(\mathcal{S})$.

Definition 12 (World-based Consistency)**.** *A* SPOT *database* $\mathcal{S}$ *is* consistent *iff*

$$\mathcal{I}^{\mathcal{W}}(\mathcal{S}) \neq \emptyset.$$

Definition 13 (World-based Entailment)**.** *For* SPOT *databases* $\mathcal{S}_1, \mathcal{S}_2$*, and* SPOT *atoms* sa_1, sa_2 *we say*

- $\mathcal{S}_1$ entails $\mathcal{S}_2$ *(denoted* $\mathcal{S}_1 \models \mathcal{S}_2$*) iff* $\mathcal{I}^{\mathcal{W}}(\mathcal{S}_1) \subseteq \mathcal{I}^{\mathcal{W}}(\mathcal{S}_2)$
- $\mathcal{S}_1$ entails sa_1 *(denoted* $\mathcal{S}_1 \models sa_1$*) iff* $\mathcal{I}^{\mathcal{W}}(\mathcal{S}_1) \subseteq \mathcal{I}^{\mathcal{W}}(sa_1)$
- sa_1 entails $\mathcal{S}_1$ *(denoted* $sa_1 \models \mathcal{S}_1$*) iff* $\mathcal{I}^{\mathcal{W}}(sa_1) \subseteq \mathcal{I}^{\mathcal{W}}(\mathcal{S}_1)$
- sa_1 entails sa_2 *(denoted* $sa_1 \models sa_2$*) iff* $\mathcal{I}^{\mathcal{W}}(sa_1) \subseteq \mathcal{I}^{\mathcal{W}}(sa_2)$.

This concludes the formalization of the world based semantics.

3.5 SPOT Queries

3.5.1 Selection Query Semantics

In this work I place major emphasis on the running time of selection queries. There are two kinds of selection queries, optimistic and cautious. The optimistic queries return sets of facts which *can* be true, while the cautious ones return sets of facts which *must* be true.

You will notice that these queries are applicable to both the point-based and world-based semantics. However, for ease of presentation, and because most of the query algorithms focus on point-based semantics, I will assume point-based semantics for the introduction of these queries.

Definition 14 (Optimistic selection)**.** *Let $\mathcal{S}$ be a* SPOT *database and* $(?id, r, ?t, [\ell, u])$ *be a selection query. The* optimistic answer *to* $(?id, r, ?t, [\ell, u])$ *is the set*

$$\{(id, r, t, [\ell, u]) \mid id \in ID \wedge t \in T \wedge (id, r, t, [\ell, u]) \Subset \mathcal{S}\}.$$

Example 13. *Again referring to database $\mathcal{S}_{exm}$ from Figure 3.1. The cell phone company is interested in knowing who will be using the cell tower and when they will be expected to use it. Since the tower services only those in region Q, this question can be answered by the* SPOT *database with the selection query* $(?id, Q, ?t, [0.75, 1])$*, which asks who will be in the region Q served by the cell tower with a probability of at least 75%. The optimistic answer tells who could be in Q with at least probability 75%. The optimistic answer is: Phone1 at times 0 and 1, Phone2 at times 0 and 1, and Phone3 at time 1. Phone3 cannot be in the query region at time* 0 *as it will be in the regions R4 and R5 with high enough probability to eliminate the possibility of being in the query region.*

Definition 15 (Cautious selection)**.** *Let $\mathcal{S}$ be a* SPOT *database and* $(?id, r, ?t, [\ell, u])$ *be a selection query. The* cautious answer *to* $(?id, r, ?t, [\ell, u])$ *is the set*

$$\{(id, r, t, [\ell, u]) \mid id \in ID \wedge t \in T \wedge \mathcal{S} \models (id, r, t, [\ell, u])\}.$$

Example 14. *If the cell phone company wants to know who is definitely in the query region with a probability of at least 75%, the query can be posed as a cautious query. The cautious answer is: Phone2 at times 0 and 1.*

It is clear that the optimistic answer to any selection query is a superset of (or equal to) the cautious answer. Optimistic and cautious answers to queries build upon the notion of optimistic and cautious semantics for non-monotonic logics (also known as the brave and credulous semantics, respectively) [22].

Pruning techniques and algorithms for solving optimistic and cautious selection queries are given in Chapter 7.

Chapter 4

Consistency Checking in SPOT Databases

Consistency checking algorithms check that there is a satisfying interpretation for a given SPOT database. Because there are two different kinds of interpretations for the two semantics, two approaches to consistency checking are explored.

All of the algorithms in this section will use linear programming, an attractive technique due to both its polynomial running time and the depth of existing research on linear program solving. Most of the below algorithms create a linear program and solve it using standard techniques, making their running times polynomial in the size of the created linear program. Much is gained by shrinking the sizes of the created linear programs, and many of these results focus on slimming the representation or on applying divide-and-conquer techniques to ensure faster computation.

4.1 Point-based semantics

The point based semantics allow several consistency checking algorithms, some more efficient than others.

4.1.1 Naïve Consistency Algorithm

We now provide an algorithm to check the consistency of a SPOT database under the point based semantics. The algorithm uses linear programming. By finding a solution to one linear program per $\langle id, t \rangle$, we can guarantee the existence of of a point-based interpretation satisfying the database.

In the linear program, each point $(x, y) = L \in \mathcal{L}$ has an associated variable $v_{x,y}$ that specifies the probability that the object with id id is at (x, y) at time t. The linear program contains two constraints per SPOT atom as well as constraints ensuring $v_{x,y} \geq 0$ and the sum of all variables is 1.

Definition 16 ($LP(\cdot)$)**.** *The* set of linear constraints*,* $LP(\mathcal{S}, id, t)$ *associated with* $\mathcal{S}, id, t$ *contains exactly the following constraints:*

- *For all atoms* $(id, r, t, [\ell, u]) \in \mathcal{S}^{id,t}$*, the constraint* $\ell \leq \sum_{(x,y)\in r} v_{x,y}$ *is in* $LP(\mathcal{S}, id, t)$.
- *For all atoms* $(id, r, t, [\ell, u]) \in \mathcal{S}^{id,t}$*, the constraint* $u \geq \sum_{(x,y)\in r} v_{x,y}$ *is in* $LP(\mathcal{S}, id, t)$.
- *The constraint* $\sum_{(x,y)\in\mathcal{L}} v_{x,y} = 1$ *is in* $LP(\mathcal{S}, id, t)$.
- *The constraints* $v_{x,y} \geq 0$ *for all* $(x, y) \in \mathcal{L}$ *are in* $LP(\mathcal{S}, id, t)$.

The constraint listed in the first bullet forces any solution to satisfy the lower bound requirements for any SPOT atom referencing $\langle id, t \rangle$. Likewise, the second bullet's constraint forces satisfaction of those atoms' upper bound. The constraint listed in the third bullet simply says the object must exist somewhere in the overall space with probability 1, and the constraints in the fourth bullet ensure no $v_{x,y}$ can have a negative value (necessary because each $v_{x,y}$ represents a probability).

The algorithm for checking consistency feeds the above inequalities, combined for all id and t, into any linear constraint solver [30] and returns true if there is a solution. That is, $\mathcal{S}$ is consistent if, for every $id \in ID$ and $t \in T$, $LP(\mathcal{S}, id, t)$ has a solution. We call this algorithm the *naïve* consistency checking algorithm.

Proposition 1. *The naïve consistency checking algorithm checks the consistency of a* SPOT *database $\mathcal{S}$ in time polynomial in $|ID|$, $|T|$, $|\mathcal{L}|$, and $|\mathcal{S}|$.*

Proof. The time taken for naïve consistency checking to run is bounded by $|ID| \times |T| \times p$ where p bounds the time to solve the linear program $LP(\mathcal{S}, id, t)$. $LP(\mathcal{S}, id, t)$ is always polynomial in the size of $\mathcal{L}$ (there are $|\mathcal{L}|$ variables) and $\mathcal{S}$ (there are at most $2 \times |\mathcal{S}| + 1$ constraints) and since linear programs are solvable in time polynomial in the size of the linear program [32], p must is a polynomial, making $|ID| \times |T| \times p$ a polynomial in $|ID|$, $|T|$, $|\mathcal{L}|$, and $|\mathcal{S}|$. □

Though the above theorem establishes a polynomial result for consistency checking, it should be noted that the size of $\mathcal{L}$ can be quite large (one million for a 1000×1000 point space) and hence, the algorithm for consistency checking can be inefficient.

4.1.2 BSP Consistency Checking

To account for the excessive but polynomial number of variables needed to compute solutions to $LP(\cdot)$, we introduce a partitioning scheme which combines points contained in all the same inequalities into one variable, thereby reducing the number of variables.

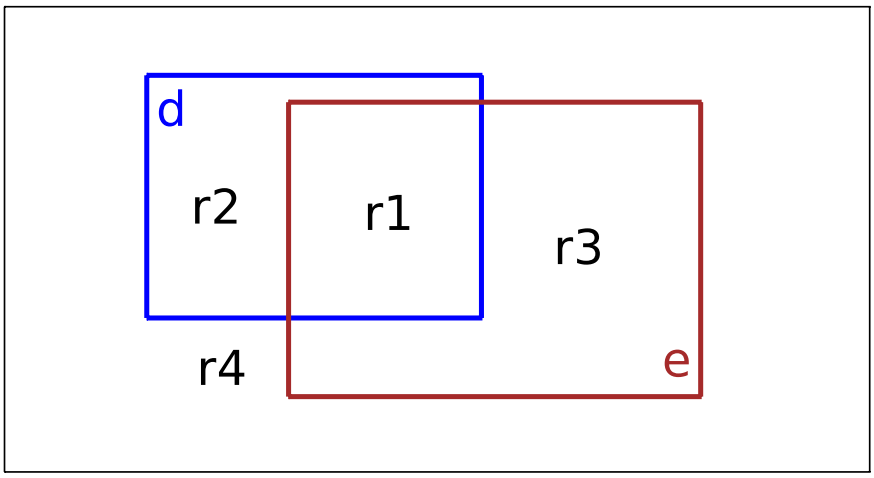


Figure 4.1: Example regions for SPOT atoms.

Intuition

This technique creates regions where all variables will be homogeneous. Consider if we have two rectangles e and d pictured in Figure 4.1. If we have the SPOT database

$$\mathcal{S} = \{(id, e, t, [0.8, 0.9]), (id, d, t, [0.9, 1])\}$$

then this vehicle could be in one of four disjoint regions:

$$r_1 = d \cap e, \qquad r_2 = d - e,$$
$$r_3 = e - d, \qquad r_4 = \mathcal{L} - (d \cup e).$$

Suppose v_i denotes the probability that the vehicle is in region r_i at time t. Rather than have a set of constraints including one variable for each point in the space, we only need these four variables, We can therefore rewrite our linear constraints as follows.

$$0.8 \leq v_1 + v_2 \leq 0.9 \tag{4.1}$$

$$0.9 \leq v_1 + v_3 \leq 1 \tag{4.2}$$

$$v_1 + v_2 + v_3 + v_4 = 1. \tag{4.3}$$

Constraint 4.1 is derived from the SPOT atom $(id, d, t, 0.8, 0.9)$. id is in region d in the cases corresponding to rectangles r_1 and r_2 — hence, the variables v_1 and v_2 must add up to a value in the range 0.8 to 0.9 which is what the first constraint above says. The reader will readily believe that this linear program can be solved more efficiently than $LP(\mathcal{S}, id, t)$. This will also greatly speed up the naïve consistency checking algorithm.

Variable elimination: formalization

We start with the notion of location equivalence.

Definition 17 (Location Equivalence)**.** *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$ and t is a time point. Locations L_1 and L_2 are equivalent w.r.t. $\mathcal{S}, id, t$ denoted $p_1 \sim p_2$ ($\mathcal{S}, id, t$ are understood) iff for all $(id, r, t, [\ell, u]) \in \mathcal{S}^{id,t}$*

$$L_1 \in r \text{ iff } L_2 \in r.$$

Equivalent points are any two points whose probabilities are governed by exactly the same sets of SPOT atoms. For instance, if two points are both in r for some atom $(id, r, t, [\ell, u])$, and neither point is in any other region for any other SPOT atom in $\mathcal{S}^{id,t}$, then they are equivalent. However, if one of the two points were to be contained in r' for some $(id, r', t, [\ell', u'])$ not containing the other point, then the points are not equivalent.

The following result states that two equivalent points will have the same range of probabilities assigned to them. The minimum and maximum probabilities of an object being at two equivalent points at the same time are the same.

Lemma 1. *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$ and t is a time point. If $(x_1, y_1) \sim (x_2, y_2)$ then*

- $\min_{I \models \mathcal{S}}(I(id, (x_1, y_1), t)) = \min_{I \models \mathcal{S}}(I(id, (x_2, y_2), t))$
- $\max_{I \models \mathcal{S}}(I(id, (x_1, y_1), t)) = \max_{I \models \mathcal{S}}(I(id, (x_2, y_2), t))$

Proof. By Definition 17 v_{x_1,y_1} and v_{x_2,y_2} occur in exactly the same way in $LP(\mathcal{S}, id, t)$ (Definition 16). Thus their minimization and maximization are the same. □

Suppose we are given a SPOT database $\mathcal{S}$, a vehicle $id \in ID$, and a time-point t. Since location equivalence w.r.t. $\mathcal{S}, id, t$ is an equivalence relation (i.e. it is reflexive, transitive and symmetric), we can use it to partition $\mathcal{L}$ into equivalence classes $\mathcal{P}_1, \cdots, \mathcal{P}_m$. Let us call this partition $\mathcal{P}$. Notice that there is exactly one such partition $\mathcal{P}$ for any $\mathcal{S}, id, t$. Suppose now that r is an arbitrary region in $\mathcal{L}$. Let $\mathcal{P}(r)$ be the set $\{\mathcal{P}_i \mid \mathcal{P}_i \cap r \neq \emptyset\}$.

Proposition 2. *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$ and t is a time-point. Let $\mathcal{P}$ be the partitioning of $\mathcal{L}$ described above and let r be a rectangle mentioned in an atom of $\mathcal{S}$. Then $\bigcup_{\mathcal{P}_i \in \mathcal{P}(r)} \mathcal{P}_i = r$.*

We can now associate a single variable, v_i with each member $\mathcal{P}_i$ of the partition induced by $\sim$ on $\mathcal{S}^{id,t}$, and create a new set of linear constraints based on the partitioning.

Definition 18 (Partitioned Linear Program (PLP))**.** *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$, and t is a time-point. Let $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_m)$ be the partitioning induced by the location equivalence relation $\sim$ w.r.t. $\mathcal{S}, id, t$. Then the* partitioned linear constraints*, $PLP(\mathcal{S}, id, t)$, associated with $\mathcal{S}, id, t$ is defined as follows:*

- *For all $(id, r, t, [\ell, u]) \in \mathcal{S}^{id,t}$, the constraint $\ell \leq \sum_{\mathcal{P}_i \in \mathcal{P}(r)} \mathsf{v}_i \leq u$*
- *The constraint $\sum_{\mathcal{P}_i \in \mathcal{P}} \mathsf{v}_i = 1$*

- *The constraint* $\mathsf{v}_i \geq 0$ *for all* $\mathcal{P}_i \in \mathcal{P}$.

This formulation of the linear program is useful mainly due to its relationship with LP. In particular, PLP is solvable if and only if LP is solvable.

Theorem 1. *Suppose* $\mathcal{S}$ *is a* SPOT *database,* $id \in ID$ *and* t *is a time-point.* $PLP(\mathcal{S}, id, t)$ *is solvable iff* $LP(\mathcal{S}, id, t)$ *is solvable.*

Proof. ($\Leftarrow$): Let W be a solution to $LP(\mathcal{S}, id, t)$ with $\mathsf{v}_{x,y}(W)$ the value of $\mathsf{v}_{x,y}$ in W. To construct a solution $\bar{W}$ to $PLP(\mathcal{S}, id, t)$ assign to $\mathsf{v}_i(\bar{W})$ the value $\sum_{(x,y)\in\mathcal{P}_i} \mathsf{v}_{x,y}(W)$ (the sum of the probabilities of the points in region $\mathcal{P}_i$). By Proposition 2, each r mentioned in some SPOT atom of $\mathcal{S}$ is exactly the union of the points in $\mathcal{P}(r)$, which is the union of the points in the $\mathcal{P}_i$ comprising r. Because W solves $LP(\mathcal{S}, id, t)$, all the inequalities in $PLP(\mathcal{S}, id, t)$ must hold using the $\mathsf{v}_i(\bar{W})$ as assigned.
($\Rightarrow$): Let $\bar{W}$ be a solution to $PLP(\mathcal{S}, id, t)$. This means that each $\mathsf{v}_i(\bar{W})$ represents the probability of a region $\mathcal{P}_i \in \mathcal{P}$. For each $\mathcal{P}_i \in \mathcal{P}$ pick an arbitrary point (x_i, y_i), and let $\mathsf{v}_{x_i,y_i}(W) = \mathsf{v}_i(\bar{W})$ and let all other points (x, y) not equal to any (x_i, y_i) have $\mathsf{v}_{x,y}(W) = 0$. Thus we concentrate the probability of region $\mathcal{P}_i$ in exactly one point in $\mathcal{P}_i$. Hence, because $\bar{W}$ satisfies the inequalities in $PLP(\mathcal{S}, id, t)$ all the inequalities in $LP(\mathcal{S}, id, t)$ must hold for W. □

As illustrated in Section 4.1.2, the above theorem gives us the ability to check consistency with a new set of linear constraints.

Corollary 1. *A* SPOT *database* $\mathcal{S}$ *is consistent iff for all* id, t, $PLP(\mathcal{S}, id, t)$ *is solvable.*

The number of variables in $PLP(\mathcal{S}, id, t)$ is m - the number of partitions. Each $\mathcal{P}_i$ in the partitions can capture a large number of points and represents them by using

a single variable, and will never capture less than one unique point. Thus, $m \leq |\mathcal{L}|$. As a consequence, the number of variables in $PLP(\mathcal{S}, id, t)$ is guaranteed to be less than or equal to the number of variables in $LP(\mathcal{S}, id, t)$. The number of constraints is the same for both $PLP(\mathcal{S}, id, t)$ and $LP(\mathcal{S}, id, t)$. Thus $PLP(\mathcal{S}, id, t)$ is guaranteed to be no larger than $LP(\mathcal{S}, id, t)$, and will in many cases be substantially smaller.

Corollary 2. *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$, t is a time-point, and r is a region of $\mathcal{L}$ and $[\ell, u]$ is any probability interval. Then:*

1. *(Compatibility)*

   $(id, r, t[\ell, u]) \Subset \mathcal{S}$ *iff there is a solution to* $PLP(\mathcal{S} \cup \{(id, r, t, [\ell, u])\}, id, t)$.

2. *(Entailment)*

   $\mathcal{S} \models (id, r, t, [\ell, u])$ *iff*

   - $\ell \leq$ ***minimize*** $\Sigma_{\mathcal{P}_i \in \mathcal{P}(r)} v_i$ ***subject to*** $PLP(\mathcal{S} \cup \{(id, r, t, [0, 1])\}, id, t)$ *and*
   - $u \geq$ ***maximize*** $\Sigma_{\mathcal{P}_i \in \mathcal{P}(r)} v_i$ ***subject to*** $PLP(\mathcal{S} \cup \{(id, r, t, [0, 1])\}, id, t)$.

The above result provides an immediate algorithm for checking entailment and satisfaction. The running time is polynomial in the size of $PLP(\mathcal{S}, id, t)$, which consists of at most m variables and $|\mathcal{S}| + 1$ constraints (without counting the constraints requiring that all variables are non-negative). In contrast, the results in [40] use the linear program $LP(\mathcal{S}, id, t)$, which contains $|\mathcal{L}|$ variables and exactly the same number of constraints. As $m \leq |\mathcal{L}|$, the results hold promise that the algorithm derived from the preceding theorem and the above corollary will be more efficient - for consistency checking, as well as both cautious and optimistic entailment - than the algorithms proposed in [40]. This intuition is verified experimentally in Section 4.1.4.

Constructing the Partition

For rectangles $r_1, \ldots, r_k$, we compute a partition consisting of up to $min(|\mathcal{L}|, 2^k)$ possible regions. This is done by a straightforward binary space partitioning (BSP) approach.

Our BSP is a tree data structure where each node contains three fields:

1. node.rectangle: the rectangle labeling the node.
2. node.in: the child node for rectangles that intersect the node's region.
3. node.out: the child node for rectangles that intersect the node's region's complement.

When inserting a rectangle r into this BSP, recursively visit nodes through a visit operation. When visiting node N, check if $N.reg \cap r$ is non-empty. If so, visit $N.in$. Also check if $r \cap \overline{N.reg}$ is non-empty, i.e. does r intersect $N.reg$'s complement? If so, visit $N.out$. It may therefore be the case that r gets inserted into both subtrees of N. If the visited node is NIL, create a new node to fill that spot. The complexity of the insertion procedure is linear in the number of nodes in the tree, as every node in the tree may be visited if r covers every region in the tree plus some space not in any region in the tree.

Example 15. *In Figure 4.2 we see a BSP that may be constructed for Phone3 from Figre 3.1 (page 32). Each of the leaves of the BSP will correspond to a variable in* $PLP(\mathcal{S}_{exm}, Phone3, 0)$.

Note that the BSP is used to find a partition $\mathcal{P}$ which, in turn, is used to build the set $PLP(\mathcal{S}, id, t)$ of partitioned linear constraints described in the preceding section.

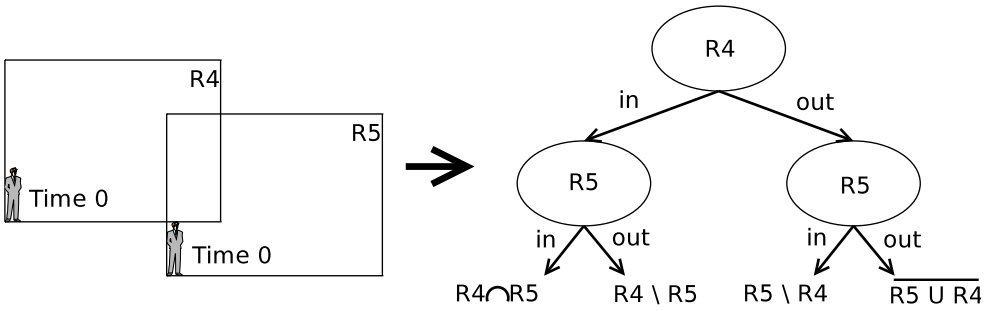


Figure 4.2: An example BSP for Phone3 at time 0.

4.1.3 Atom Clustering

In this section, I develop a set of theoretical results that can be used to divide a SPOT database into independently solvable components in order to divide-and-conquer the consistency checking problem. I do this by first defining r-equivalence between SPOT DBs (intuitively this means that the restrictions one SPOT DB imposes on a region r coincide exactly with the restrictions of the second SPOT DB for the same region). I then present a theorem that r-equivalence can be used to prune SPOT atoms in $\mathcal{S}$ that are irrelevant to the region r. An e-atom is a special atom that entails all regions mentioned in $\mathcal{S}^{id,t}$ minimally. I then define the notion of clusters and provide an algorithm to check consistency using clusters - the algorithm is proven correct.

Suppose we consider a region r. r is not necessarily convex or closed, but is a subset of $\mathcal{L}$. For an interpretation I, we let I_r be I with a range bounded to r. That is, $I_r : ID \times T \times r \rightarrow [0, 1]$ and $I_r(id, t, L) = I(id, t, L)$.

Definition 19 (r-Equivalence)**.** *For* SPOT *databases $\mathcal{S}$, $\mathcal{T}$ and region r, we have $\mathcal{S} \overset{r}{\equiv} \mathcal{T}$ iff $\{I_r \mid I \models \mathcal{S}\} = \{J_r \mid J \models \mathcal{T}\}$.*

In other words, two SPOT databases $\mathcal{S}$ and $\mathcal{T}$ are r-equivalent iff for every in-

terpretation I that satisfies $\mathcal{S}$, there is an interpretation J that satisfies $\mathcal{T}$ where J is identical to I for region r and vice versa.

Example 16. *Imagine two disjoint rectangles r_1 and r_2 that cover space ($r_1 \cup r_2 = \mathcal{L}$). $\mathcal{S}_1 = \{(id, r_1, 1, [0.5, 0.5])\}$ is r_1'-equivalent to $\mathcal{S}_2 = \{(id, r_2, 1, [0.5, 0.7])\}$ for any strict subset r_1' of r_1 ($r_1' \subsetneq r_1$). The probability bounds for any strict subset of r_1 are $[0, 0.5]$ in $\mathcal{S}_1$, which are the same bounds as are allowed for any such region by $\mathcal{S}_2$.*

r-equivalence can informally be used as follows. Suppose r is a region specified in a query to a SPOT database $\mathcal{S}$. We would like to prune away all SPOT atoms in $\mathcal{S}$ that are not relevant to the region r, and focus on the portion $\mathcal{T}$ of $\mathcal{S}$ that is relevant. By focusing on a much smaller $\mathcal{T}$, we hope to generate a smaller linear program. The following theorem tells us that we can use this intuition.

Theorem 2. *Suppose $\mathcal{S}$ and $\mathcal{T}$ are* SPOT *databases such that $\mathcal{S} \overset{q}{\equiv} \mathcal{T}$ w.r.t. a region q. Let f be* any *linear objective function over the variables v_p for $p \in q$. Let $id \in ID$ and t be a time-point. Then the minimization/maximization of f w.r.t. $LC(\mathcal{S}, id, t)$ equals the minimization/maximization of f w.r.t. $LC(\mathcal{T}, id, t)$.*

Proof. By Theorem 1, every SPOT interpretation I that satisfies $\mathcal{S}$ corresponds to a solution of $LP(\mathcal{S}, id, t)$ of the form $\mathsf{v}_p = I(id, t, p)$ for all $p \in \mathcal{L}$. Since $\mathcal{S}$ and $\mathcal{T}$ are q-equivalent, there always exist (possibly distinct) interpretations assigning the same values to $p \in q$ satisfying $\mathcal{S}$ and $\mathcal{T}$. Thus there always exist solutions to each set of linear constraints that assign the same values to v_p for $p \in q$ as a solution to the other set of linear constraints. Thus for any value of f given by a solution to one set of linear constraints, there is a solution to the other set of linear constraints which produces the same values for v_p, $p \in q$. □

The following result states that when $\mathcal{S} \overset{q}{\equiv} \mathcal{T}$, entailment and consistency of atoms involving region q are the same.

Corollary 3. *Suppose $\mathcal{S}$ and $\mathcal{T}$ are* SPOT *databases such that $\mathcal{S} \overset{q}{\equiv} \mathcal{T}$. Then, for any* SPOT *-atom sa of the form $(id, q, t, [\ell, u])$:*

1. $\mathcal{S} \models sa$ *iff* $\mathcal{T} \models sa$.
2. $sa \Subset \mathcal{S}$ *iff* $sa \Subset \mathcal{T}$.[1]

We are interested in taking a SPOT database $\mathcal{S}$ and dividing it up into several smaller databases $\mathcal{T}_1, \ldots, \mathcal{T}_k$. By solving the smaller linear program for each of the smaller $\mathcal{T}_i$ and combining the answers, we should be saving time over simply computing for $\mathcal{S}$, as linear programming takes time polynomial in the product of the number of constraints and number of variables in the linear program. To communicate information from solving $\mathcal{T}_i$, we will use a special kind of SPOT atom defined below.

Definition 20 (e-atom)**.** *Suppose $\mathcal{S}$ is a* SPOT *database, $id \in ID$ and t is a timepoint. The entailing* SPOT *atom for the union of the regions mentioned in $\mathcal{S}^{id,t}$ with tightest probability bounds (e-atom for short) is an expression of the form $(id, R, t, [\ell, u])$ where:*

- $R = \cup_{(id,r,t,[\ell,u]) \in \mathcal{S}} r$
- $\ell = \min_{I \models \mathcal{S}} \left(\sum_{p \in R} I(id, p, t) \right)$
- $u = \max_{I \models \mathcal{S}} \left(\sum_{p \in R} I(id, p, t) \right)$

Example 17. *For instance, if we have* SPOT *database*

$$\mathcal{S} = \{(id, r_1, 1, [0.5, 0.6]), (id, r_2, 1, [0.2, 0.4])\}$$

[1] Recall that $sa \Subset \mathcal{S}$ denotes that sa is compatible with $\mathcal{S}$ as specified in Definition 6 on page 34.

(where r_1 and r_2 are disjoint rectangular regions) there is e-atom $(id, r_1 \cup r_2, 1, [0.7, 1])$ that entails that same probability bounds for the region $r_1 \cup r_2$ as were entailed by $\mathcal{S}$. Notice however that the SPOT *atom $(id, r_1 \cup r_2, 1, [0.6, 1])$ is not an e-atom despite the fact that it is compatible with $\mathcal{S}$: 0.6 is not a maximal lower bound.*

Our goal is to find e-atoms for appropriate "clusters" of SPOT atoms in a SPOT database.

Definition 21 (SPOT overlaps)**.** *We say that* SPOT *atom $(id, r, t, [\ell, u])$* overlaps SPOT *atom $(id, r', t, [\ell', u'])$ iff $r \cap r' \neq \emptyset$. We use the symbol $\triangle$ to denote the "overlaps" relation and $\triangle^\star$ to denote the transitive closure of this relation.*

It is easy to see that $\triangle^\star$ is an equivalence relation.

Definition 22 (Cluster)**.** *Suppose $\mathcal{S}$ is a* SPOT *database. A cluster w.r.t. $\mathcal{S}$ is* any $\triangle^\star$-equivalence class *of $\mathcal{S}$.*

Example 18. *Considering the database in Figure 3.1, $(Phone3, R4, 0, [0.8, 0.9])$ and $(Phone3, R5, 0, [0.7, 0.9])$ are in the same $\triangle^\star$ equivalence class. However if there were an atom $(Phone3, R1, 0, [0.1, 0.1])$, it would be in a different $\triangle^\star$ equivalence class.*

The following theorem considers SPOT databases that refer to a single id and t. It says that when we consider the set of all points in $\mathcal{L}$ not covered by any rectangle mentioned in $\mathcal{S}$, then we can find a SPOT database that is equivalent to $\mathcal{S}$ w.r.t. this set of points. The method is simple: find the clusters of the SPOT database and then find the e-atom for each cluster.

Theorem 3. *Suppose $\mathcal{S}$ is a consistent* SPOT *database, $id \in ID$ and t is a time point. Let $C_1, ..., C_k$ be the clusters of $\mathcal{S}^{id,t}$. Let $U = \bigcup_{(id,r,t,[\ell,u]) \in \mathcal{S}} r$. Suppose $\bar{U}$ is*

Algorithm 1 Algorithm for computing the cluster associated with SPOT atom sa in the database $\mathcal{S}^{id,t}$

Cluster Compute($sa, \mathcal{S}^{id,t}$)

$A = \emptyset$

$A_n = \{sa\}$

while $A \neq A_n$ **do**

$A = A_n$

$$A_n = \left\{ (id, r, t, [\ell, u]) \in \mathcal{S}^{id,t} \;\middle|\; \begin{array}{c} (id, r', t, [\ell', u']) \in A \\ \wedge\, r \cap r' \neq \emptyset \end{array} \right\}$$

end while

return A.

Algorithm 2 Algorithm for computing all clusters of $\mathcal{S}^{id,t}$.

Clusters($\mathcal{S}^{id,t}$)

$Clusters = \emptyset$

while $\cup_{C \in Clusters} C \neq \mathcal{S}^{id,t}$ **do**

Take $sa \in \mathcal{S}^{id,t} \setminus (\cup_{C \in Clusters} C)$

Let $Clusters = Clusters \cup \{$Cluster Compute($sa, \mathcal{S}^{id,t}$)$\}$.

end while

return $Clusters$

non-empty and suppose e_i is the e-atom for cluster C_i. Let $\mathcal{T} = \{e_1, \ldots, e_k\}$. Then $\mathcal{S}$ is $\bar{U}$-equivalent to $\mathcal{T}$.

Proof. Clearly, if $I \models \mathcal{S}$ then $I \models \mathcal{T}$, hence $\{I_{\bar{U}}\} \subseteq \{J_{\bar{U}}\}$ in Definition 19. We must also show that if $J \models \mathcal{T}$ then $\{J_{\bar{U}}\} \subseteq \{I_{\bar{U}}\}$ for $I \models \mathcal{S}$. Note that $J \models \mathcal{T}$ does not necessarily mean that $J \models \mathcal{S}$ because $\mathcal{S}$ may be more precise in allocating probabilities to regions. However, this does not matter as far as $J_{\bar{U}}$ is concerned because no explicit probability restriction is given for any part of $\bar{U}$. So any such J, where $J \models \mathcal{T}$ may be modified on U to an I, where $I \models \mathcal{S}$, without changing the probability distribution on $\bar{U}$. Hence $\{J_{\bar{U}}\} \subseteq \{I_{\bar{U}}\}$. □

The above theorem can be applied to a SPOT database that references multiple ids and ts by applying it to each (id, t) pair.

Moreover, clusters may allow us to further improve the efficiency of checking if a SPOT database is consistent. Algorithm 3 checks the consistency of clusters individually and then checks the sum of the lower and upper bounds of the e-atoms for the clusters.

Algorithm 3 Checks the consistency of $\mathcal{S}$ using clusters.

Consistency Checking with Clusters($\mathcal{S}$)
  for every id, t **do**
    Compute clustering $C_1, ..., C_m$ of $\mathcal{S}^{id,t}$ via Algorithm 2
    for each C_i **do**
      return false if C_i is not internally consistent.
      Let r_i be the union of the regions of all SPOT atoms in the associated C_i.
      Compute ℓ_i as the $\max_{I \models \mathcal{S}} \sum_{p \in r_i} I(id, t, p)$
      Compute u_i as the $\min_{I \models \mathcal{S}} \sum_{p \in r_i} I(id, t, p)$
      Let $e_i = (id, r_i, t, [\ell_i, u_i])$ be the e-atom for C_i.
    end for
    return false **if** $\sum_{i=1}^{m} \ell_i > 1$.
    if $\mathcal{L} = \bigcup_{1 \leq i \leq m} r_i$ **then**
      return false **if** $\sum_{i=1}^{m} u_i < 1$.
    end if
  end for
  return true.

4.1.4 Experimental Results

We implemented all the point-based consistency checking algorithms and tested them. This section describes the results of these tests. Our system runs on a PC machine with a Xeon 3.4 GHz processor and RedHat/Linux operating system and consists of approximately 3000 lines of Java-1.5 code (plus the QSOPT library for linear programming [2] and the spatialindex[2] library for R*-tree like structures).

[2] `http://research.att.com/~marioh/spatialindex/index.html`

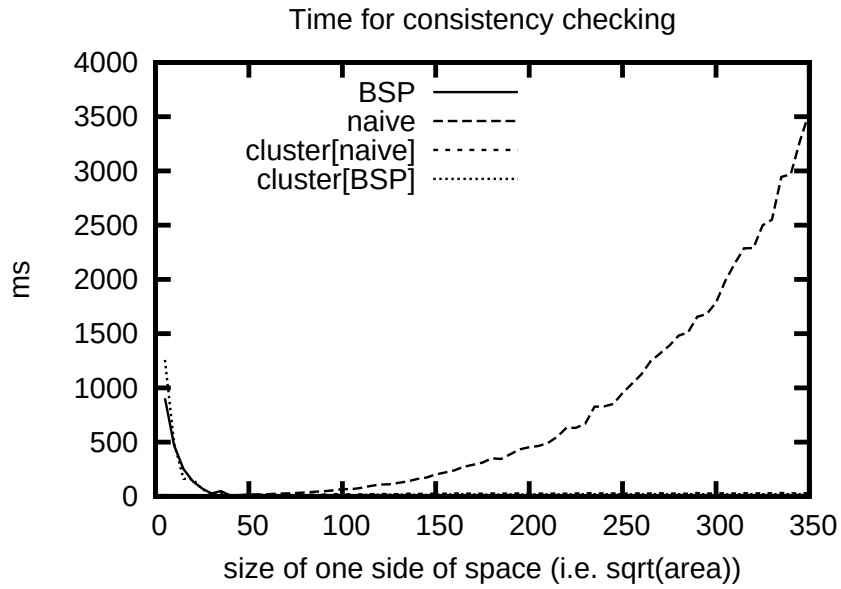


Figure 4.3: 1st experiment: varying size of space with synthetic datasets and point-based semantics.

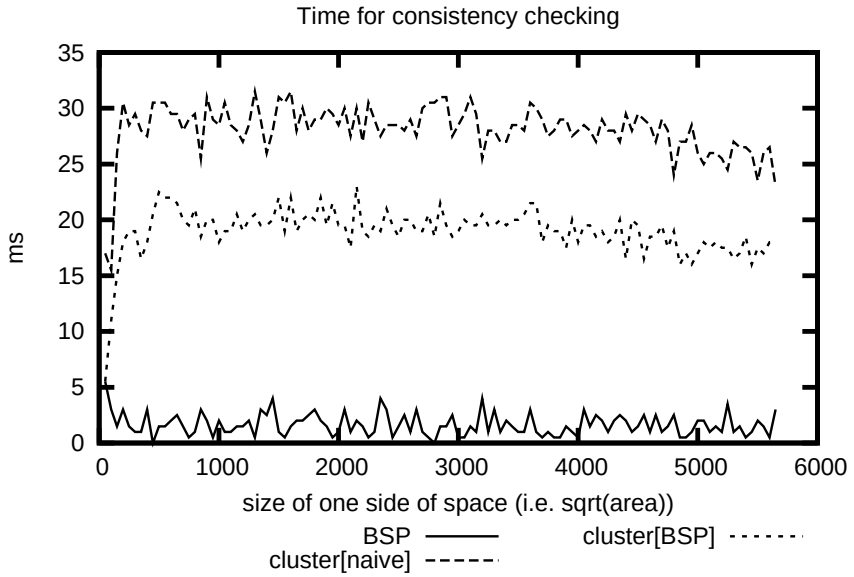


Figure 4.4: Large size for space with synthetic datasets and point-based semantics.

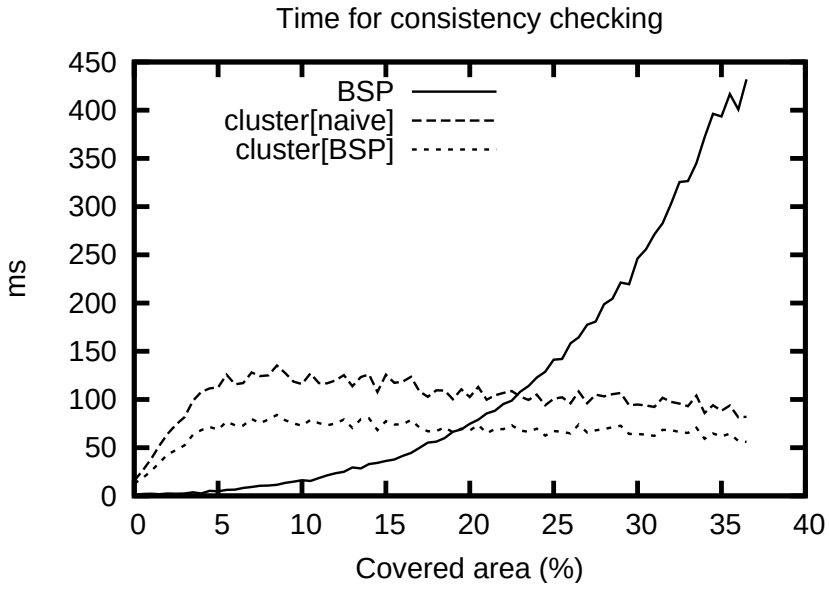


Figure 4.5: Time taken for consistency checking related to covered area with synthetic datasets and point-based semantics.

We use randomly generated datasets in this experiment. Randomly generated atoms have an average size of 10×10, a minimum size of 1×1 and a maximum size of 20×20 (width and length are not related). The position of an atom is also chosen randomly over $\mathcal{L}$. Upper bounds are generated randomly between 0.5 and 1.0, while lower bounds are chosen randomly between 0 and the previously chosen upper bound. In the first consistency experiment (Figures 4.3, 4.4), we use 20 atoms and vary the size of the space. In the other experiments (Figure 4.5), the number of atoms increase with a space of 500×500.

We use the name `naive` to refer to uses of the naive linear program based on LP (Definition 16). We use the name `BSP` to refer to uses of the BSP-based linear program PLC (Definition 18). We use `cluster[naive]` and `cluster[BSP]` to refer to uses of the clustering technique in Algorithm 3 respectively using naive linear constraints and constraints based on PLC to compute ℓ_i and u_i

Fig. 4.3 shows that the naive algorithm for consistency checking takes an amount of time that is super-linear with respect to the size of one side on the figure. All methods developed in this paper take a very small amount of time compared to it (all other curves hug the X-axis at this scale), except on a very small space.

In the case of very small spaces, Figure 4.3 shows that methods based on the BSP clustering produces poor behavior. The reason is that in these cases, the BSP tree is very deep, reducing dramatically the performance of this algorithm.

Fig. 4.4 shows that the behavior of all algorithms except for the naive one are independent of the size of the space, even for very large spaces.

In order to explore this behavior, we conducted another set of experiments to determine the behavior of these algorithms when varying the number of atoms from 0 to 1000 per (id, t) pair for a space size of 500 by 500, and measured the time used depending on the percentage of area covered by the atoms. Fig. 4.5 shows clearly that BSP without clustering rapidly worsens when the covered area increases because this implies an increased overlap between the areas covered by the atoms in the BSP. This graph also shows that the clustering technique limits the bad behavior for these difficult cases.

4.1.5 Comments on Time-Point Consistency Checking

In this section I have described three methods for checking consistency of SPOT databases under the time point semantics. The first method uses a naïve linear program lifted almost directly from the semantics itself. While the method runs in time polynomial in the number of atoms, we will expect it to perform poorly due to its dependence on the size of space $\mathcal{L}$ – a value that is sometimes quite large in practice.

The second method I detailed partitions $\mathcal{L}$ according to $\mathcal{S}^{id,t}$ such that only those

points governed by exactly the same SPOT atoms are in the same partitions. By using variables representing partitions rather than variables representing points in space for the linear programs, I was able to achieve substantially smaller linear programs. Further, since there can be at most one partition per member of $\mathcal{L}$, in the worst case this partitioning scheme produces linear programs as large as the naïve method.

The third method I describe partitions the database itself into semi-independent subsets. By computing bounds for the subsets via linear programming and then computing if those bounds are compatible globally, I hope to reduce the overall running time by reducing the size of the linear programs. Linear programming is known to run in time $O(n^3)$. Therefore, if we chop the linear program into k pieces, solve each one of them and check that the solutions are compatible, we'll have a running time of $O(k \cdot (\frac{n}{k})^3 + k^3)$. This is the intuition behind the potential for clustering to provide faster algorithms: it is a divide-and-conquer technique. Of note is the fact that clustering can be used along with either naïve consistency checking or BSP consistency checking. It does not depend on a particular type of underlying linear program.

The techniques in this section were developed in [40, 38].

4.2 World-based Semantics

4.2.1 World-based Naïve Linear Program

In this section, we start by observing that we can also check consistency of SPOT database under the world-based semantics by solving a set of linear constraints. Recall that $\mathcal{W}$ is the set of worlds and $\mathcal{I}^{\mathcal{W}}$ is the set of probability distributions over worlds. For each world w, let v_w be a variable representing $I(w)$.

Definition 23 (WLP constraints for a SPOT atom)**.** *For atom* $a = (id, r, t, [\ell, u])$*, let* $\mathbf{WLP}(a)$ *be the set of inequalities:*

1. $\sum_{w \in \mathcal{W}} \mathsf{v}_w = 1$,
2. *For all* $w \in \mathcal{W}$, $0 \leq \mathsf{v}_w \leq 1$.
3. $\sum_{w \in \mathcal{W}, w(id,t)=L} \mathsf{v}_w \geq \ell$,
4. $\sum_{w \in \mathcal{W}, w(id,t)=L} \mathsf{v}_w \leq u$,

If $\mathcal{S}$ *is a* SPOT *database, we set* $\mathbf{WLP}(\mathcal{S}) = \bigcup_{a \in \mathcal{S}} \mathbf{WLP}(a)$.

To see how these constraints work, let I be a world-based interpretation defined such that $I(w) = v_w$ for any solution to the constraints. The first and second constraints force any solution to be a proper probability distribution. The third forces the sum of the probabilities of the worlds in which a given vehicle id is at location L at time t to be at least ℓ and the fourth forces them to be at most u. The following result give us connections between consistency of a SPOT theory, and the above set of constraints.

Proposition 3. *A* SPOT *theory* $\mathcal{S}$ *is consistent under the world-based semantics iff* $\mathbf{WLP}(\mathcal{S})$ *is solvable.*

Proof. Consider any solution to $\mathbf{WLP}(\mathcal{S})$ and create an interpretation I where $I(w) = v_w$. I satisfies $\mathcal{S}$. Now consider any interpretation satisfying $\mathcal{S}$. We can create a solution satisfying $\mathbf{WLP}(\mathcal{S})$ by assigning $v_w = I(w)$. □

An obvious problem with the above result is that the size of the input to the linear program for $\mathbf{WLP}(\mathcal{S})$ is on the order of $|\mathcal{L}|^{|T| \cdot |ID|} \times |\mathcal{S}|$. This is too large for the

above algorithms to tractably solve any reasonably sized problem. One may wonder whether consistency checking for SPOT databases is NP-hard. It is not, as we will shortly see.

4.2.2 Partial Path Probabilities

$\mathbf{WLP}(\mathcal{S})$ associates a variable in the linear program with each world. Instead, one might associate a variable $p[id, t, L, L']$ denoting the probability that a vehicle with ID id travels from L to L' leaving at time t. We call this a *path probability variable*. It is clear that as long as we only look at a bounded time horizon, the number of path probability variables is polynomial with respect to the number of time points, the size of $\mathcal{L}$ and the number of vehicles. We now reformulate $\mathbf{WLP}(\mathcal{S})$ in terms of these variables so that the resulting set of constraints is polynomial in the size of the SPOT database.

Definition 24 (Interpretation Compatibility)**.** *Given $p[id, t, L, L']$ defined for every id, t, L, L' and interpretation I, we say I is compatible with p iff*

$$p[id, t, L, L'] = \sum_{w(id,t)=L, w(id,t+1)=L'} I(w)$$

Theorem 4. *Suppose θ is an assignment to all path probability variables. There is an interpretation I compatible with θ iff p satisfies*

1. *For each $t \in T, id \in ID$, $\sum_{L \in \mathcal{L}} \sum_{L' \in \mathcal{L}} p_\theta[id, t, L, L'] = 1$.*
2. *For each $t \in T, id \in ID, L, L' \in \mathcal{L}$ $p_\theta[id, t, L, L'] \geq 0$.*
3. *For $id \in ID$ and $L, L' \in \mathcal{L}$, $\neg reachable(id, L, L') \rightarrow \forall t, p_\theta[id, t, L, L'] = 0$.*
4. *For each $t, t+1 \in T, id \in ID, L \in \mathcal{L}$,*
   $\sum_{L' \in \mathcal{L}} p_\theta[id, t, L', L] = \sum_{L' \in \mathcal{L}} p_\theta[id, t+1, L, L']$.

Proof. ($\Leftarrow$): Let θ be a solution satisfying the given constraints. To construct a satisfying interpretation I, let $\alpha[id, L]$ be the probability that id is at L at the first time point, time 0. This can be computed from θ as follows: $\alpha[id, L] = \sum_{L' \in \mathcal{L}} p_\theta[id, 0, L, L']$. Now define $\delta[id, t, L, L']$ to be the probability of moving from L to L' at time t, or:

$$\delta[id, t, L, L'] = \frac{p_\theta[id, t, L, L']}{\sum_{L'' \in \mathcal{L}} p_\theta[id, t, L, L'']}$$

(when $\sum_{L'' \in \mathcal{L}} p_\theta[id, t, L, L''] = 0$, δ is defined to be 0 as well). We can now define I for all $w \in \mathcal{W}$ as:

$$I(w) = \prod_{id \in ID} \alpha[id, w(id, 0)] \prod_{t, t+1 \in T} \delta[id, t, w(id, t), w(id, t+1)]$$

To show that I is a valid interpretation, it suffices to show that $\sum_{w \in \mathcal{W}} I(w) = 1$.

We first define a bit of notation. Let $L_0, \cdots, L_{n_t}$ be a sequence of locations, and let $\mathcal{L}(w, id)$ be the sequence of locations defined by w: $w(id, t) = L_t$, and let $\mathcal{W}(id)$ be the set of all sequences of locations: $\mathcal{W}(id) = \{\mathcal{L}(w, id) \mid w \in \mathcal{S}\}$. Now we have the following algebra (explained below):

$$\sum_{w \in \mathcal{W}} I(w) \tag{4.4}$$

$$= \sum_{w \in \mathcal{W}} \prod_{id \in ID} \alpha[id, w(id, 0)] \prod_{t, t+1 \in T} \delta[id, t, w(id, t), w(id, t+1)] \tag{4.5}$$

$$= \prod_{id \in ID} \left(\sum_{L_0, \cdots, L_{n_t} \in \mathcal{W}(id)} \alpha[id, L_0] \prod_{t, t+1 \in T} \delta[id, t, L_t, L_{t+1}] \right) \tag{4.6}$$

$$= \prod_{id \in ID} \left(\sum_{L_0, \cdots, L_{n_t} \in \mathcal{L}^{|T|}} \alpha[id, L_0] \prod_{t, t+1 \in T} \delta[id, t, L_t, L_{t+1}] \right) \tag{4.7}$$

$$= \prod_{id \in ID} \left(\sum_{L_0 \in \mathcal{L}} \sum_{L_1 \in \mathcal{L}} \cdots \sum_{L_{n_t} \in \mathcal{L}} \alpha[id, L_0] \prod_{t,t+1 \in T} \delta[id, t, L_t, L_{t+1}] \right) \tag{4.8}$$

$$= \prod_{id \in ID} \left(\sum_{L_0 \in \mathcal{L}} \alpha[id, L_0] \sum_{L_1 \in \mathcal{L}} \delta[id, 0, L_0, L_1] \cdots \sum_{L_{n_t} \in \mathcal{L}} \delta[id, n_t, L_{n_t-1}, L_{n_t}] \right) \tag{4.9}$$

$$= \prod_{id \in ID} 1 = 1 \tag{4.10}$$

Starting with equation 4.4, we first substitute in the defined meaning of $I(w)$ to get equation 4.5. From here, we rearrange the sum and product to accommodate summing over $\mathcal{W}(id)$ rather than $\mathcal{W}$, leaving out the intermediate steps which convert $w \in \mathcal{W}$ into a collection of sequences, one for each id, then recombines them in such a way that the product is on the outside rather than the inside. This gives equation 4.6, which, due to the constraint $v_{id,t,L,L'} = 0$ if $\neg reachable(id, L, L')$, can be transformed to equation 4.7. By distributing the sum in equation 4.8, we can factor the terms in the product, leaving us with 4.9. Since, $\sum_{L' \in \mathcal{L}} \delta[id, t, L, L'] = 1$ for any id, t and L, we can substitute the rightmost sum with 1 continuously until all sums have been reduced to 1, giving equation 4.10.

($\Rightarrow$): Let I be an interpretation compatible with p_θ. We first show that p_θ satisfies the third constraint. Consider L, L', id such that $\neg reachable(id, L, L')$. $p_\theta[id, t, L, L'] = \sum_{w(id,t)=L, w(id,t+1)=L'} I(w)$, but since there are no worlds $w \in \mathcal{W}$ for which $w(id, t) = L$ and $w(id, t+1) = L'$ (because $\neg rechable(id, L, L')$) this sum is 0.

To show that p_θ satisfies the first constraint, pick $id \in ID$ and $t \in T$. Consider $\sum_{L \in \mathcal{L}} \sum_{L' \in \mathcal{L}} p_\theta[id, t, L, L']$. Since we already know p_θ satisfies the third constraint, this is equal to $\sum_{L \in \mathcal{L}} \sum_{L' \in \mathcal{L}} \sum_{w(id,t)=L, w(id,t+1)=L'} I(w)$ which is 1.

That p_θ satisfies the second constraint is trivial.

That p_θ satisfies the fourth constraint can be seen from the following for any given

id, t, L:

$$\begin{aligned}
\sum_{L' \in \mathcal{L}} p_\theta[id, t, L', L] &= \sum_{L' in \mathcal{L}} \left(\sum_{w(id,t)=L, w(id,t+1)=L'} I(w) \right) \\
&= \left(\sum_{w(id,t)=L} I(w) \right) \\
&= \sum_{L' \in \mathcal{L}} \left(\sum_{w(id,t-1)=L', w(id,t)=L'} I(w) \right) \\
&= \sum_{L' \in \mathcal{L}} p_\theta[id, t-1, L', L]
\end{aligned}$$

□

The above theorem provides us the ammunition needed to associate a new set of linear constraints with a SPOT database $\mathcal{S}$. Our variables for this linear program will correspond to each path probability: $\mathsf{v}_{id,t,L,L'}$.

Definition 25 ($\mathbf{PWLP}$)**.** *For* SPOT *database* $\mathcal{S}$*,* $\mathbf{PWLP}(\mathcal{S})$ *is the associated set of partial path based linear equations. Without loss of generality we assume the maximum time point* T *to be larger than any time point mentioned in* $\mathcal{S}$*.*

1. *Let* $\mathbf{PWLP}(\cdot)$ *be the constraints obtained by replacing* $p_\theta[id, t, L, L']$ *with* $\mathsf{v}_{id,t,L,L'}$ *in all constraints of Theorem 4.*

2. *For* SPOT *atom* $a = (id, r, t, [\ell, u])$*, let* $\mathbf{PWLP}(a)$ *contain*

   - $\sum_{L \in r} \sum_{L' \in \mathcal{L}} \mathsf{v}_{id,t,L,L'} \geq \ell$
   - $\sum_{L \in r} \sum_{L' \in \mathcal{L}} \mathsf{v}_{id,t,L,L'} \leq u$

For SPOT *database* $\mathcal{S}$*,* $\mathbf{PWLP}(\mathcal{S}) = \mathbf{PWLP}(\cdot) \cup \bigcup_{a \in \mathcal{S}} \mathbf{PWLP}(a)$.

The following proposition shows that solveability of $\mathbf{PWLP}(\mathcal{S})$ determines consistency of $\mathcal{S}$. The importance of this theorem will lie in the fact that $\mathbf{PWLP}(\mathcal{S})$, apart from determining consistency under world-based semantics, has size polynomial in $\mathcal{S}$.

Proposition 4. *For* SPOT *database* $\mathcal{S}$ $\mathbf{PWLP}(\mathcal{S})$ *has a solution iff* $\mathcal{S}$ *is consistent. Further, consistency is determined in time polynomial in the size of* $\mathcal{S}$.

Proof. By theorem 4, we know there is an interpretation I compatible with any solution to $\mathbf{PWLP}(\mathcal{S})$, so all we need to do is show that there is an interpretation satisfying $\mathcal{S}$ iff there is a solution to $\mathbf{PWLP}(\mathcal{S})$.

($\Leftarrow$): Suppose we have interpretation I satisfying $\mathcal{S}$. Construct solution to $\mathbf{PWLP}(\mathcal{S})$ p_θ as follows:

$$p_\theta[id, t, L, L'] = \sum_{w(id,t)=L, w(id,t+1)=L'} I(w).$$

Since I satisfies $\mathcal{S}$, we know that for all $sa = (id, r, t, [\ell, u]) \in \mathcal{S}$,

$$\sum_{w(id,t)\in r} I(w) = \sum_{L\in\mathcal{L}} \left(\sum_{w(id,t)\in r, w(id,t-1)=L} I(w) \right) \tag{4.11}$$

$$= \sum_{L\in\mathcal{L}} \left(\sum_{w(id,t)\in r, w(id,t-1)=L} \prod_{id'\in ID} \alpha[id', w(id,0)] \right. \tag{4.12}$$

$$\left. \times \prod_{t,t+1\in T} \delta[id, t, w(id', t), w(id', t+1)] \right) \tag{4.13}$$

By applying the same simplification techniques as above, we have:

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_0 \in \mathcal{L}} \alpha[id, L_0] \sum_{L_1 \in \mathcal{L}} \delta[id, 0, L_0, L_1] \cdots \right. \tag{4.14}$$

$$\left(\delta[id, t-2, L_{t-2}, L] \sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \tag{4.15}$$

$$\left. \cdots \sum_{L_{n_t} \in \mathcal{L}} \delta[id, n_t - 1, L_{n_t - 1}, L_{n_t}] \right) \times \prod_{id' \in ID \setminus \{id\}} \gamma \tag{4.16}$$

Where $\prod_{id' \in ID \setminus \{id\}} \gamma$ is like equation 4.6, and we can therefore apply the same reasoning as leads to equation 4.10 to get that $\prod_{id' \in ID \setminus \{id\}} \gamma = 1$. Thus we have:

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_0 \in \mathcal{L}} \alpha[id, L_0] \sum_{L_1 \in \mathcal{L}} \delta[id, 0, L_0, L_1] \cdots \sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \right. \tag{4.17}$$

$$\left. \left(\delta[id, t-2, L_{t-2}, L] \sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \right) \tag{4.18}$$

We can then re-combine and reverse the simplification.

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_0, \cdots, L_{t-2} \in \mathcal{L}^{t-1}} \alpha[id, L_0] \prod_{0 \leq t' < t-2} \delta[id, t', L_{t'}, L_{t'+1}] \right. \tag{4.19}$$

$$\left. \left(\delta[id, t-2, L_{t-2}, L] \sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \right) \tag{4.20}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_{t-2} \in \mathcal{L}} \sum_{L_{t-3} \in \mathcal{L}} \cdots \sum_{L_0 \in \mathcal{L}} \alpha[id, L_0] \prod_{0 \leq t' < t-2} \delta[id, t', L_{t'}, L_{t'+1}] \right. \tag{4.21}$$

$$\left. \left(\delta[id, t-2, L_{t-2}, L] \sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \right) \tag{4.22}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \tag{4.23}$$

$$\left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \right. \tag{4.24}$$

$$\left. \cdots \sum_{L_1 \in \mathcal{L}} \delta[id, 1, L_1, L_2] \sum_{L_0 \in \mathcal{L}} \delta[id, 0, L_0, L_1] \alpha[id, L_0] \right) \tag{4.25}$$

Now we substitute back for δ and α and are able to simplify out each summation (starting with the last).

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \tag{4.26}$$

$$\left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \right. \tag{4.27}$$

$$\left. \cdots \sum_{L_1 \in \mathcal{L}} \delta[id, 1, L_1, L_2] \sum_{L_0 \in \mathcal{L}} \frac{p_\theta[id, 0, L_0, L_1]}{\sum_{L'' \in \mathcal{L}} p_\theta[id, 0, L_0, L'']} \sum_{L' \in \mathcal{L}} p_\theta[id, 0, L_0, L] \right) \tag{4.28}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \tag{4.29}$$

$$\left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \right. \tag{4.30}$$

$$\left. \cdots \sum_{L_1 \in \mathcal{L}} \delta[id, 1, L_1, L_2] \sum_{L_0 \in \mathcal{L}} v_{id,0,L_0,L_1} \right) \tag{4.31}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \tag{4.32}$$

$$\left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \right. \tag{4.33}$$

$$\left. \cdots \sum_{L_1 \in \mathcal{L}} \frac{v_{id,1,L_1,L_2}}{\sum_{L'' \in \mathcal{L}} v_{id,1,L_1,L''}} \sum_{L_0 \in \mathcal{L}} v_{id,1,L_1,L_0} \right) \tag{4.34}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \right. \tag{4.35}$$

$$\left. \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \cdots \sum_{L_1 \in \mathcal{L}} p_\theta[id, 1, L_1, L_2] \right) \tag{4.36}$$

$$= \sum_{L \in \mathcal{L}} \left(\sum_{L_t \in r} \delta[id, t-1, L, L_t] \right) \left(\sum_{L_{t-2} \in \mathcal{L}} \delta[id, t-2, L_{t-2}, L] \right. \tag{4.37}$$

$$\left. \sum_{L_{t-3} \in \mathcal{L}} \delta[id, t-3, L_{t-3}, L_{t-2}] \cdots \sum_{L_1 \in \mathcal{L}} p_\theta[id, 2, L_2, L_1] \right) \tag{4.38}$$

Continuing in this fashion, we can keep eliminating summations until we get:

$$\sum_{w \in \mathcal{W}} I(w) = \sum_{L \in \mathcal{L}} \sum_{L_t \in r} p_\theta[id, t-1, L, L_t] \tag{4.39}$$

Thus p_θ satisfies all constraints pertaining to SPOT atoms. That p_θ satisfies the other constraints follows from Theorem 4.

($\Leftarrow$): Suppose we have a solution θ to $\mathbf{PWLP}(\mathcal{S})$ defining p_θ. We know, by theorem 4, that there exists an interpretation I compatible with p_θ. To show that I satsifies $\mathcal{S}$, we need only show that for any $(id, r, t, [\ell, u]) \in \mathcal{S}$, that I satisfies $(id, r, t, [\ell, u])$. This follows by following the reasoning for equations 4.13 through 4.39 in reverse order. □

PWLP's Size

$\mathbf{PWLP}(\mathcal{S})$ is significantly smaller than that of $\mathbf{WLP}(\mathcal{S})$ because it only contains $|ID| \cdot |T| \cdot |\mathcal{L}|^2$ variables and

$$|T| \cdot |ID| + |T| \cdot |\mathcal{L}|^2 + |T| \cdot |ID| \cdot |\mathcal{L}| + |\mathcal{S}|$$

equations. The size of this linear program is therefore bounded by $O(|ID|^2 \cdot |T|^2 \cdot |\mathcal{L}|^4 \times |\mathcal{S}|)$. In contrast, $\mathbf{WLP}(\mathcal{S})$ had a size of $|\mathcal{L}|^{|T| \cdot |ID|} \times |\mathcal{G}|$. Because linear programs are solvable in polynomial time, this makes the determination of consistency under the world-based semantics achievable in polynomial time.

Furthermore, it turns out that there are alternate ways of expressing $\mathbf{PWLP}(\mathcal{S})$ which result in more easily solvable linear programs.

Variable Pruning

The first simplification we can make to $\mathbf{PWLP}(\mathcal{G})$ or any set of linear equations comes when we know $\mathsf{v}_{id,t,L,L'}$ to be zero due to the reachability definition. In such cases, we can safely eliminate $\mathsf{v}_{id,t,L,L'}$ from $\mathbf{PWLP}(\mathcal{G})$.

4.2.3 An Alternative Linear Program: AWLP

Many SPOT databases only mention some time points from the set of all time points T. This can be leveraged to create a smaller linear program that does not reference the unmentioned time points. In this linear program we only use the time points $t_1, \cdots, t_n$ mentioned in the databases. The meaning of the variable $\bar{\mathsf{v}}_{t_i,L,L'}$ then changes to the probability that object id is at L at time point t_i and at L' at time point t_{i+1}. In this way, we no longer need to include any $\bar{\mathsf{v}}_{t,L,L'}$ unless $t_i = t$. This results in a smaller linear program.

Definition 26 (Alternate Linear Program)**.** *Let $\mathcal{S}$ be a set of* SPOT *atoms and id be a vehicle. Let $t_1 < t_2 \cdots < t_n$ be all the time points such that for every* SPOT *atom $(id, r, t, [\ell, u]) \in \mathcal{S}$ there is $t_i = t$. Without loss of generality, we can assume t_n is not the maximum time point (i.e. there is $t' \in T$ such that $t' > t_n$ which we call t_{n+1}). $\mathbf{AWLP}(\mathcal{S}, id)$ is the following alternate set of linear equations:*

1. *For each $a = (id, r, t_i, [\ell, u]) \in \mathcal{S}$:*
   - *(a)* $\sum_{L \in r} \sum_{L' \in \mathcal{L}} \bar{\mathsf{v}}_{t,L,L'} \geq \ell \in \mathbf{AWLP}(\mathcal{S}, id)$
   - *(b)* $\sum_{L \in r} \sum_{L' \in \mathcal{L}} \bar{\mathsf{v}}_{t,L,L'} \leq u \in \mathbf{AWLP}(\mathcal{S}, id)$
2. *For each $t_1 \leq t_i \leq t_n$,* $\sum_{L \in \mathcal{L}} \sum_{L' \in \mathcal{L}} \bar{\mathsf{v}}_{t_i,L,L'} = 1 \in \mathbf{AWLP}(\mathcal{S}, id)$

3. *For each* $t_1 \leq t_i \leq t_n$*, and for each L,L' if* $\neg reachable(t_{i+1} - t_i, L, L')$ *then* $\mathsf{v}_{t_i,L,L'} = 0 \in \mathbf{AWLP}(\mathcal{S}, id)$.

4. *For each L,L' if* $\neg reachable(id, L, L')$ *then* $\bar{\mathsf{v}}_{t_n,L,L'} = 0 \in \mathbf{AWLP}(\mathcal{S}, id)$.

5. *For each* $t_1 < t_i \leq t_n$*, for each L we have* $\sum_{L' \in \mathcal{L}} \bar{\mathsf{v}}_{t_{i-1},L',L} - \sum_{L' \in \mathcal{L}} \bar{\mathsf{v}}_{t_i,L,L'} = 0 \in \mathbf{AWLP}(\mathcal{S}, id)$

Theorem 5. $\mathbf{PWLP}(\mathcal{S})$ *is solvable iff* $\mathbf{AWLP}(\mathcal{S}, id)$ *is solvable for every* id*.*

Proof. Because of its similarity to earlier proofs, this proof is presented as a sketch. ($\Rightarrow$): Let θ be a solution to $\mathbf{PWLP}(\mathcal{S})$ and let $\bar{I}$ be a world-based interpretation compatible with the solution θ. Consider $\mathbf{AWLP}(\mathcal{S}, id)$ for some $id \in ID$. Construct a solution $\bar{\theta}$ to $\mathbf{AWLP}(\mathcal{S}, id)$ as: $\mathsf{v}_{t_i,L,L'}\bar{\theta} = \sum_{w(id,t_i)=L \wedge w(id,t_{i+1})=L'} \bar{I}(w)$. The rest follows from algebra similar to that in the proofs of Theorem 4 and Proposition 4. ($\Leftarrow$): Let θ_{id} be a solution to $\mathbf{AWLP}(\mathcal{S}, id)$. Construct a solution θ to $\mathbf{PWLP}(\mathcal{S})$ in the following manner: let interpretation I be compatible with all θ_{id} satisfying the following constraints for all id, t_i, L, L': $\sum_{w(id,t_i)=L, w(id,t_{i+1}=L'} I(w) = \mathsf{v}_{t_i,L,L'}\theta_{id}$. The constraints in the linear program guarantee the existence of such an I (this again can be established via algebra similar to that in the proof of Theorem 4). Further, the existence of this interpretation implies a compatible assignment θ to the variables $\mathsf{v}_{id,t,L,L'}$ (due to Theorem 4). □

This theorem provides us with added efficiency in two ways. First, it has significantly fewer variables. Second, it divides the linear program into $|ID|$ linear programs, one for each vehicle. When the entire linear program is considered, the running time is $O(r^3)$, where r is the number of variables. But, if we consider each vehicle individually, the running time is proportional to $|ID| \cdot O((r/|ID|)^3)$, giving a

speedup of $O(|ID|^2)$. We can further exploit this trick of dividing the linear program into smaller sub-problems by considering complete points in a SPOT database.

A complete time point for a vehicle is where there is only one potential probability distribution over the locations in $\mathcal{L}$ for a given object id at a given time t.

Definition 27 (Complete Time Point)**.** SPOT *database $\mathcal{S}^{id,t}$ is complete if there is a subset $S' \subset \mathcal{S}^{id,t}$, such that*

- *For all $(id, r_1, t, [\ell_1, u_1]), (id, r_2, t, [\ell_2, u_2]) \in S'$, r_1 is disjoint from r_2.*
- *For all $(id, r, t, [\ell, u]) \in S'$, $r = \{L\}$ for some $L \in \mathcal{L}$.*
- $\sum_{(id,r,t,[\ell,u])\in S'} \ell = 1$.

Proposition 5 (Complete Time Point)**.** *Given a* SPOT *database $\mathcal{S}$, if a time point $t \in T$ is* complete *for a vehicle $id \in ID$ then there are values $\alpha[L]$ for each $L \in \mathcal{L}$ such that:*

$$\forall I \in \mathcal{I}(\mathcal{S}), \sum_{w(id,t)=L} I_1(w) = \alpha[L]$$

Example 19. *Let space contain the points L_1 and L_2, and let the reachability predicate always be true. Consider $\mathcal{K} = \{(id, \{L_1\}, 0, [1, 1]), (id, \{L_2\}, 1, [0.4, 0.5])\}$. For id, time point 0 is complete because all interpretations must assign a probability of 1 to id being at L_1 at time 0. However, time point 1 is not complete, as the probability of being at L_2 could be anything between 0.4 and 0.5.*

Completeness ends up being very useful for dividing the database in half and dealing with each half independently. Consider the following theorem. In this theorem we use the notation $\mathcal{S}^{id,t_1,t_2}$ to represent $\cup_{t_1\leq t\leq t_2}\mathcal{S}^{id,t}$.

Theorem 6. *For* SPOT *database $\mathcal{S}$ and vehicle id complete for id at time t, let t_n be the max time point referenced by the database,*

1. $\mathbf{AWLP}(\mathcal{S}, id)$ *is solvable iff* $\mathbf{AWLP}(\mathcal{S}^{id,0,t}, id)$ *and* $\mathbf{AWLP}(\mathcal{S}^{id,t,t_n}, id)$ *are solvable.*

2. *The solution to* $\mathbf{AWLP}(\mathcal{S}^{id,0,t}, id)$ *and* $\mathbf{AWLP}(\mathcal{S}^{id,t,t_n}, id)$ *gives a solution for* $\mathbf{AWLP}(\mathcal{S}, id)$.

Proof. First we show (2), which will also suffice for showing ($\Rightarrow$) of (1).

(2): Let $v_{t,L,L'}$ be the variables for both $\mathbf{AWLP}(\mathcal{S}^{id,0,t^*}, id)$ and $\mathbf{AWLP}(\mathcal{S}^{id,t^*,t_n})$. The only constraints in $\mathbf{AWLP}(\mathcal{S}, id)$ not in $\mathbf{AWLP}(\mathcal{S}^{id,0,t^*}, id) \cup \mathbf{AWLP}(\mathcal{S}^{id,0,t^*}, id)$ are the constraints for movement through t^*, for all L

$$\sum_{L' \in \mathcal{L}} v_{t^*,L',L} = \sum_{L' \in \mathcal{L}} v_{t',L,L'} \tag{4.40}$$

(where t' is the next time point after t^* containing a SPOT atom). However, since $\mathcal{S}^{id}$ is complete at time t^*, there are values $\alpha[L]$ such that for all $L \in \mathcal{L}$, the probability of being at L at time t^* is $\alpha[L]$. Thus both the right hand side and the left hand side of equation 4.40 are $\alpha[L]$ and therefore equal.

(1) $\Leftarrow$: A solution $v_{t,L,L'}$ to $\mathbf{AWLP}(\mathcal{S}, id)$ is also a solution to both $\mathbf{AWLP}(\mathcal{S}^{id,0,t^*}, id)$ and $\mathbf{AWLP}(\mathcal{S}^{id,t^*,t_n})$, since $\mathbf{AWLP}(\mathcal{S}^{id,0,t^*}, id) \cup \mathbf{AWLP}(\mathcal{S}^{id,t^*,t_n}) \subset \mathbf{AWLP}(\mathcal{S}, id)$.

□

AWLP Size

Now, instead of solving **WLP**, a massive linear program, we are solving many smaller $\mathbf{AWLP}(\mathcal{S}^{id}_{t_1-t_2}, id)$.

For vehicle id in theory $\mathcal{S}$ with n mentioned time points of which every c^{th} one is complete, and each two consecutive time points, t_1 and t_2, we must solve $\mathbf{AWLP}(\mathcal{S}^{id}_{t_1-t_2}, id)$. This will happen in $O(n^3)$ where n is the number of variables. The number of variables is bounded by $(c+1) \cdot |\mathcal{L}|^2$. Thus computing consistency

of $\mathcal{S}$ using **AWLP** will happen in time $|ID| \times n/c \times O(((c+1) \cdot |\mathcal{L}|^2)^3)$. There is an interesting phenomenon happening here: the running time of our algorithms can *decrease* as the size of the theory grows – so long as more complete time points are added.

4.2.4 Experimental Results

Experiments were performed on a prototype implementation testing the feasibility of these linear programs. The implementation was in MatLab and run on a Pentium 4 (3.80GHz) processor running under Windows XP and with 2GB of memory. Our system implements all algorithms described in this paper for both complete and incomplete theories.

We ran several experiments to test the performance of these algorithms and identify the important factors that affect the performance other than the obvious ones such as number of atoms and time points referenced. We performed our experiments on theories that refer to a single vehicle using **AWLP** type linear programs.

The maximum number of SPOT atoms per time point in a complete theory plays an important role in checking consistency. This number gives a maximum number of path probabilities for each time point. Another important factor we considered was the maximum speed of the vehicle because this affects the maximum number of reachable locations and hence the total number of path probability variables in our final linear program. To test the effect of speed and atom density, we created random theories in a 50×50 grid. We varied the maximum number of atoms per time point from 2 to 5, with maximum speeds of 1 and 8. The number of distinct time points in the theory varied from 10 to 100. We derived the speed values as follows: suppose we use the 50×50 grid to represent the USA and one time point equals one hour. Then

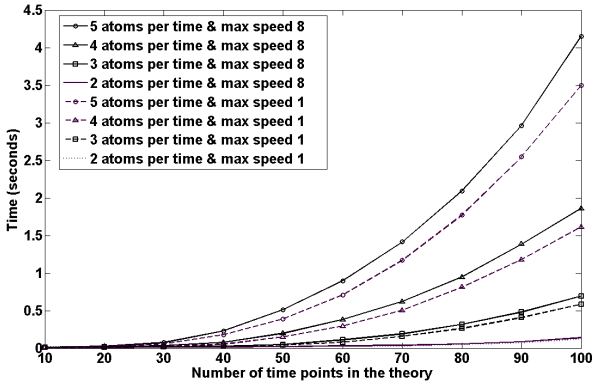


Figure 4.6: Time to check consistency of complete theories with 2 to 5 SPOT atoms per time point when maximum speed is 1 and 8.

a speed of 1 will coincide with that of a car (70 mph) while a speed of 8 will coincide with that of a plane (500 mph).

Consistency check time for complete theories: Figure 4.6 shows the time taken for consistency checking for 8 kinds of complete theories. The data points represent the average over 50 randomly generated theories. As seen in the figure the effect of increasing number of SPOT atoms per time point has a greater on impact on performance than increasing maximum speed. The reader can see that it only takes a few seconds to reason about 100 time points.

Consistency check time for incomplete theories: For incomplete theories, the size of the grid has a great impact on the time required to check consistency. We say a time point t is *incomplete* iff it is not complete. We investigated how the structure of the theory affects the consistency checking algorithm. Let $Incmax$ be the maximal number of incomplete time points followed by a complete time point in a theory. For this experiment we created random theories in a 10 by 10 grid with maximum speed 1, maximum number of SPOT atoms per complete time point ranging from 2 to 4, $Incmax = 1$ or $Incmax = 2$ and the number of referenced time points ranging from

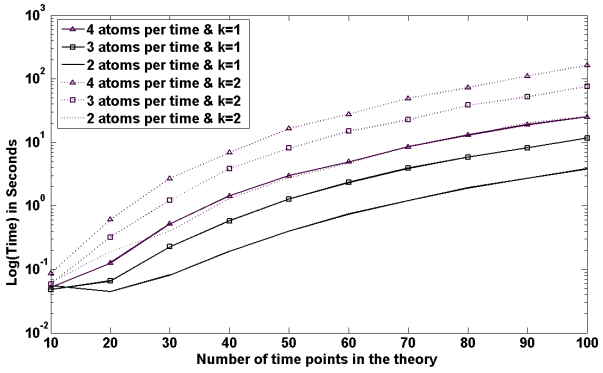


Figure 4.7: Time to check consistency of incomplete theories with 2, 3, and 4 SPOT atoms per time point, a maximum speed of 1 or 8, and $Incmax$ varying from 1 to 2 (displayed as k in the graph's key).

10 to 100. Furthermore every complete time point is followed by $Incmax$ incomplete time points in the theory. So when $Incmax = 1$ and the total number of time points in the theory is 50, there are 25 incomplete time points interleaved with 25 complete points. Figure 4.7 shows the time taken to check consistency for 6 kinds of randomly generated theories. The data points represent the average of 20 runs — note that the y-axis uses a logarithmic scale. As seen in the figure, the effect of increasing the number of SPOT atoms per time point has a similar effect on the on the performance. However, increasing $Incmax$ affects the running time dramatically.

Complete-queries. Figure 4.8 shows the time required to check consistency of a random atom against complete databases. Temporal density of a theory is the ratio of time points referenced in the theory and the total number of time points. For these experiments we set the grid size to 25 by 25 and maximum time points to 500 and number of SPOT atoms per time point to 3. For example when the theory has a temporal density of 1, it has a total of 1500 time points. The data points in the graph are an average of 100 runs. The reader can easily see that the time taken drops exponentially

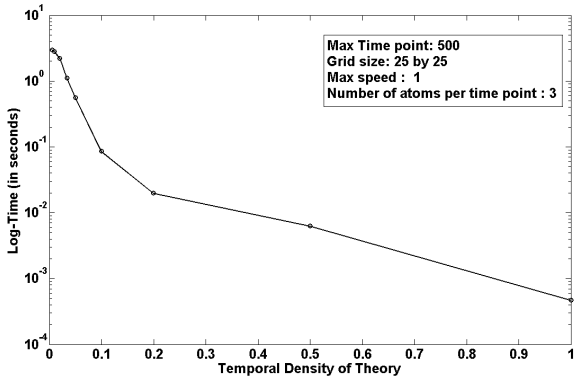


Figure 4.8: Time to answer in-queries w.r.t. complete theories of varying temporal density.

with an increase in the density. Since a rise in density corresponds to an increase in theory size, these results are particularly interesting yet consistent with Theorem 6. It shows our algorithms' running time *decreases* as the number of atoms in the database increases. This is sensible: when one is working with probabilistic data, one should sometimes find it easier to answer queries as the amount of data increases, because fewer possible satisfying interpretations for the data need be considered.

4.2.5 Comments on World-Based Consistency Checking

World-based consistency checking is understandably less efficient than point-based consistency checking – world-based consistency checking accounts for reachability constraints while point-based consistency checking does not. Given this, it is surprising that we were able to find a polynomial time consistency checking algorithm. The existence of such a linear programing based algorithm, and the progressively more efficient variations on the linear program: from LP to $\mathbf{PWLP}$ to $\mathbf{AWLP}$ bring the use of these sorts of consistency checking algorithms into the realm of the potentially

practical. One can easily imagine a large database being checked periodically (nightly or weekly) for consistency. Later, in Chapter 5, I will introduce some methods which can even automatically repair these databases when they are found inconsistent.

A unique quality of these algorithms is the relationship between running time and input size. With the right kind of input, **AWLP** actually decreases in running time as the size of the database increases (see Figure 4.8). While for traditional database theory this is counter-intuitive, it makes sense under probabilistic considerations. More probabilistic data (of the right kind) should rule out possibilities, making it unnecessary for the algorithms to check as many different things, and resulting in a runtime speedup.

The key to the efficiency of these algorithms is the re-formulation of the problem from a naïve one which assigns a variable to each world to one which assigns a probability to an object's moving from one location to another. For the purposes of SPOT databases, this section shows the two approaches to be equivalent.

Chapter 5

SPOT Database Revision

5.1 Introduction and Motivation

In deployment, we expect SPOT databases to seamlessly handle often changing and sometimes inconsistent data. They will need to properly integrate newly created probabilistic spatio-temporal data with existing information, even when the new information creates an inconsistency. Consider the following example:

Example 20. *Imagine a spatio-temporal database specifying the locations of tagged wildlife. With some regularity, a tagged animal will be spotted in a given region at a given time. These sightings are generally not guaranteed – people sometimes see things that are not there – so this information is properly characterized as probabilistic spatio-temporal.*

The GPS signal from the tag goes in and out and is subject to error. The research team receives one signal at noon saying the animal is in a region R1 covered in forest, producing the SPOT *atom* $(animal, R1, noon, [0.7, 1.0])$. *However, several members of another team spot the same animal at noon in the region R2, which is over 10 miles away, producing the atom* $(animal, R2, noon, [1.0, 1.0])$. *When the other team adds*

their data to the database, it will be inconsistent with the GPS-derived data. With an automatic database revision procedure, this inconsistency can be repaired seamlessly as the database is updated, either by throwing the data $(animal, R1, noon, [0.7, 1.0])$ *out or by some other means.*

The algorithms in this chapter will show how to integrate the new information with past information even when the new information is inconsistent with the previous knowledge.

We justify our algorithms' correctness as proper revisions by appealing to the AGM axioms[1]. These axioms are recognized to be a minimal standard which must be met for any type of revision of a knowledge base.

5.2 AGM Axioms

We now present AGM-style postulates [1] for revising SPOT databases. A revision operator $\dot{+}$ is a binary function that takes a SPOT database and a SPOT atom as input, and produces a SPOT database as output. $\dot{+}$ is required to satisfy the AGM axioms [1] expressed in our framework. In the following, $\mathcal{S}$ is a SPOT database and a is a SPOT atom.

(A1) $\mathcal{S} \dot{+} a$ is SPOT database.

(A2) $\mathcal{S} \dot{+} a \models a$.

(A3) $(\mathcal{S} \cup \{a\}) \models (\mathcal{S} \dot{+} a)$.

(A4) If a is consistent with $\mathcal{S}$ then $(\mathcal{S} \dot{+} a) \models (\mathcal{S} \cup \{a\})$.

[1] As SPOT databases are atomic, we do not discuss AGM axioms involving negation and disjunction.

(A5) if $\mathcal{S} \dot{+} a$ is inconsistent then $\{a\}$ is inconsistent.

(A6) If $a \equiv a'$ then $\mathcal{S} \dot{+} a \equiv \mathcal{S} \dot{+} a'$.

Axiom **(A2)** says that the revised knowledge (a) must be part of the updated knowledge base. **(A3)** and **(A4)** together tell us that if $\mathcal{S} \cup \{a\}$ is consistent then $\mathcal{S} \dot{+} sa$ is logically equivalent to $\mathcal{S} \cup \{sa\}$. **(A5)** says that no unwarranted inconsistency will be introduced by the revision. **(A6)** says that the syntax of the new information is irrelevant to the update.

5.3 Consistency Checking

Since much of the inconsistency in real-world data is a direct result of objects' available movement, we consider only world-based semantics (Section 3.4 on page 36) for belief revision.

We will assume consistency checking is done via the linear program $\mathbf{PWLP}(\cdot)$ (definition 25 on page 65). While potentially faster linear programming methods exist with these semantics, (for instance, $\mathbf{AWLP}(\cdot)$ of definition 26), we will eventually be modifying the linear program and the modifications are most clear when made to $\mathbf{PWLP}(\cdot)$. Once understood, these modifications can be transfered to $\mathbf{AWLP}(\cdot)$.

We note that all the mentioned methods run in polynomial time so long as time and space are bounded *a priori*.

Throughout the rest of this chapter, we assume that $\mathcal{S}$ is consistent. Inconsistencies may arise when we insert a PST-atom into $\mathcal{S}$.

5.4 Some belief revision strategies

The reader can easily see that the revision of a SPOT database $\mathcal{S}$ with a SPOT atom a may be handled in many different ways when $\mathcal{S} \cup \{a\}$ is inconsistent. We could change the t part of a SPOT atom, or the r part of a SPOT atom, or the $[\ell, u]$ part of a PST-atom, or the id part of the SPOT atom. We could also study maximal consistent subsets as in [5, 24].

5.5 Maximal Consistent Subsets

We can define a revision operator $\dot{+}_m$ based on maximal consistent subsets as follows.

Definition 28. *Suppose $\mathcal{S}$ is a* SPOT *database and a is a* SPOT *atom. Then $\mathcal{S}' \cup \{a\}$* accomplishes the revision of $\mathcal{S}$ by adding a via the subset strategy *iff $\mathcal{S}'$ is a subset of $\mathcal{S}$ and $\mathcal{S}' \cup \{a\}$ is consistent.*

$\mathcal{S}' \cup \{atm\}$ optimally accomplishes the revision of $\mathcal{S}$ by adding atm via the max-subset strategy *iff it accomplishes the revision of $\mathcal{S}$ by adding a via the subset strategy and there is no other $\mathcal{S}'' \cup \{a\}$ that accomplishes the same revision such that $\mathcal{S}' \subsetneq \mathcal{S}''$.*

We use the notation $\mathcal{S} \dot{+}_m a$ to denote a $\mathcal{S}' \cup \{a\}$ that optimally accomplishes the revision of $\mathcal{S}$ by adding a via the max-subset strategy. [2]

We verify that $\dot{+}_m$ satisfies the AGM axioms.

Proposition 6. *Any function $\dot{+}_m$ that optimally accomplishes the revision via the max-subset strategy satisfies the AGM axioms.*

[2] There is some non-determinism in this definition. A strict total ordering can be induced on all $\mathcal{S}'$ satisfying the above definition and the minimal element of the strict total ordering can be picked in order to induce determinism. **Throughout the rest of this chapter, we assume such a strict total ordering called O_T is available.**

Proof. Axioms **(A1)** to **(A4)** are straightforward. Axiom **(A5)** is straightforward because $\mathcal{S} \dot{+} a = \mathcal{S}' \cup a$, where $\mathcal{S}'$ is the largest subset of $\mathcal{S}$ consistent with a. The only way for $\mathcal{S} \dot{+} a$ to be inconsistent is that $\mathcal{S}'$ is empty and a is inconsistent by itself, and if a is inconsistent by itself, then no number of removals from $\mathcal{S}$ will make $\mathcal{S} \dot{+}_m a$ consistent. Axiom **(A6)** is verified because of the strict total ordering will always chose the same database when multiple are available. □

Unfortunately, computing $\dot{+}_m$ is intractable.

Theorem 7. *Determining if $\mathcal{S}' \cup \{a\}$ optimally accomplishes the revision of $\mathcal{S}$ with a via the max-subset strategy is coNP-complete.*

Proof. **Membership**: Suppose $\mathcal{S}'$ is not an optimal max-subset revision of $\mathcal{S}$ w.r.t. a. We know an optimal max-subset revision exists (in the extreme case, $\emptyset$ can be a max-subset revision), so we can assume $\mathcal{S}''$ is that revision. That $\mathcal{S}''$ is a max-subset revision can be verified in polynomial time: to establish that it is in fact maximal, check the consistency of $\mathcal{S}''$ with each atom from $\mathcal{S} \setminus \mathcal{S}''$. If all are inconsistent, the $\mathcal{S}''$ is a maximal subset. This is a polynomial time operation due to the polynomial time consistency checking algorithm and the fact that we only check consistency at most $|\mathcal{S}|$ times. Since we suppose that computing the relationship between $\mathcal{S}''$ and $\mathcal{S}'$ via O_T can be done in polynomial time, that $\mathcal{S}''$ is better than $\mathcal{S}'$ according to O_T is also done in polynomial times. Thus we can verify the counter-example $\mathcal{S}''$ in polynomial time.

For hardness part of this proof, we use the coNP-complete problem minimum subset sum:

Definition 29 (Minimum Subset Sum)**.** *For multiset* $S = \{s_1, \ldots, s_n\}$ *constant* c*, and* $S' \subseteq S$*,* (S, c, S') *is in* minimum subset sum *iff* $\sum_{s \in S'} s = c$ *and there is no* $S'' \subset S$ *s.t.* $|S''| < |S'|$ *and* $\sum_{s \in S''} s = c$*.*

We now prove the above problem is coNP-complete. **Proof that Minimum Subset Sum is coNP-complete**

Membership: If (S, c, S') is not in minimum subset sum then there is a witness $S'' \subseteq S$ such that $\sum_{s \in S''} s = c$ and $|S''| < |S'|$.

Hardness: Consider subset sum problem $(V = \{v_1, \ldots, v_n\}, c)$ (where V can be a multiset). Deciding if there is $V' \subset V$ such that $\sum_{v \in V'} v = c$ is NP-complete. We construct a minimum subset sum instance: Let m be such that $gcd(c, n+m) = 1$. Let $s_i = (n+m) \cdot v_i$ for $1 \leq i \leq n$, and $s_i = c$ for $n < i \leq n + (n+m)$ and S be the multiset $\{s_i | 1 \leq i \leq n + (n+m)\}$. Let S' be the multiset $\{s_i | n < i \leq n + (n+m)\}$ (S' contains $n + m$ copies of c). Notice that $S' \subset S$ and $\sum_{s \in S'} s = c \cdot (n+m)$. $(S, c \cdot (n+m), S')$ is a member of minimum subset sum iff (V, c) is not a member of subset sum.

($\Rightarrow$) Consider $(S, c \cdot (n+m), S')$ a member of minimum subset sum, and suppose (V, c) is a member of subset sum to derive a contradiction. Since (V, c) is a member of subset sum, there is $V' \subseteq V$ s.t. $|V| \leq n$ and $\sum_{v \in V'} v = c$. Construct $S'' = \{v \cdot (n+m) | v \in V'\}$. S'' is a counter-example for $(S, c \cdot (n+m), S')$ being in subset sum: $\sum_{s \in S''} s = \sum_{v \in V'} v \cdot (n+m) = c \cdot (n+m)$ and $|S''| = |V'| \leq n < |S'|$. Contradiction.

($\Leftarrow$) Consider if (V, c) is not a member of subset sum. Suppose without loss of generality that $c > 1$. Suppose $(S, c \cdot (n+m), S')$ is not a member of minimum subset sum to derive a contradiction. Thus there is $S'' \subset S$ such that $\sum_{s \in S''} s = c \cdot (n+m)$ and $|S''| < |S'|$. S'' can be divided into two multi-sets: $S''_1 = \{v_i \cdot (n+m) | i \leq n\}$ and

$S_2'' = \{s_i | i > n\}$ such that $S'' = S_1'' \cup S_2''$. Notice that S_2'' is non-empty, otherwise we have, $V' = \{v_i | v_i \cdot (n+m) \in S_1''\}$ such that $V' \subset V$ and $\sum_{v \in V'} v = c$, contradicting that (V, c) is not a member of subset sum. With the following algebra, we derive the contradiction to the original supposition that $(S, c \cdot (n+m), S')$ is not a member of minimum subset sum:

$$c \cdot (n+m) = \sum_{s_i \in S''} s_i = \sum_{s_i \in S_1''} s_i + \sum_{s_i \in S_2''} s_i = \sum_{s_i \in S_1''} v_i \cdot (n+m) + \sum_{s_i \in S_2''} c$$

$$\Rightarrow c \cdot (n+m) - (\sum_{s_i \in S_1''} v_i) \cdot (n+m) = c \cdot |S_2''|$$

Since the left hand side is divisible by $n+m$, $c \cdot (|S_2''|)$ is divisible by $n+m$. Since m was chosen such that $gcd(n+m, c) = 1$, this further implies that $|S_2''|$ is divisible by $n+m$. However, since S_1'' is non-empty, $|S_2''| < n+m$, and we therefore have that there is a number smaller than $n+m$ which is non-zero and divisible by $n+m$. Contradiction.

Hardness of optimal subset revision: Take an instance of minimum subset sum problem $(S = \{s_1, \cdots, s_n\}, c, S')$. Let $tot = \sum_{s_i \in S} s_i$. Let $\mathcal{L} = \{p_1, \cdots, p_n, p_{n+1}\}$ ($n+1$ point space) and

$$\mathcal{K} = \{(id, \{p_i\}, 0, [s_i/tot, s_i/tot] | s_i \in S\}.$$

Let $sa = (id, \{p_{n+1}\}, 0, [c/tot, c/tot])$, and let

$$\mathcal{K}' = \{(id, \{p_i\}, 0, [s_i/tot, s_i/tot] | s_i \notin S'\}.$$

Consider ordering O_T that prefers revision $\bar{\mathcal{K}}$ to $\bar{\mathcal{K}}'$ ($\bar{\mathcal{K}} < \bar{\mathcal{K}}'$) whenever

$$\sum_{(id,r,t,[\ell,u]) \in \bar{\mathcal{K}}} \ell = 1 - c/tot \text{ and } \sum_{(id,r,t,[\ell,u]) \in \bar{\mathcal{K}}'} \ell \neq 1 - c/tot.$$

If $\sum_{(id,r,t,[\ell,u]) \in \bar{\mathcal{K}}} \ell = \sum_{(id,r,t,[\ell,u]) \in \bar{\mathcal{K}}'} \ell$ then $\bar{\mathcal{K}} < \bar{\mathcal{K}}'$ if $|\bar{\mathcal{K}}| > |\bar{\mathcal{K}}'|$. That is, O_T prefers any *maximally sized* database whose atom's lower bounds sum to $1 - c/tot$. $\mathcal{K}'$ is

a minimal subset revision of $\mathcal{K}$ wrt sa iff (S, c, S') is an instance of minimal subset sum.

($\Rightarrow$): Suppose $\mathcal{K}'$ is a minimal subset revision of $\mathcal{K}$ wrt sa and (S, c, S') is not an instance of minimal subset sum to derive a contradiction. Then there is $S'' \subset S$ such that $\sum_{s \in S''} s = c$ and $|S''| < |S|$. Construct $\mathcal{K}''$:

$$\mathcal{K}' = \{(id, \{p_i\}, 0, [s_i/tot, s_i/tot] | s_i \notin S''\}.$$

$\mathcal{K}'' \cup \{a\}$ is consistent, $\sum_{(id,r,t,[\ell,u]) \in \mathcal{K}''} \ell = 1 - c/tot$ and $|\mathcal{K}''| > |\mathcal{K}'|$. Therefore $\mathcal{K}''$ is preferred by O_T over $\mathcal{K}'$ and $\mathcal{K}'$ cannot be the optimal subset revision. Contradiction.

($\Leftarrow$): Suppose $\mathcal{K}'$ is not a minimal subset revision of $\mathcal{K}$ wrt sa and (S, c, S') is an instance of minimal subset sum to derive a contradiction. Let $\mathcal{K}''$ be the counter-example showing $\mathcal{K}'$ is not a minimal subset sum. Thus there is $\mathcal{K}''$ such that $\sum_{(id,r,t,[\ell,u]) \in \mathcal{K}''} \ell = 1 - c/tot$ and $|\mathcal{K}''| > |\mathcal{K}'|$. Construct $S'' = \{s_i | (id, \{p_i\}, 0, [s_i/tot, s_i/tot]) \notin \mathcal{K}''\}$. Note that $\sum_{s_i \in S''} s_i = c$ and that $|S''| < |S'|$, making S'' a counter example to (S, c, S') being an instance of minimal subset sum. Contradiction.

□

5.6 Minimizing Spatial Change

One may think that we can revise $\mathcal{S}$ by changing the spatial component r of atoms in $\mathcal{S}$. A *spatial revision* of PST-atom $a = (id, r, t, [\ell, u])$ is an atom of the form $a' = (id, r', t, [\ell, u])$, where r' is a revised region. The distance $d_S(a, a')$ is given by $abs(|r \cup r'| - |r \cap r'|)$. A spatial revision of SPOT database $\mathcal{S}$ is a database $\mathcal{S}'$ containing at most one spatial revision of each atom in $\mathcal{S}$. The distance between a SPOT database and its spatial revision ($d_S(\mathcal{S}, \mathcal{S}')$) is the sum of the distances between the individual atoms and their associated spatial revision.

Definition 30. *A spatial revision $\mathcal{S}'$ of $\mathcal{S}$ w.r.t. an inserted* SPOT *atom a is* optimal *iff $\mathcal{S}' \cup \{a\}$ is consistent and there is no other spatial revision $\mathcal{S}''$ of $\mathcal{S}$ w.r.t. a such that $\mathcal{S}'' \cup \{a\}$ is consistent and $d_S(\mathcal{S}, \mathcal{S}'') < d_S(\mathcal{S}, \mathcal{S}')$. We use $\mathcal{S} \dot{+}_s a$ to denote an optimal spatial revision $\mathcal{S}'$. As in the case of the max-subset strategy, there may be multiple optimal spatial revision strategies and we use the total order O_T from footnote 2 to deterministically pick one when several are possible.*

Unfortunately, in general, as the following example shows, there may be cases where no spatial revision satisfies AGM axioms **(A1)** and **(A5)**.

Example 21. *Suppose $ID = \{id\}$ and $\mathcal{L} = \{p_1, p_2\}$. Let $\mathcal{S} = \{a_1\}$ where $a_1 = (id, \{p_1\}, 0, [0.5, 0.5])$. Let $a = (id, \{p_1\}, 0, [0, 0])$. By **(A1)**, $\mathcal{S} \dot{+}_s a$ must be a* SPOT *database. However, $\mathcal{S} \cup \{a\}$ is inconsistent and $\mathcal{S}$ must be revised. There are 3 possible revised KBs depending on which subset of $\{p_1, p_2\}$ is used as the spatial component of a_1. These three choices are: $(id, \{p_1\}, 0, [0.5, 0.5])$, $(id, \{p_2\}, 0, [0.5, 0.5])$, and $(id, \{p_1, p_2\}, 0, [0.5, 0.5])$. None of these atoms is consistent with a. Hence in all possible spatial revisions, Axiom **(A5)** is violated.*[3]

Thus we cannot achieve spatial update in the general case.

5.7 Minimizing Temporal Change

In this section, we study what happens when we revise a SPOT database $\mathcal{S} = \{a_1, \ldots, a_n\}$ by changing $a_i = (id_i, r_i, t_i, [\ell_i, u_i])$ to $a_i' = (id_i, r_i, t_i', [\ell_i, u_i])$. In other words, the only change allowed in a atom is the modification of the time stamp. Given a SPOT database $\mathcal{S}$ of the above form, we call such a revised SPOT database a *temporal variant* of $\mathcal{S}$.

[3]Note that this example does not depend upon how the distance function d_S is defined.

The distance between a temporal variant $\{a'_1, \ldots, a'_n\}$ of a SPOT database $\mathcal{S} = \{a_1, \ldots, a_n\}$, denoted $d_T(\mathcal{S}, \mathcal{S}')$ is given by $\sum_{i=1}^{n} |t_i - t'_i|$.

$\mathcal{S}'$ is called a *temporally optimal variant* of $\mathcal{S}$ w.r.t. an inserted SPOT atom a iff (i) $\mathcal{S}' \cup \{a\}$ is consistent, (ii) $\mathcal{S}'$ is a temporal variant of $\mathcal{S}$ and (iii) there is no other temporal variant $\mathcal{S}''$ of $\mathcal{S}$ such that $\mathcal{S}'' \cup \{a\}$ is consistent and $d_T(\mathcal{S}, \mathcal{S}'') < d_T(\mathcal{S}, \mathcal{S}')$. As in the case of the previous two revision strategies, there can be multiple temporally optimal variants - we assume the existence of a strict total ordering O_T (from footnote 2) to determine which is best returned. We denote this temporally optimal variant of $\mathcal{S}$ w.r.t. atom a by $\dot{+}_t$. The following result shows that checking for temporally optimal variants is NP-hard.

Theorem 8. *Suppose $\mathcal{S}$ is a* SPOT *database and a is an insertion. Checking if $\mathcal{S}'$ is a temporally optimal variant of $\mathcal{S}$ is NP-hard.*

Proof. For this proof, we use the coNP-complete problem minimum subset sum (definition 29):

Let $(\mathcal{K}, ra, \mathcal{K}')$ be an instance of the optimal temporal revision problem where $\mathcal{K}, \mathcal{K}'$ are knowledgebases and ra is a revision atom. $(\mathcal{K}, ra, \mathcal{K}')$ is a positive instance iff $\mathcal{K}'$ is an optimal temporal revision of $\mathcal{K}$ with respect to ra (here we do not consider the total order O_T).

We do a reduction from minimum subset sum. Consider an instance of minimum subset sum (Definition 29) $(S = \{s_1, \ldots, s_n\}, c, S')$. Let $\mathcal{L} = \{p_1, \ldots, p_{2n+1}\}$. Let time $T = [0, \infty]$. Create a reachability predicate where $reachable(id, p_i, p_j)$ is false unless: $j = n + 1$ and $i < n + 1$, or $j > n$ and $j = i + 1$, or $j < n + 1$ and $i = 2n + 1$. Let $t = \sum_{s_i \in S} s_i$. Let $\mathcal{K} = \{(id, \{p_i\}, 0, [s_i/t, s_i/t]) | s_i \in S\}$, $ra = (id, \{p_{n+1}\}, 1, [1 - c/t, 1 - c/t])$, and

$$\mathcal{K}' = \{(id, \{p_i\}, 1, [s_i/t, s_i/t]) | s_i \in S'\} \cup \{(id, \{p_i\}, 0, [s_i/t, s_i/t]) | s_i \notin S'\}$$

(S, c, S') is minimal subset sum iff $(\mathcal{K}, ra, \mathcal{K}')$ is an optimal temporal revision.

($\Rightarrow$): Notice $d_t(\mathcal{K}, \mathcal{K}') = |S'|$. To derive a contradiction, suppose that (S, c, S') is minimal subset sum and $(\mathcal{K}, ra, \mathcal{K}')$ is not an optimal temporal revision. Then there is a temporal revision $\mathcal{K}''$ s.t. $d_t(\mathcal{K}, \mathcal{K}'') < d_t(\mathcal{K}, \mathcal{K}')$. Let

$$\mathcal{K}''_{\neq 0} = \{(id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}'' | t_i \neq 0\}$$

and let $\mathcal{K}''_0 = \mathcal{K}'' \setminus \mathcal{K}''_{\neq 0}$. Observe several things about $\mathcal{K}''_{\neq 0}$:

- For all $(id, \{p_i\}, t_i, [s_i/t, s_i/t])$, $t_i \leq n$. If not, then $d_t(\mathcal{K}, \mathcal{K}'') > n$ and since $|S'| \leq n$, $d_t(\mathcal{K}, \mathcal{K}')$ would be smaller and $\mathcal{K}''$ would not be the counter example.

- Because of the reachability predicate, all $t_i > 0$ are equal to 1. If ra is consistent with $\mathcal{K}''$, then ra is consistent with

  $$\mathcal{K}''_0 \cup \{(id, \{p_i\}, 1, [s_i/t, s_i/t]) | (id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}''_{\neq 0}\}.$$

  Since $\mathcal{K}''$ is supposed to have minimal $d_t(\mathcal{K}, \mathcal{K}'')$ this implies that for every $(id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}''_{\neq 0}$, $t_i = 1$.

- Because $d_T(\mathcal{K}, \mathcal{K}'') < d_T(\mathcal{K}, \mathcal{K}')$, $|\mathcal{K}''_{\neq 0}| < |S'|$.

- Because all atoms in $\mathcal{K}''_{\neq 0}$ are at time point 1, to be consistent with ra, it must be the case that: $\sum_{(id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}''} s_i/t \leq c/t$.

- Because of the reachability predicate, the probability left in locations $p_1, \ldots, p_n$ at time 0 all goes to the location p_{n+1}. Thus to be consistent with ra, it must be the case that: $\sum_{(id, \{p_i\}, 0, [s_i/t, s_i/t]) \in \mathcal{K}''_0} s_i/t \leq 1 - c/t$.

- Since it is further the case that $\sum_{s_i} s_i/t = 1$, and since $\mathcal{K}''_0$ and $\mathcal{K}''_{\neq 0}$ are disjoint and cover $\mathcal{K}''$, we have that $\sum_{(id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}''_{\neq 0}} s_i/t \geq c/t$. Since

we already have established that $\sum_{(id,\{p_i\},t_i,[s_i/t,s_i/t])\in\mathcal{K}''} s_i/t \leq c/t$ we have: $\sum_{(id,\{p_i\},t_i,[s_i/t,s_i/t])\in\mathcal{K}''_{\neq 0}} s_i/t = c$.

Construct $S'' = \{s_i | (id, \{p_i\}, t_i, [s_i/t, s_i/t]) \in \mathcal{K}''_{\neq 0}$. Since

$$\sum_{(id,\{p_i\},t_i,[s_i/t,s_i/t])\in\mathcal{K}''_{\neq 0}} s_i/t = c,$$

we know that $\sum_{s_i\in S''} s_i = c$. Further, since $d_T(\mathcal{K}, \mathcal{K}'') < d_T(\mathcal{K}, \mathcal{K}')$ we know that $|S''| < |S'|$. Thus S'' is a counter example to (S, c, S') contradicting our assumption that (S, c, S') is in minimum subset sum.

($\Leftarrow$): Suppose $(\mathcal{K}, ra, \mathcal{K}')$ is an optimal temporal revision and (S, c, S') is not minimal subset sum to derive a contradiction. Let S'' be the counter-example to (S, c, S'). Construct $\mathcal{K}''$ as:

$$\mathcal{K}'' = \{(id, \{p_i\}, 1, [s_i/t, s_i/t]) | s_i \in S''\} \cup \{(id, \{p_i\}, 0, [s_i/t, s_i/t]) | s_i \notin S''\}.$$

Note that since $|S''| < |S'|$, $d_T(\mathcal{K}, \mathcal{K}'') < d_T(\mathcal{K}, \mathcal{K}')$. Further notice that since $\sum_{s\in S''} s = c$, the total assigned probability at time 1 for id is 1 in $\mathcal{K}'' \cup \{ra\}$ and it is consistent. Thus $\mathcal{K}''$ is a conterexample to $(\mathcal{K}, ra, \mathcal{K}')$. However, this contradicts the assumption that $(\mathcal{K}, ra, \mathcal{K}')$ is an optimal temporal revision.

□

It is not always possible to do temporal revision while satisfying the AGM axioms. Consider:

Example 22. *Consider* $T = \{t_0, \cdots, t_n\}$ *and* $\mathcal{L} = \{L_0, \cdots, L_n\}$. *Let*

$$\mathcal{S} = \{(id, \{L_i\}, t_i, [1, 1]) \,|\, 0 \leq i \leq n\}$$

$\mathcal{S}$ *assigns probability* 1 *to object* id *being at* L_i *at time* t_i. *Now, we consider a revision atom* $ra = (id, \{L_1\}, t_0, [1, 1])$. *Any temporal variant of* $\mathcal{S}$ *which is consistent with*

ra will be inconsistent: there will always be some time point t_i which contains both $(id, \{L_0\}, t_i, [1,1])$ and $(id, \{L_i\}, t_i, [1,1])$.

The above example proves that not all temporal revision can satisfy Axiom **(A5)**. The example relies upon the range of the temporal update being finite, so we introduce a new assumption: that T is unbounded. Unfortunately, in this case there are some reachability predicates which eliminate the possibility of AGM-compliant temporal revision.

Example 23. *Consider $\mathcal{L} = \{L_0, L_1\}$, $T = \{0, \cdots, \infty\}$, and $ID = \{id\}$ and let $reachable(id, L_i, L_j)$ be true iff $i = j$. Further consider the* SPOT *database*

$$\mathcal{S} = \{(id, \{L_0\}, 0, [1,1])\}$$

and the revision atom $ra = (id, \{L_1\}, 0, [1,1])$. Note that there is no t such that

$$(id, \{L_1\}, 0, [1,1]), (id, \{L_0\}, t, [1,1])$$

is consistent, as there is no way to reach L_0 from L_1 and vice versa. Thus there is no possible temporal revision in this case.

This example relies on a restricted reachability predicate to ensure that temporal revision is not possible. However, if we assume that there are no "islands" of mutually unreachable locations, then the above counter-example does not apply.

We therefore introduce the assumption that for any pair of locations $L, L' \in \mathcal{L}$ there is a path $L = L_1, \cdots, L_k = L'$ such that $reachable(id, L_i, L_{i+1})$ for all $1 \leq i < k$. We call this the *full-reachability* assumption. Under this assumption, temporal revision may be accomplished.

The ***TemporalRevision($\mathcal{S}, a$)*** algorithm works by using *unary* temporal variants. $(id, r, t', [\ell, u])$ is a unary temporal variant of $(id, r, t, [\ell, u])$ iff $abs(t - t') = 1$. The

algorithm creates a search tree - each node N in the search tree has an $N.DB$ field. The root of the search tree is initialized to $Root.KB = \mathcal{S}$. Every child C of a node N is just like N except that exactly one PST atom in $N.DB$ is replaced by a unary temporal variant. Further, each child knowledge base is required to be further (according to d_T) from $\mathcal{S}$ than its parent. When visiting a node N, the algorithm checks if $N.DB \cup \{a\}$ is consistent. By creating and visiting this tree in breadth first order, we are guaranteed that the first node that satisfies this consistency check is an optimal temporal variant of $\mathcal{S}$ that accomplishes the insertion of a.

Algorithm 4 ***TemporalRevision***$(\mathcal{S}, a)$ Search over potential temporal changes to $\mathcal{S}$.

```
If {a} is inconsistent, return "error".
Get new node Root. Set Root.DB = S;
TODO = [ Root ]. {TODO is an ordered list.}
while True do
  Let nextTODO be an empty list.
  {iterate over TODO in order.}
  for N in TODO do
    if N.DB ∪ {a} is consistent return N.DB ∪ {a}.
    Insert each child of N into nextTODO.
  end for
  Let TODO=nextTODO.
  sort TODO according to strict total ordering O_T.
end while
```

Theorem 9. *Under the full-reachability assumption, if T is unbounded then Algorithm* ***TemporalRevision*** *is correct, i.e.* ***TemporalRevision***$(\mathcal{S}, a)$ *returns a temporally optimal variant of $\mathcal{S}$ that accomplishes the insertion of a as long as a is consistent. Moreover, assuming T has non-finite size and under the full-reachability assumption,* ***TemporalRevision***$(\mathcal{S}, a)$ *satisfies the AGM-axioms.*

Proof. **(A1)** holds if *TemporalRevision*$(\mathcal{S}, a)$ returns (that is, there is no input that causes the algorithm to run forever). Clearly the algorithm will eventually try ev-

ery possible temporal revision of $\mathcal{S}$. Thus if a temporal revision consistent with a exists, it will be returned. To show that any $\mathcal{S}$ has a temporal revision that is consistent with a let t_m be the maximum amount of time it takes for any object to go from one point to another. Since we assume every point to be reachable from every other point (this is the full-reachability assumption), t_m exists and is finite. Let $\mathcal{S} = \{(id_1, r_1, t_1, [\ell_1, u_1]), \cdots, (id_n, r_n, t_n, [\ell_n, u_n])$. Furher, without loss of generality, let $a = (id, r, 0, [\ell, u])$. Now let $\mathcal{S}' = \{(id_1, r_1, t_m, [\ell_1, u_1]), \cdots, (id_i, r_i, i \cdot t_m, [\ell_i, u_i]), (id_n, r_n, n \cdot t_m, [\ell_n, u_n])$. Clearly $\mathcal{S}' \cup \{a\}$ is consistent, and clearly $\mathcal{S}'$ is a temporal variant of $\mathcal{S}$. Thus, in the worse case $\mathcal{S}' \cup \{a\}$ can be an optimal temporal variant of $\mathcal{S}$ (however unlikely), and **(A1)** holds.

(A2) holds trivially, due to the condition before the return statement. **(A3)** also holds trivially, as if a is inconsistent with $\mathcal{S}$ then the set of interpretations satisfying $\mathcal{S} \cup \{a\}$ is empty, and if a is consistent with $\mathcal{S}$ then $\mathcal{S} \cup \{a\} = \mathcal{S} \dot{+}_t a$. **(A4)** holds – $\mathcal{S} \cup \{a\}$ is returned on the first pass through the while loop in this case. **(A5)** holds due to the check before the return statement. **(A6)** is guaranteed by the strict total ordering O_T. □

The ***TemporalRevision*** algorithm takes exponential time (as expected due to Theorem 8.)

5.8 Minimizing Probability Change

In this section, we propose a belief revision operator that replaces SPOT atoms of the form $(id, r, t, [\ell, u])$ in $\mathcal{S}$ by SPOT atoms $(id, r, t, [\ell', u'])$ where $[\ell, u] \subseteq [\ell', u']$. In other words, this belief revision operator expands the probability bounds of atoms in

$\mathcal{S}$ in order to retain consistency when a is added. Obviously, we want to minimize the expansion of the probability interval $[\ell, u]$ to $[\ell', u']$.

Definition 31. *Suppose* $a = (id, r, t, [\ell, u])$ *is a* SPOT *atom and* $[\ell, u] \subseteq [\ell', u']$. *Then the* SPOT *atom* $a' = (id, r, t, [\ell', u'])$ *is called a* weakening *of* a. *The distance,* $d_P(a, a')$ *between* a *and* a' *is defined as* $(\ell - \ell') + (u' - u)$.

A PST-KB $\mathcal{S}'$ *is called a* weakening *of a PST-KB* $\mathcal{S}$ *iff there is a bijection* β *from* $\mathcal{S}$ *to* $\mathcal{S}'$ *such that for all* $a \in \mathcal{S}$, $\beta(a)$ *is a weakening of* a. *The distance* $d_P(\mathcal{S}, \mathcal{S}')$ *between* $\mathcal{S}$ *and* $\mathcal{S}'$ *is defined as* $\Sigma_{a \in \mathcal{S}} d_P(a, \beta(a))$.

In most cases, β can be derived directly by manipulating the probability bounds associated with a SPOT atom $a \in \mathcal{S}$. In the sequel we assume β is known.

Definition 32. *Suppose* $\mathcal{S}$ *is a* SPOT *database and* a *is a PST-atom. A weakening* $\mathcal{S}'$ *of* $\mathcal{S}$ *is called an* optimal weakening *of* $\mathcal{S}$ *w.r.t. the insertion of* a *iff: (i)* $\mathcal{S}' \cup \{a\}$ *is consistent and (ii) for every other weakening* $\mathcal{S}''$ *of* $\mathcal{S}$ *such that* $\mathcal{S}'' \cup \{a\}$ *is consistent,* $d_P(\mathcal{S}, \mathcal{S}') \leq d_P(\mathcal{S}, \mathcal{S}'')$.

We can find an optimal weakening of SPOT databases by setting up a linear program with variables $v_{id,t,p,q}$ each representing the probability of an object id being at location p at time t and at location q at time $t + 1$. Notice that these are the same variables as were used in the world-based semantics linear program $\mathbf{PWLP}$ (definition 25 on page 65). We limit the range of id to the set ID provided *a priori* and the range of t to the bounded set T also provided *a priori* (we assume a bounded set of timepoints T for probabilistic revision). For each SPOT atom $a_i = (id_i, r_i, t_i, [\ell_i, u_i])$ in $\mathcal{S}$, we also include variables low_i and up_i for the atoms' modified lower and upper bounds.

Definition 33 (Probability Revision Linear Program (PRLP))**.** *Let* $PRLP(\mathcal{S}, a)$ *contain only the following:*

1. *For each* $a_i = (id_i, r_i, t_i, [\ell_i, u_i]) \in \mathcal{S}$*:*
   (a) $0 \leq \left(\sum_{p \in r_i} \sum_{q \in S} v_{id_i,t_i,p,q}\right) - low_i$
   (b) $0 \geq \left(\sum_{p \in r_i} \sum_{q \in S} v_{id_i,t_i,p,q}\right) - up_i$.
   (c) $\ell_i \geq low_i$, $low_i \geq 0$, $u_i \leq up_i$, *and* $up_i \leq 1$
2. *For* $a = (id', r', t', [\ell, u])$*:*
   (a) $\ell \leq \sum_{p \in r'} \sum_{q \in S} v_{id',t',p,q}$ *and* $u \geq \sum_{p \in r'} \sum_{q \in S} v_{id',t',p,q}$
3. *For each* id *in* ID *and each* t *in* T
   (a) *For all* $p, q \in S$, $v_{id,t,p,q} \geq 0$.
   (b) $\sum_{p \in S} \sum_{q \in S} v_{id,t,p,q} = 1$
   (c) *For all* $p, q \in S$, *if* $\neg reachable(id, p, q)$: $v_{id,t,p,q} = 0$
   (d) *For all* $p \in S$: $\sum_{q \in S} v_{id,t,q,p} = \sum_{q \in S} v_{id,t+1,p,q}$

We now compute an optimal weakening of $\mathcal{S}$ by minimizing the distance function $\sum_{a_i \in \mathcal{S}} d_P(a_i, \beta(a_i))$ subject to $PRLP(\mathcal{S}, a)$. As in the case of all our revision strategies, when there are multiple solutions to this linear program, we assume there is a mechanism to deterministically pick one. We are now able to define a probabilistic revision strategy.

Definition 34 (Probabilistic Revision)**.** *Suppose* $\mathcal{S}$ *is a* SPOT *database and* a *is a* SPOT *atom. Let* θ *be a (deterministically) selected solution of the linear program*

minimize $\sum_{a_i \in \mathcal{S}}((\ell_i - low_i) + (up_i - u_i))$ ***subject to*** $PRLP(\mathcal{S}, a)$. *Return the* SPOT *database, denoted* $\mathcal{S} \dot{+}_p a$ *defined as*

$$\{(id_i, r_i, t_i, [low_i\theta, up_i\theta])|\, (id_i, r_i, t_i, [\ell_i, u_i]) \in \mathcal{S}\} \cup \{sa\}\,.$$

Since the number of points in space objects and time points is constant, only the number of atoms in $\mathcal{S}$ affect the number of variables in $PRLP()$, which is $O(|\mathcal{S}|)$. The number of constraints is similarly limited by $O(|\mathcal{S}|)$. Thus the size of the entire linear program created by PRLP is polynomial in $|\mathcal{S}|$. Since solving linear programs is also polynomial [32], and we can assume our mechanism for picking a solution deterministically runs in polynomial time[4] the above procedure computes $\mathcal{S} \dot{+} a$ in polynomial time.

This polynomial time probabilistic revision strategy also satisfies the requisite AGM axioms.

Proposition 7. $\mathcal{S} \dot{+}_p a$ *satisfies* ***(A1)-(A6)****.*

Proof. Axiom **(A1)** is straightforward. **(A2)** follows from the inequalities specified in 2a of PRLP. Axiom **(A3)** follows from the fact that upper bounds are increased and lower bounds are decreased (according to the inequalities in 1c of PRLP), loosening the knowledge base such that any interpretation satisfying $\mathcal{S} \cup \{a\}$ must also satisfy $\mathcal{S} \dot{+}_t a$. Axiom **(A4)** follows from the fact that the minimum value possible for the distance function occurs when there is not change to the knowledge base. Thus if at all possible, the algorithm returns the original values for the lower and upper bounds of the knowledge base making the updated knowledgebase equal to the original knowledgebase. Thus the set of satisfying interpretations are equal. Axiom **(A5)** follows

[4]Such mechanisms clearly exist: consider a strict total ordering over the variables that specifies the order with which the linear program solver should minimize variables.

Technique	AGM	complexity
Maximal Subset	Yes	NP-complete
Minimizing Spatial Change	No	N/A
Minimizing Temporal Change	No	N/A
with non-finite T	No	N/A
with full-reachability	Yes	coNP-hard
Minimizing Probability Change	Yes	polynomial

Figure 5.1: Complexity and AGM-compliance of revision techniques.

from the fact that any solution to $FTLP$ corresponds to a consistent knowledge base. Consider if θ is a solution to $FTLP(\mathcal{S}, a)$. θ (minus any assignments to the variables low_i and up_i) is also a solution to $LP(\mathcal{S} \dot{+}_P a)$. Axiom **(A6)** follows from the use of the strict total order. If $a \equiv a'$, then the solutions to $FTLP(\mathcal{S}, a)$ will be exactly the solutions to $FTLP(\mathcal{S}, a')$. Thus the minimal member of both sets of solutions will be the same according to the strict total order and the same repaired knowledge base will be returned. □

5.9 Comments on Belief Revision

In this section I introduced belief revision in the SPOT database. This is a process whereby new information, inconsistent with currently stored information in the SPOT database, can be added without causing the database to be inconsistent. It is of clear use, and possibly even necessary, for the sorts of situations where a solution based on SPOT might be employed: such situations are characterized by a relatively high degree of uncertainty about data's accuracy.

To verify the effectiveness of potential revision strategies, I compared each strategy to the AGM revision axioms, and discovered some cases where AGM-compliant revision is impossible to guarantee (spatial revision, temporal revision with finite T, etc). There were also some coNP-hard strategies (i.e. max subset revision). The final revision strategy analyzed was one that modified the atoms' probabilities and nothing else. I found I could compute revisions under this probability change strategy in polynomial time. Unless other considerations take precedent, this result suggests that probability change should be the default revision strategy.

I close with one final comment on the polynomial time probability change revision strategy: since most most other non-probabilistic logics allow only NP-complete revision strategies commensurate with the AGM axioms, this is a rare example of where computational rather than representational factors favor the inclusion of probabilities in the representation.

Chapter 6

SPOT Algebra

In this chapter, the basics of the SPOT algebra are covered. First, tightness is dealt with. In a tight SPOT database, all atoms have the smallest bounds possible. Then, the algebraic operations of union, intersection, and difference are considered. A approach to "join" that shows how one might combine the SPOT databases also appears. Finally, algorithms involving expected distance queries are discussed, with some experimental results.

6.1 SPOT Database Tightness

The probability bounds associated with a SPOT database can often be tightened. To achieve this, we use the idea of database equivalence (Definition 8 on page 35) databases are equivalent.

Definition 35. *A* SPOT *database* $\mathcal{S}$ *is called* tight *if for every* SPOT *database* $\mathcal{T}$ *such that* $\mathcal{S} \equiv \mathcal{T}$*, for every* SPOT *atom* $(id_i, r_i, t_i, [\ell_i, u_i]) \in \mathcal{S}$ *there is no* SPOT *atom* $(id_i, r_i, t_i, [\ell_j, u_j]) \in \mathcal{T}$ *such that* $\ell_i < \ell_j$ *or* $u_j < u_i$*.*

Example 24. *Consider two databases in a space where $\mathcal{L} = \{L_1, L_2\}$:*

$$\mathcal{S}_1 = \{(id, \{L_1\}, 0, [0.5, 0.5]), (id, \{L_2\}, 0, [0.5, 0.5])\}$$

$$\mathcal{S}_2 = \{(id, \{L_1\}, 0, [0.5, 0.5]), (id, \{L_2\}, 0, [0.2, 0.6])\}$$

$\mathcal{S}_1$ is tight, while $\mathcal{S}_2$ is not.

Figure 5 shows an algorithm tighten, which tightens a SPOT database.

Algorithm 5 Tightens a SPOT database $\mathcal{S}$

tighten($\mathcal{S}$)

Let $\mathcal{S}' := \emptyset$.

for $(id, r, t, [\ell, u]) \in \mathcal{S}$ **do**

$L :=$ **minimize** $\sum_{(x,y) \in r} v_{x,y}$ **w.r.t.** $LC(\mathcal{S}, id, t)$;

$U :=$ **maximize** $\sum_{(x,y) \in r} v_{x,y}$ **w.r.t.** $LC(\mathcal{S}, id, t)$;

$\mathcal{S}' := \mathcal{S}' \bigcup \{(id, r, t, [L, U])\}$

end for

Return $\mathcal{S}'$.

Theorem 10. *For all $\mathcal{S}$, tighten($\mathcal{S}$) is tight and equivalent to $\mathcal{S}$.*

Proof Intuition: Since tighten minimizes u and maximizes ℓ, for a specific id, r, and t, there cannot be an equivalent database containing an atom with larger ℓ or smaller u for the same id, r, and t.

Tightness is important because it can often lead to simpler and faster algorithms.

6.2 Union, Intersection, Difference, and Join

Some queries require the standard operations: union, intersection, difference, and join. In this section we show how to define these operations for consistent SPOT databases.

Union

We start with union. The union of SPOT databases will be normal set union.

Definition 36 (SPOT Union)**.** $\mathcal{S}_1 \cup_{\mathsf{SPOT}} \mathcal{S}_2 = \{a | a \in \mathcal{S}_1 \vee a \in \mathcal{S}_2\}$

This definition is natural both syntactically and semantically, as it satisfies the equality $\mathcal{I}(\mathcal{S}_1 \cup_{\mathsf{SPOT}} \mathcal{S}_2) = \mathcal{I}(\mathcal{S}_1) \cap \mathcal{I}(\mathcal{S}_2)$.

Intersection

Intersection and difference are more complicated for SPOT databases because of the possibility that $sa_1 = (id, r_1, t, [\ell_1, u_1]) \in \mathcal{S}_1$, $sa_2 = (id, r_2, t, [\ell_2, u_2]) \in \mathcal{S}_2$, $r_1 \neq r_2$, and $r_1 \cap r_2 \neq \emptyset$. The meaning of intersection and difference is not clear in such a case. We therefore define intersection and difference only for compatible SPOT databases. We say databases S_1 and S_2 are compatible if whenever $sa_1 = (id, r_1, t, [\ell_1, u_1]) \in \mathcal{S}_1$, and $sa_2 = (id, r_2, t, [\ell_2, u_2]) \in \mathcal{S}_2$, then either $r_1 = r_2$ or $r_1 \cap r_2 = \emptyset$.

Consider intersection first. In general, union adds restrictions while intersection removes them. Ideally we would like $\mathcal{I}(\mathcal{S}_1 \cap_{\mathsf{SPOT}} \mathcal{S}_2) = \mathcal{I}(\mathcal{S}_1) \cup \mathcal{I}(\mathcal{S}_2)$. But suppose that $\mathcal{S}_1 = \{sa_1 = (id, r, t, [.1, .3])\}$ and $\mathcal{S}_2 = \{sa_2 = (id, r, t, [.5, .7])\}$. Here $\mathcal{I}(\mathcal{S}_1) \cup \mathcal{I}(\mathcal{S}_2)$ contains all interpretations where $.1 \leq I(id, r, t) \leq .3$ or $.5 \leq I(id, r, t) \leq .7$, and that cannot be expressed using SPOT atoms. So we define $\mathcal{S}_1 \cap \mathcal{S}_2$ in such a way that $\mathcal{I}(\mathcal{S}_1 \cap \mathcal{S}_2)$ is a superset of $\mathcal{I}(\mathcal{S}_1) \cup \mathcal{I}(\mathcal{S}_2)$.

Definition 37 (Intersection)**.**

$$\mathcal{S}_1 \cap_{\mathsf{SPOT}} \mathcal{S}_2 =$$
$$\left\{ (id, r, t, [\ell, u]) \;\middle|\; \begin{array}{ll} (id, r_1, t, [\ell_1, u_1]) \in \mathcal{S}_1 \wedge (id, r_2, t, [\ell_2, u_2]) \in \mathcal{S}_2) & \wedge \\ (r_1 = r_2 = r \wedge \ell = min(\ell_1, \ell_2) \wedge u = max(u_1, u_2)) & \end{array} \right\}.$$

Proposition 8. *$\mathcal{I}(\mathcal{S}_1 \cap_{\mathsf{SPOT}} \mathcal{S}_2)$ is a superset of $\mathcal{I}(\mathcal{S}_1) \cup \mathcal{I}(\mathcal{S}_2)$.*

Difference

Difference, like intersection, also removes restrictions, in general. Again, we would like $\mathcal{I}(\mathcal{S}_1 \setminus_{\mathsf{SPOT}} \mathcal{S}_2) = \mathcal{I}(\mathcal{S}_1) \cup \overline{\mathcal{I}}(\mathcal{S}_2)$ where $\overline{\mathcal{I}}(\mathcal{S}_2) = \{I | I \notin \mathcal{I}(\mathcal{S}_2)\}$. The following example shows why this is not always possible. Let $\mathcal{S}_1 = \{sa_1 = (id, r, t, [.1, .5])\}$ and $\mathcal{S}_2 = \{sa_2 = (id, r, t, [.3, .7])\}$. $\mathcal{I}(\mathcal{S}_1) \cup \overline{\mathcal{I}}(\mathcal{S}_2)$ contains all interpretations where $.1 \leq I(id, r, t) \leq .5$, and $I(id, r, t) < .3$ or $I(id, r, t) > .7$, and that cannot be expressed using SPOT atoms. So we define $\mathcal{S}_1 \setminus_{\mathsf{SPOT}} \mathcal{S}_2$ in such a way that $\mathcal{I}(\mathcal{S}_1 \setminus \mathsf{SPOT}\ \mathcal{S}_2)$ is a superset of $\mathcal{I}(\mathcal{S}_1) \cup \overline{\mathcal{I}}(\mathcal{S}_2)$.

Definition 38 (Difference)**.**

$$\mathcal{S}_1 \setminus_{\mathsf{SPOT}} \mathcal{S}_2 = \left\{ (id, r, t, [\ell, u]) \;\middle|\; \begin{array}{ll} ((id, r, t, [\ell_1, u_1]) \in \mathcal{S}_1 \wedge \mathcal{S}_2^{id,t} = \emptyset) & \vee \\ ((id, r, t, [\ell_1, u_1]) \in \mathcal{S}_1 & \wedge \\ (id, r, t, [\ell_2, u_2]) \in \mathcal{S}_2 \wedge (\ell_2 = 0 \vee u_2 = 1) & \wedge \\ \textit{if } \ell_2 = 0 \textit{ then } \ell = \ell_1 \textit{ else } \ell = 0 & \wedge \\ \textit{if } u_2 = 1 \textit{ then } u = u_1 \textit{ else } u = 1) & \end{array} \right\}$$

Proposition 9. *$\mathcal{I}(\mathcal{S}_1 \setminus_{\mathsf{SPOT}} \mathcal{S}_2)$ is a superset of $\mathcal{I}(\mathcal{S}_1) \cup \overline{\mathcal{I}}(\mathcal{S}_2)$.*

Join

In non-probabilistic settings, join operations typically take tuples that match according to some criteria, and concatenate or combine them in some way. This same idea is also used in the definition of join for SPOT databases. Suppose j is a function that combines two rectangles into a region (not necessarily a rectangle). For example, j could return the intersection of the two rectangles or the minimal bounding rectangle of the union of the two rectangles, or some other rectangle altogether. Our notion of join combines rectangles together using a given j as follows.

Definition 39 (Join)**.** *We define $S_1 \bowtie_j S_2$ in terms of each id and t. Let $\Gamma^{id,t}$ be the union of the sets of linear constraints defined by $\mathcal{S}_1^{id,t}$ and $\mathcal{S}_2^{id,t}$.*

$$(S_1 \bowtie_j S_2)^{id,t} = \left\{ \begin{array}{l} S_1^{id,t} \; if \; S_2^{id,t} = \emptyset \\ S_2^{id,t} \; if \; S_1^{id,t} = \emptyset \\ \left\{ (id, r, t, [\ell, u]) \;\middle|\; \begin{array}{l} (id, r_1, t, [\ell_1, u_1]) \in S_1 \wedge \\ (id, r_2, t, [\ell_2, u_2]) \in S_2 \wedge \\ r = j(r_1, r_2) \wedge \\ \ell = min \sum_{p \in r} v_p \textit{ subj.t. } \Gamma^{id,t} \wedge \\ u = max \sum_{p \in r} v_p \textit{ subj.t. } \Gamma^{id,t} \end{array} \right\} \end{array} \right.$$

If $\Gamma^{id,t}$ has no solutions, $S_1 \bowtie_j S_2$ does not exist.

6.3 Expected Distance Queries

6.3.1 Expected Distance

In this section, we use the statistical notion of an expected value to define the *expected distance* from an object to a given point or object. Examples of the use of expected

distance include calculating quantities such as expected fuel usage, expected time of arrival, and best guess threat level in a military application. As we don't know exactly where objects are at any given time, nor do we know the exact probabilities involved, we give a *minimal* expected distance and a *maximal* expected distance between an object and a point, and later extend these concepts to minimum and maximum expected distances between objects.

In order to define expected distance formally we first define distance between points in the usual way.

Definition 40. The Euclidean distance, $ed((x, y), (x', y'))$, between 2 points (x, y) and (x', y'), *is given by* $\sqrt{(x - x')^2 + (y - y')^2}$.

The standard notion of expected value in statistics looks at all possible values, multiplies each of them by the probability of that value, and adds such products together. We now define the concept of expected distance between a fixed point p and an object w.r.t. an interpretation I. Informally speaking, this is done as follows: for each point q in $\mathcal{L}$, compute the product of the distance between p and q and the probability that the object is at q (according to I) and then add up the product values obtained for the different q's in $\mathcal{L}$.

Definition 41. Expected Distance between a point and an object for SPOT interpretation I:

$distToPoint^I(id, (x, y), t) = \sum_{(x',y') \in \mathcal{L}} I(id, t, (x', y')) \times ed((x, y), (x', y'))$

Now we can define the expected distance between two objects.

Definition 42 (Expected Distance between two objects for a SPOT interpretation I.)**.**

$$dist^I(id, id', t) = \sum_{\substack{(x,y)\in\mathcal{L},\\(x',y')\in\mathcal{L}}} (I(id, t, (x, y)) \otimes I(id', t, (x', y'))) \times ed((x, y), (x', y'))$$

In the above definition, $I(id, t, (x, y)) \otimes I(id', t, (x', y'))$ specifies the joint probability that the two objects are at the locations specified. For the remainder of the paper, we assume independence of all object locations making $\otimes$ the multiplication operation.

In general, many different SPOT interpretations satisfy a given SPOT database and there can be a different expected distance between two objects for every interpretation. Thus we define the minimal (resp. maximal) expected distances between an object and a point (or another object) to be the minimal (resp. maximal) expected distance over all SPOT interpretations that satisfy the SPOT database.

Definition 43. Minimal and Maximal expected distances between an object and a point*:*

$$mindistToPoint(\mathcal{S}, id, t, (x, y)) = \min\{distToPoint^I(id, t, (x, y))|I \models \mathcal{S}\},$$
$$maxdistToPoint(\mathcal{S}, id, t, (x, y)) = \max\{distToPoint^I(id, t, (x, y))|I \models \mathcal{S}\}$$

Definition 44. Minimal and Maximal expected distance between objects

$$\begin{aligned} mindist(\mathcal{S}, id, id', t) &= min\{dist^I(id, id', t)|I \models \mathcal{S}\}, \\ maxdist(\mathcal{S}, id, id', t) &= max\{dist^I(id, id', t)|I \models \mathcal{S}\} \end{aligned}$$

The functions, $mindistToPoint$ and $maxdistToPoint$ can be computed by linear program solvers. However, we have a more efficient non-linear programming approach when the database is disjoint.

Definition 45 (Disjoint SPOT Database)**.** SPOT *database $\mathcal{S}$ is disjoint if, for all id and all t such that there is $(id, r_1, t, [\ell_1, u_1]), (id, r_2, t, [\ell_2, u_2]) \in \mathcal{S}$ the r_1 is disjoint from r_2.*

We now present algorithms that solve or bound $mindist$ and $maxdist$ queries on disjoint SPOT databases. All the functions in this section use a "helper" function χ (Algorithm 6). χ is parameterized by a function $v : \mathcal{L} \rightarrow \mathbb{R}$. v is *any* function that associates a value with a given point p in $\mathcal{L}$. For example, $v(p)$ might be distance between the input p and some other point p': $v(p) = ed(p, p')$. χ^v is a general function for finding an interpretation that satisfies the SPOT database and assigns (for a given id and time point t) the smallest possible value to the sum of all the products of the interpretation at p times $v(p)$ in the entire $\mathcal{L}$. That is, χ returns the value

$$\chi^{v()}(\mathcal{S}, id, t) = \min_{I \models \mathcal{S}} \left(\sum_{(x,y) \in \mathcal{L}} I(id, t, (x, y)) \times v(x, y) \right)$$

To do this, first χ orders the points in $\mathcal{L}$ according to $v(p)$. Then, starting with the minimal $v(p)$ value, χ iterates over the points in order assigning to each the largest possible probability based on the SPOT atoms in $\mathcal{S}^{id,t}$ and the probabilities already assigned. This function runs in time $O(n \cdot N \cdot \log(N))$ where n is the number of SPOT atoms in the database and N is the number of points in $\mathcal{L}$. This algorithm *does not use linear programming*, and is a huge improvement over LP methods that run in either exponential time (e.g. simplex) or superlinear polynomial time (e.g. interior point methods such as Khachiyan [32]) with respect to the size of space and linear time with respect to the size of the database.

First we present the correctness of Algorithm 6. The correctness of the later algorithms are all essentially corollaries of this result.

Algorithm 6 Find $min_{I \models \mathcal{S}} \sum_{p \in \mathcal{L}} I(id, t, p) \times v(p)$.

$\chi^{v()}(\mathcal{S}, id, t)$

Require: $\mathcal{S}$ is consistent, tight, and disjoint.

1: Let $k := |\mathcal{S}^{id,t}|$.
2: Let r_i be s.t. $(id, r_i, t, [\ell_i, u_i]) \in \mathcal{S}^{id,t}$, for $1 \leq i \leq k$.
3: Let $r_{k+1} := \mathcal{L} - \cup_{i=1}^{k} r_i$ {all points not in any region specified by a SPOT atom}
4: Let p_{i,m_i}, $1 \leq i \leq n$, $1 \leq m_i \leq k+1$ be a list of all the points in $\mathcal{L}$ s.t. $v(p_{i,m_i}) \leq v(p_{i+1,m_{i+1}})$ and $p_{i,m_i} \in r_{m_i}$. {The first p subscript orders the points according to the function v and the second p subscript gives the region number where the point is located}
5: Let $\ell_b := \sum_{j=1}^{k} \ell_j$ {sum of lower bounds for regions not yet considered}
6: {the following if block computes the tightest bounds for r_{k+1}}
7: **if** $r_{k+1} \neq \emptyset$ **then**
8: Let $u_b := \sum_{j=1}^{k} u_j$
9: Let $\ell_{k+1} := max(0, 1 - u_b)$
10: Let $u_{k+1} := 1 - \ell_b$
11: Let $\mathcal{S}^{id,t} := \mathcal{S}^{id,t} \cup \{(id, r_{k+1}, t, [\ell_{k+1}, u_{k+1}])\}$
12: **end if**
13: Let $i := 1$ {index through the points in $\mathcal{L}$ (space) using the ordering in line 4}
14: Let $tot := 0$ {the sum of the probabilities in the interpretation constructed so far}
15: Let $covered := \emptyset$ {the regions covered so far}
16: Let $retValue := 0$
17: {Give each point the highest possible probability, excepting every point in a region already in $covered$ gets 0 probability}
18: **while** $i \leq n \wedge tot < 1$ **do**
19: **if** $m_i \notin covered$ **then**
20: Let $covered := covered \cup \{m_i\}$.
21: Let $tot' := tot + \ell_b - \ell_{m_i}$ {sum of the interpretation so far plus sum of lower bounds that must still be considered}
22: Let $u'_i := min(u_{m_i}, 1 - tot')$. {the largest possible interpretation at p_{im_i}}
23: Let $tot := tot + u'_i$. {update sum of the interpretations so far}
24: Let $\ell_b := \ell_b - \ell_{m_i}$ {update sum of lower bounds that must still be considered}
25: Let $retValue := retValue + u'_i \times v(p_{i,m_i})$ {update sum of the products so far}
26: **end if**
27: Let $i := i + 1$.
28: **end while**
29: **return** $retValue$

Theorem 11. *If $\mathcal{S}$ is consistent and disjoint, then*

$$\chi^{v()}(\mathcal{S}, id, t) = \min_{I \models \mathcal{S}} \left(\sum_{(x,y) \in \mathcal{L}} I(id, t, (x, y)) \times v(x, y) \right)$$

Proof Intuition: χ orders the points in space according to v, then proceeds to find the minimal possible probability for the highest value point in turn. The variables $tot, tot', \ell_b, covered, etc$ all serve book-keeping functions ensuring that there will be an interpretation with the assigned values.

We can now use χ to compute many of the distance functions. Algorithm 7 shows how to apply χ to find the minimum distance to a point function. Then, Algorithm 8 uses the earlier algorithms to substitute a v function for χ.

Corollary 4. $mindistToPoint\_ftn(\mathcal{S}, id, t, (x, y))$ *(algorithm 7) returns* $mindistToPoint(\mathcal{S}, id, t, (x, y))$.

maxdistToPoint can be computed in the same manner.

6.3.2 Experimental Results

We have implemented all the algorithms specified above in Java. The results verify expectations: linear programming takes longer than the specialized methods we developed. We did all of our experiments on a 3.2 GHz P4 computer with 2 GB of RAM. The amount of RAM available was an issue for certain linear programming problems: the amount of memory needed passes 2 GB once we have about one million variables. We used the GPLed linear problem solver *lp_solve*[1], which implements a version of the simplex method.

[1]From `http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml`

Description of Experimental Environment

Experiments are based on data about the locations of all post offices in Washington DC[2]. We modeled the motion (uncertain over space and time) of postal trucks between the 287 post offices in DC.

Trucks travel a route between randomly chosen post offices. In each time period, a truck is assumed to travel between its current post office and the next post office on its route. When a truck is to travel from one PO to another PO, we draw a straight line between the POs. We then make d disjoint rectangles near this path. Each rectangle then is made into a SPOT atom with probability dependent on the rectangle's distance from the path. We call this number d the *density* of the database. This models the fact that a truck is most likely to travel the route defined by the shortest path, yet is also capable of traveling a different path.

In putting a grid over the entire area of Washington DC, we used differing levels of granularity. DC maps onto a rectangle sized 21.50 km by 26.54 km. We used three different grids over this space, the first of which resulted in a space of size 192 by 302. Each one by one square in this grid represents a 112 meter by 88 meter sized region of Washington DC. The second granularity used has dimension 976 by 1510, and in it, each one by one square represents a 22 meter by 18 meter region. The last, and most fine grained partitioning of space partitions our map of Washington DC into a 1952 x 3021 grid. Each square in that grid represents a 11 meter by 9 meter region.

Expected Distance to a Point

The expected pattern emerges when computing expected distance to a point. Those functions dependent on linear programming were consistently beaten by our special-

[2]Obtained from `http://www.zip-codes.com`

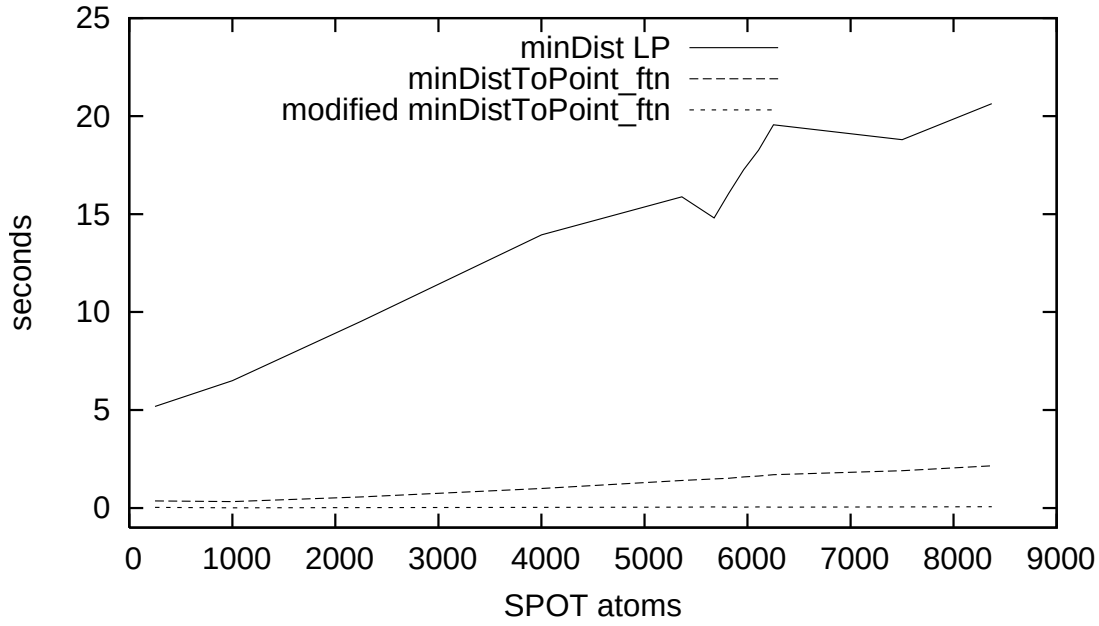


Figure 6.1: LP vs $minDistToPoint\_ftn$ in space sized 58890.

ized algorithms. In particular, we were able to get the greatest speedup with modified versions of $minDistToPoint\_ftn$ and $maxDistToPoint\_ftn$. The modifications were simple amendments to the preprocessing, whereby only one point would be considered per atom (rather than all points in space). These algorithms substantially outperform the linear programming approaches, as exemplified in Figure 6.1. In the experiments in those figures, we used DC's postal system to create the SPOT database, and increased the number of SPOT atoms by increasing the resolution of our SPOT database while by keeping the number of trucks fixed. In these experiments, we used a 195 by 302 grid over DC. We had exactly 1 truck travel a 10 post office route, and we used the densities of 25, 100, 225, 400, 525, and 900. This resulted in SPOT databases of sizes ranging from 251 to 8371.

6.3.3 Nearest Neighbor

With the above expected distance algorithms and definitions, we can now easily define and implement nearest neighbor algorithms. There is a nearest neighbor function corresponding to each type of expected distance, however, we only define the min expected nearest neighbor. Only simple modifications to this function are needed for other nearest neighbor algorithms.

Definition 46 (Min Expected Nearest Neighbor)**.** *The min expected nearest neighbor to point p at time t is given by:*

$$MENN(\mathcal{S}, p, t) = argmin_{id' \in ID}(mindistToPoint(\mathcal{S}, id, t, p)).$$

$MENN$ admits an immediate exhaustive search algorithm. The other sorts of nearest neighbor algorithms can be obtained by changing $mindistToPoint$ to, for instance, $maxdistToPoint$.

Algorithm 7 Calculates $mindistToPoint(\mathcal{S}, id, t, (x, y))$

$mindistToPoint\_ftn(\mathcal{S}, id, t, (x, y))$

Let $v(x', y')$ be the function returning $ed(x, y, x', y')$.

return $\chi^{v()}(\mathcal{S}, id, t)$

Algorithm 8 Calculates a lower bound for $mindist(\mathcal{S}, id_1, id_2, t)$

$mindist\_lb(\mathcal{S}, id_1, id_2, t)$

Let $v(x, y)$ be the function returning $mindistToPoint\_ftn(\mathcal{S}, id_1, x, y)$

return $\chi^{v()}(\mathcal{S}, id_2, t)$

Algorithm 9 Calculate an upper bound for $mindist(\mathcal{S}, id_1, id_2, t)$

$mindist\_ub(\mathcal{S}, id_1, id_2, t)$

Let $v(x, y)$ compute and return $maxdistToPoint\_ftn(\mathcal{S}, id_1, x, y)$

return $\chi^{v()}(\mathcal{S}, id_2, t)$

6.4 Comments on SPOT Algebra

Much of the work in this section represents some of the earliest work on SPOT and is presented in [40]. The goal of avoiding linear programming is clearly present in this chapter and that paper. It was, however, dropped when faster linear programming-based methodologies were found (i.e. $PLP(\cdot)$ from Definition 18). Further, while there exist reasonably straightforward algorithms like χ (Algorithm 6) for disjoint databases, it is unclear if there are algorithms for arbitrary databases that can substantially outperform linear programming techniques.

Chapter 7

Selection Algorithms

7.1 Introduction

The selection query has been a central focus of study for SPOT databases. Selection queries have many potential uses: anytime anyone with a SPOT database wants to know when objects are in a given location, a selection query will answer the question.

Example 25. *A fleet of fishing ships is tracking several schools of fish. Each ship is equipped with sonar, which identifies schools of fish and several basic measurable characteristics (size, depth, speed, etc). Using ocean current and movement information, potential locations for that school can be determined for the next several days, and specified as* SPOT *atoms:* $(school, region, time, [\ell, u])$ *(*ℓ *and* u *are determined by the accuracy of the sonar and the predictive algorithms). A captain of one vessel would like to go to region* r_1*, which is close to shore, but only wants to do so if and when he is likely to catch a full load there. The captain can determine how many of what kind of school will be in the given region by running a selection query on the fleet's* SPOT *database. The query* $(?id, r_q, ?t, [0.7, 1])$*, answered cautiously, will tell which schools of fish will be in region* r_1 *at which times with at least an 70%*

probability, allowing the captain to more efficiently plan his voyage.

For the entirety of this chapter, we will be using point-based semantics.

7.2 Selection Via Consistency Checking

7.2.1 Exhaustive Query Check

The most obvious way for computing the answer to a selection is to check each possible answer. Such a technique is inefficient, but an important starting point. I give an algorithm accomplishing this as Algorithm 10.

Algorithm 10 Exhaustively check all possible answers to an optimistic query.

$exhaustiveOptimisticAnswer(\mathcal{S}, (?id, q, ?t, [\ell, u]))$

```
Let ret = ∅.
for id ∈ ID do
  for t ∈ T do
    if (id, q, t, [ℓ, u]) is consistent with S then
      ret = ret ∪ {(id, q, t, [ℓ, u])}.
    end if
  end for
end for
return ret.
```

Clearly we can do better. In particular, we can avoid checking those $(id, q, t, [\ell, u])$ where $\mathcal{S}^{id,t}$ is inconsistent. We can also do much deeper and much more clever pruning.

A note needs to be made about the method by which we check consistency (Algorithm 10 simply checks consistency without specifying the method). One can use any of the methods for checking point-based consistency of a SPOT database detailed in Chapter 4 (page 42), for our purposes, it will not matter which method is used. All of the consistency checking methods are polynomial time or worse, and so any method

that can eliminate the need to check consistency in sub-polynomial time will be a win. I call such methods pruning methods, and I will introduce two different types: SPOT tree based pruning, and pruning methods based on potential solutions to the SPOT database's original linear program.

First we will detail the SPOT tree, an I/O-optimizable structure for performing pruning operations, and then we will explain methods for bounding the SPOT database's linear program.

7.3 SPOT Trees

SPOT trees build on top of a large volume of work on spatial indexes [43, 7, 8, 44]. In almost all trees that store spatial information, data is stored in the leaf nodes of the tree. Each node of the tree represents a region. The region associated with a node is a superset of the region associated with each of its children.

In SPOT trees we adapt spatial data structures to handle *logical spaces* and we replace the concept of inclusion with the concept of entailment. Each node in our SPOT tree is labeled by one *composite* SPOT *atom* (to be defined in this section). Each child of a composite SPOT atom labeling a node entails the composite SPOT atom that is its parent. This implies that each composite SPOT atom, in a sense, implicitly represents the set of all SPOT atoms that are compatible with it. As one goes down a branch of the tree, this set decreases. Thus, SPOT trees adapt spatial data structures to logical problems by (i) labeling nodes with composite SPOT atoms and not with geometric regions, (ii) having an entailment relationship in which parent node labels are entailed by child node labels, and (iii) having an inclusion relationship in which the set of SPOT atoms compatible with a node's composite SPOT atom

label is a superset of the set of SPOT atoms compatible with the label of a child.

This section proceeds as follows: we first define the necessary constraints to effectively rule out potential answers to an optimistic selection query. We then define composite atoms that combine (id, t)-pairs. Finally, we show how to build the SPOT tree and use it for answering optimistic selection queries, using the necessary constraints mentioned above to prune the search space.

7.3.1 Necessary Constraints

When an atom sa is compatible with a SPOT database $\mathcal{S}$, there are certain conditions that any SPOT atom entailed by $\mathcal{S}$ SPOT database will satisfy. These conditions are the following necessary constraints.

Definition 47 (Necessary constraints)**.** *Let* $ga_1 = (id, r_1, t, [\ell_1, u_1])$ *be a* SPOT *atom and* $sa_2 = (id, r_2, t, [\ell_2, u_2])$ *be a* SPOT *atom. Define:*

$$NC(ga_1, sa_2) = \begin{cases} \ell_2 \leq 1 - \ell_1 & \textit{if } r_1 \cap r_2 = \emptyset \\ \ell_2 \leq u_1 & \textit{if } r_2 \subset r_1 \\ u_2 \geq l_1 & \textit{if } r_1 \subset r_2 \\ \ell_2 \leq u_1 \wedge u_2 \geq \ell_1 & \textit{if } r_1 = r_2 \\ true & \textit{otherwise} \end{cases}$$

The following theorem shows how we can use NC to quickly determine if a given atom is incompatible with a particular SPOT database.

Theorem 12. *For consistent* SPOT *database* $\mathcal{S}$ *and* SPOT *atom* ga_1*, let* $\mathcal{S} \models ga_1$*. For any given* SPOT *atom* sa_2 *if* $NC(ga_1, sa_2)$ *is not true then* $sa_2 \notin \mathcal{S}$*.*

Proof. All the cases are similar; we do just the case where $r_1 \cap r_2 = \emptyset$. So suppose that $\ell_2 > 1 - \ell_1$, that is, $\ell_1 + \ell_2 > 1$. Hence for every interpretation I such that $I \models \mathcal{S}$, $I \not\models sa_2$. □

This theorem allows a computationally efficient technique for determining many SPOT atoms to be incompatible with $\mathcal{S}$ without appeal to a linear program using only a SPOT atom entailed by the given SPOT database.

Merging SPOT atoms with the same (id, t) pair

We now show how to merge two SPOT atoms into a single SPOT atom.

Definition 48 (region-merge)**.** *Let* $ga_1 = (id, r_1, t, [\ell_1, u_1])$*, and* $ga_2 = (id, r_2, t, [\ell_2, u_2])$ *be* SPOT *atoms with the same* id *and* t*. Let* $combine(ga_1, ga_2)$ *be defined as follows:*

$$combine(ga_1, ga_2) = \begin{cases} [max(\ell_1, \ell_2), min(u_1, u_2)] & \text{if } r_1 = r_2 \\ [\ell_1 + \ell_2, min(1, u_1 + u_2)] & \text{if } r_1 \cap r_2 = \emptyset \\ [max(\ell_1, \ell_2), u_2] & \text{if } r_1 \subsetneq r_2 \\ [max(\ell_1, \ell_2), u_1] & \text{if } r_2 \subsetneq r_1 \\ [max(\ell_1, \ell_2), min(1, u_1 + u_2)] & \text{if } r_1 \cap r_2 \neq \emptyset \\ & \wedge (r_1 \setminus r_2 \neq \emptyset) \\ & \wedge (r_2 \setminus r_1 \neq \emptyset) \end{cases}$$

We now show how to use $combine$ to combine two SPOT atoms into one that is entailed by any database also entailing the original two SPOT atoms.

Theorem 13 (region-merge)**.** *Suppose for some* SPOT *database* $\mathcal{S}$*,* id *and* t*,* $\mathcal{S} \models \{ga_1, ga_2\}$ *where* $ga_1 = (id, r_1, t, [\ell_1, u_1])$ *and* $ga_2 = (id, r_2, t, [\ell_2, u_2])$*. Then* $\mathcal{S} \models (id, r_1 \cup r_2, t, combine(ga_1, ga_2))$.

Proof. All the cases are similar: we do the case where $r_1 \cap r_2 = \emptyset$. Suppose $\mathcal{S} \models \{ga_1, ga_2\}$. So for any I such that $I \models \mathcal{S}$, $\ell_1 + \ell_2 \leq I(r_1 \cup r_2) \leq \min(1, u_1 + u_2)$, and the result follows. □

Merging SPOT atoms with different (id, t) pairs

We can also merge atoms referring to different (id, t) pairs. The following result gives probability bounds over many different (id, t) pairs.

Proposition 10 (region-merge)**.** *Let $\mathcal{S} \models \{ga_1, ga_2\}$ where $ga_1 = (id_1, r_1, t_1, [\ell_1, u_1])$, $ga_2 = (id_2, r_2, t_2, [\ell_2, u_2])$ for SPOT database $\mathcal{S}$ and $(id_1, t_1) \neq (id_2, t_2)$. Then $\mathcal{S} \models (id_1, r_1 \cup r_2, t_1, [\min(\ell_1, \ell_2), 1])$ and $\mathcal{S} \models (id_2, r_1 \cup r_2, t_2, [\min(\ell_1, \ell_2), 1])$*

7.3.2 Composite SPOT atoms

We are now ready to describe composite SPOT atoms. A composite SPOT atom is a compact representation of a set of SPOT atoms, one for each (id, t) pair, entailed by a given database.

Definition 49 (Composite SPOT atom)**.** *A composite SPOT atom is a triple $(\delta, r, [\ell, u])$ where $\delta \subset ID \times T$, r is a region in $Space$ and $\ell, u \in [0, 1]$. A composite SPOT atom $csa = (\delta, r, [\ell, u])$* represents *the set of SPOT atoms*

$$Rep(csa) = \{(id_i, r, t_i, [\ell, u]) | (id_i, t_i) \in \delta\}.$$

We write $\mathcal{S} \models csa$ to mean $\mathcal{S} \models Rep(csa)$.

A composite SPOT atom is shorthand for a set of SPOT atoms with the same region (not necessarily rectangular) and probability bounds. Each non-leaf node in

a SPOT tree will be explicitly labeled by a composite SPOT atom. The region associated with a node will be the union of the regions of the composite SPOT atoms labeling the children of that node.

We will sometimes refer to a SPOT atom in the context of a composite SPOT atom. In this case, the atom $sa = (id, r, t, [\ell, u])$ should be considered to be the obvious composite SPOT atom $(\{(id, t)\}, r, [\ell, u])$.

Intuitively, when trying to answer an optimistic selection query $(?id, r, ?t, [\ell, u])$, we traverse nodes in a SPOT tree. Each node is labeled by a csa, and has an associated set of child nodes. If $Rep(csa)$ is incompatible with all instances of $(?id, r, ?t, [\ell, u])$, then we can avoid searching the subtrees associated with csa.

Example 26. *Consider the optimistic selection query* $(?id, q, ?t, [0, 0])$ *on* SPOT *database* $\mathcal{S}$. *Suppose* $\mathcal{S} \models csa = (\{(id_1, t_1), (id_3, t_3)\}, q, t, [1, 1])$. *By Theorem 12, since* $NC((id_1, q, t_1, [1, 1]), (id_1, q, t_1, [0, 0])$ *is not true (as* $0 \not\geq 1$*),* (id_1, t_1) *cannot possibly be an answer. As a bonus, the computation implies that* (id_3, t_3) *is also not in the answer set.*

The following algorithm, $UpdateCSA$, shows how to add a SPOT atom to a composite SPOT atom.

Theorem 14 (Correctness of Algorithm 11)**.** *UpdateCSA(*csa*,*sa*) returns a composite* SPOT *atom* csa_r *such that every* SPOT *database that entails both* sa *and* csa *also entails* csa_r.

Proof. The correctness of the case where $(id, t) \notin \delta$ follows from Proposition 10. The other case follows from Theorem 13. □

Algorithm 11 This function returns a composite SPOT atom csa_r such that every SPOT database that entails both sa and csa also entails csa_r.

UpdateCSA(csa,sa)
  Let $sa = (id, r, t, [\ell, u])$
  Let $csa = (\delta, r_{csa}, [\ell_{csa}, u_{csa}])$.
  if $(id, t) \notin \delta$ **then**
    return $(\delta \cup \{(id, t)\}, r_{csa} \cup r, [\min(\ell, \ell_{csa}), 1])$.
  end if
  Let $sa_2 = (id, r_{csa}, t, [\ell_{csa}, u_{csa}]) \in Rep(csa)$.
  Let $[\ell', u'] = combine(sa, sa_2)$.
  return $(\delta, r_{csa} \cup r, [\min(\ell', \ell_{csa}), \max(u', u_{csa})])$.

7.3.3 SPOT -Trees

We are now ready to define SPOT trees. Each node in a SPOT tree has a *capacity* K that describes the number of children that can be stored within a node. All data (i.e. SPOT atoms) are stored in leaf nodes and are considered to be the "children" of those nodes. SPOT trees differ from standard multi-dimensional indexes in that each internal node contains a composite SPOT atom instead of a minimal bounding rectangle. The composite atom *is entailed by* the composite atoms of the node's children (or the SPOT atoms of the node's children, when the children are leaves). For a given node nd in the SPOT tree, the associated composite SPOT atom will be referred to as $nd.csa$, while the children of that node will be in the set $nd.children$. If a node nd is a leaf, then it is stored as a SPOT atom in $nd.children$.

The *SpotInsert* algorithm (presented as Algorithm 12) inserts nodes into SPOT trees. It is similar to an R-tree insertion algorithm except that (i) instead of changing a node's minimal bounding rectangle, it applies Algorithm 11, (ii) the analog of node splitting in R-trees that is based on geometric considerations can be - but does not have to be - replaced by logical considerations, and (iii) the consideration of which child node to insert a SPOT atom into can be likewise based on logical considerations. The

algorithm ensures that the csa of any node is entailed by the csa's associated with its children.

Algorithm 12 Insert SPOT atom sa into the SPOT tree under node nd. Return $\emptyset$ if update happened without splitting, otherwise return the set of new nodes.

```
SpotInsert(sa, nd)
  if nd is not a leaf then
    nd.csa = UpdateCSA(nd.csa,sa)
    {Choose a child into which we'll insert the node.}
    sub ← chooseBestSubTree(sa,nd.children)
    newChildren ← SpotInsert(sub,nd)
    if newChildren ≠ ∅ then
      {sub was split below us}
      Remove sub from nd.children
      Add newChildren to nd.children.
    end if
  else
    nd.csa = UpdateCSA(nd.csa,sa)
    Add sa to nd.children. {nd is a leaf}
  end if
  {Check if we have too many items in this node}
  if |nd.children| > K then
    return splitNode(nd) {We need to split this node.}
  end if
  return ∅. {No split necessary.}
```

The insertion algorithm relies on two functions: chooseBestSubTree and splitNode. The chooseBestSubTree function is meant to tell which of the subnodes in the given set will be the "best" tree into which the SPOT atom sa should be inserted. There are many methods for making this decision. Here are a couple for node set $children$ and SPOT atom $sa = (id, r, t, [\ell, u])$:

- One can find $nd \in children$ such that an update with sa least affects the probability bounds of $nd.csa$.
- One can find $nd \in children$ with the fewest SPOT atoms beneath it.

- As in the case of an R-tree, one can find $nd \in children$ such that the size of the MBR of sa's region and the region in $nd.csa$ is minimal.

- One can find $nd \in children$ such that some weighted combination of the change in the probability bounds and the change in the region of the node is minimized.

In contrast, the splitNode function decides how to best divide a given node into two nodes. This involves re-inserting all SPOT atoms below the given node. The basic algorithm traverses the tree below the given node, collecting all the SPOT atoms to create two new nodes, and then iteratively inserts each of the collected atoms into the nodes, calling chooseBestSubTree each time to determine where to insert the next atom.

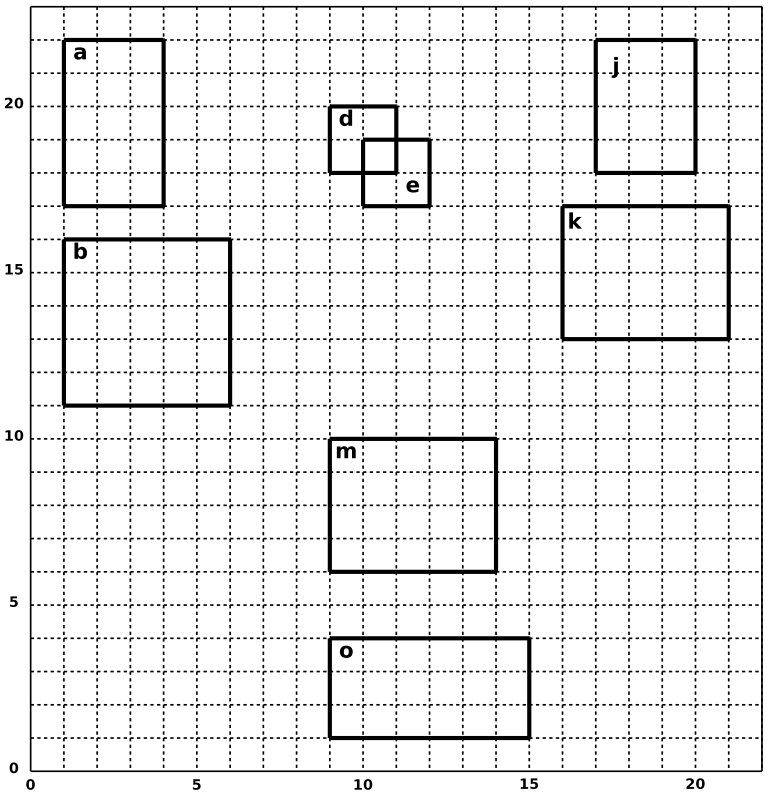
A natural question that arises is what K should be. In our implementation, the list of (id, t) pairs associated with a node is not stored in the node itself. Rather, the node contains a pointer to this list. In this case, K can be as large as $\lfloor \frac{psize - ptrsize}{nsize} \rfloor$ where $psize$ is the size of a disk page, $nsize$ is the size of a pointer to another node, and $ptrsize$ is the size of the pointer to the list of (id, t) pairs.

Example 27. *We show how to build a* SPOT *tree for the* SPOT *atoms shown in Fig. 7.1(a). We assume they are inserted in the order in which they are listed. We consider a* SPOT *tree where a node is considered full when it has 2 children, so splits occur after the insertion of a third child.*

After the first insertion, the tree has only one node, containing only one atom. The corresponding composite SPOT *-atom is straightforward (see Fig. 7.2(a)). The second atom is added in the same node, the corresponding composite* SPOT *-atom is computed accordingly. This is shown in Fig. 7.2(b). When the third atom is inserted,*

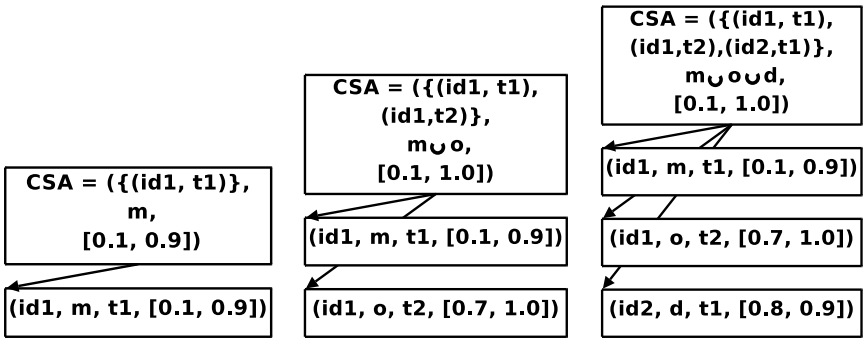
Vehicle	Area	time	min	max
id_1	m	t_1	10%	90%
id_1	o	t_2	70%	100%
id_2	d	t_1	80%	90%
id_2	e	t_1	90%	100%
id_3	j	t_1	70%	100%
id_4	k	t_2	80%	100%
id_5	a	t_1	10%	70%
id_5	b	t_1	60%	80%

(a) Information provided



(b) Corresponding areas

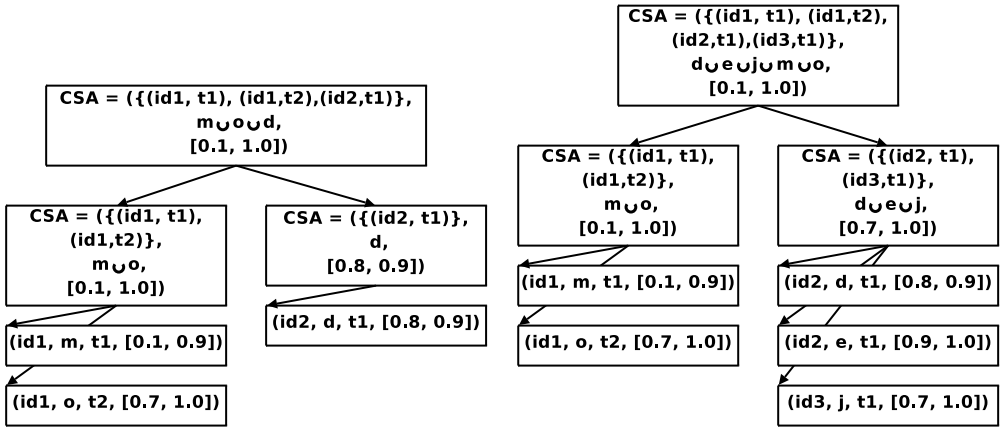
Figure 7.1: Localization of vehicles over an area



(a) After insertion of first atom

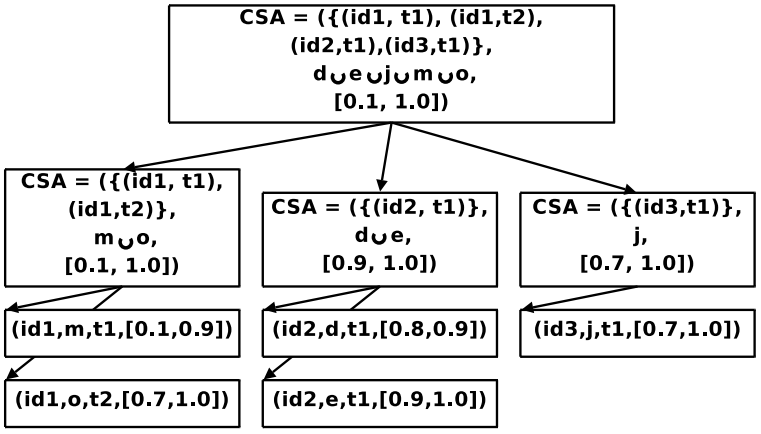
(b) After insertion of second atom

(c) Insertion of third atom: split needed



(d) After split

(e) Split needed at leaf level



(f) Split needed above root level

Figure 7.2: Different steps of insertion

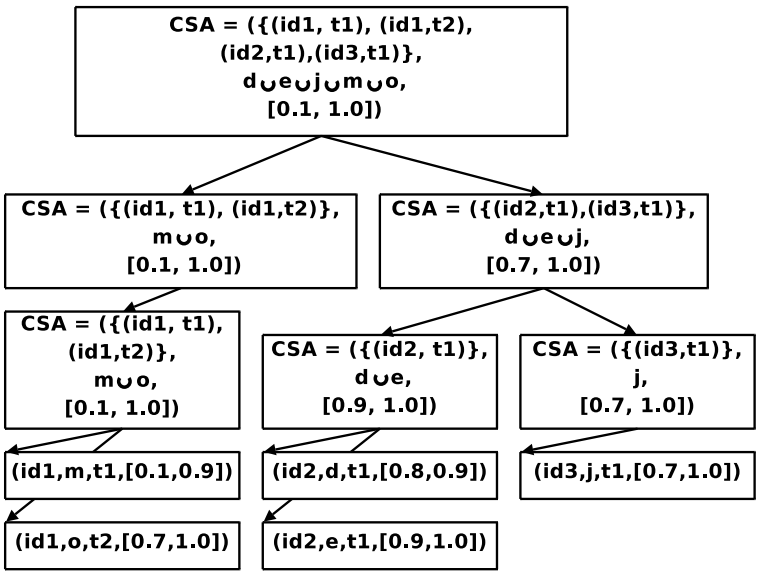


Figure 7.3: After root split.

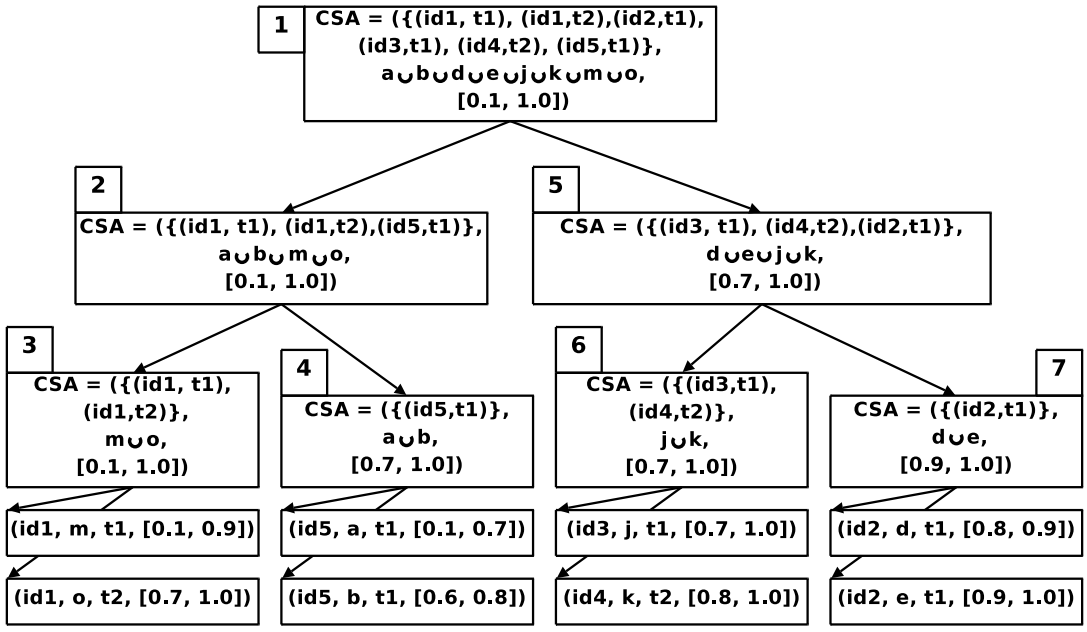


Figure 7.4: SPOT tree after insertions

the node becomes full and needs to be split (Fig. 7.2(c)). In this example, let us suppose that the splitting strategy used is as in an R-tree which tries to minimize the

area of the resulting MBRs. After the split, a new root is created, and composite atoms are computed as shown in Fig. 7.2(d). After two more insertions (Fig. 7.2(e)), the full node needs to be split, and we obtain the tree of Fig. 7.2(f), where the root node is full. (Note that after every split the ancestors of the node that is split need to be updated.) This root node in turn needs to be split, and the leaf nodes are distributed over the two new intermediate nodes.[1]*. We obtain the tree shown in Fig. 7.3. After all insertions, the resulting* SPOT *tree is shown in Fig. 7.4.*

The algorithm for answering queries utilizes the pruning steps described in Algorithm 13.

Theorem 15 (Correctness of Algorithm 13)**.** *For region q, bounds $[\ell, u]$, database $\mathcal{S}$ and associated* SPOT *tree $tree$, if $prune = pruneIdT(tree, q, [\ell, u])$ then for all $(id, t) \in prune$, $(id, q, t, [\ell, u]) \notin \mathcal{S}$.*

We now present algorithm *SpotQuery* (Algorithm 14) that may be used to answer atomic SPOT -queries. We detail an example query showing how the SPOT tree in Fig. 7.4 can be used to prune away potential answers. This algorithm is different from the corresponding algorithm for an R-tree. In particular, we do not limit our traversal of the tree to only those nodes whose MBR intersects the query region. The only time when we do not continue to a node's children is when the necessary conditions do not hold between the query region and bounds and the composite SPOT atom in that node.

[1] In this case, most R-tree algorithms remove and reinsert a large quantity of ground data in order to ensure near-optimality of the whole tree, as this is often more efficient than doing many local changes.

Algorithm 13 Given a SPOT tree node nd, a region q, and probability bounds $[\ell, u]$, return a set of (id, t) pairs such that $(id, q, t, [\ell, u])$ does not satisfy the composite SPOT atoms in nd or below nd in the SPOT tree.

pruneIdT(nd, q, $[\ell, u]$)

1: $pIdT \leftarrow \emptyset$ {$pIdT$ will be the set of prunable (id, t) pairs}
2: Let $ga = (id, r, t, [\ell', u'])$ be any member of $Rep(nd.csa)$.
3: **if** $NC(ga, (id, q, t, [\ell, u]))$ is not true **then**
4: Add all (id, t) pairs from $nd.csa$ to $pIdT$
5: **else**
6: **if** nd is a leaf **then**
7: {Check if any SPOT atoms rule out their (id, t).}
8: **for** Each atom $sa = (id, r, t, [\ell', u'])$ in $nd.children$ **do**
9: **if** $NC(sa, (id, q, t, [\ell, u]))$ is not true **then** add (id, t) to $pIdT$.
10: **end for**
11: **else**
12: {Check the children of this node.}
13: **for** Each child node c of nd **do**
14: $pIdT \leftarrow pIdT \cup$ pruneIdT(c,q,$[\ell, u]$)
15: **end for**
16: **end if**
17: **end if**
18: **return** $pIdT$

Theorem 16 (Correctness of Algorithm 14)**.** *For selection query* $(?id, q, ?t, [\ell, u])$ *and* SPOT *database* $\mathcal{S}$*, SpotQuery(*$\mathcal{S}, (?id, q, ?t, [\ell, u])$*) returns all* (id, t) *such that* $(id, q, t, [\ell, u]) \Subset \mathcal{S}$.

Proof. By Theorem 15 all SPOT atoms pruned by PruneIdT are not compatible with $\mathcal{S}$. The other SPOT atoms are checked explicitly. □

Example 28. *We consider the* SPOT *query* $(?id, q, ?t, [0.4, 0.7])$*, where* q *is the region shown in Fig. 7.5. The tree after all insertions was shown in Fig. 7.4. For ease of presentation, the composite regions of leaf nodes are shown in Fig. 7.5, and denoted with the letters* c*,* f*,* ℓ*, and* n*.*

Algorithm 14 For SPOT database $\mathcal{S}$ and selection query $(?id, q, ?t, [\ell, u])$, answer the query with pruning.

SpotQuery($\mathcal{S}$, $(?id, q, ?t, [\ell, u])$)
  IdT $\leftarrow$ all (id, t) pairs in $\mathcal{S}$
  STree $\leftarrow$ empty node
  for all $sa \in \mathcal{S}$ **do**
    SpotInsert(sa,STree)
  end for
  {Eliminate prunable pairs via Algorithm 13}
  candidateIdT $\leftarrow$ IdT $-$ pruneIdT(STree,$q, [\ell, u]$)
  answerIdT $\leftarrow \emptyset$
  for all $(id_i, t_i) \in$ candidateIdT **do**
    if $(id_i, q, t_i, [\ell, u]) \Subset \mathcal{S}^{id_i, t_i}$ **then**
      answerIdT $\leftarrow$ answerIdT $\cup (id_i, t_i)$
    end if
  end for
  return answerIdT

- *First, the pruning algorithm is called with node* 1 *(the root) as an argument. At this level, the query region* q *intersects the composite region* $a \cup b \cup d \cup e \cup j \cup k \cup m \cup o$*, hence the condition on line 3 is true (because the* NC *condition is not true). The pruning function is then called for every child on line 5.*

- *At this level, the same situation occurs for both nodes* 2 *and* 5*, (but the algorithm does a depth-first recursion, so it does not reach* 5 *at this point). The pruning function is then called for* 3 *and* 4*.*

- *At node* 3*, let* $ga = (id_1, n, t_1, [.1, 1])$*. Here, the first condition of the definition of NC applies on line* 3*, but since* $0.4 \leq 1 - 0.1$*, pruning cannot be done. But we will prune* (id_1, t_2) *when checking the children of this leaf node on lines 9-11.*

- *Node* 4 *(composite atom* $(\{(id_5, t_1)\}, c, [0.7, 1])$*), gives the same result as node* 1*. But on lines 9-11 we consider* $(id_5, a, t_1, [0.1, 0.7])$ *and* $(id_5, b, t_1, [0.6, 0.8])$*.*

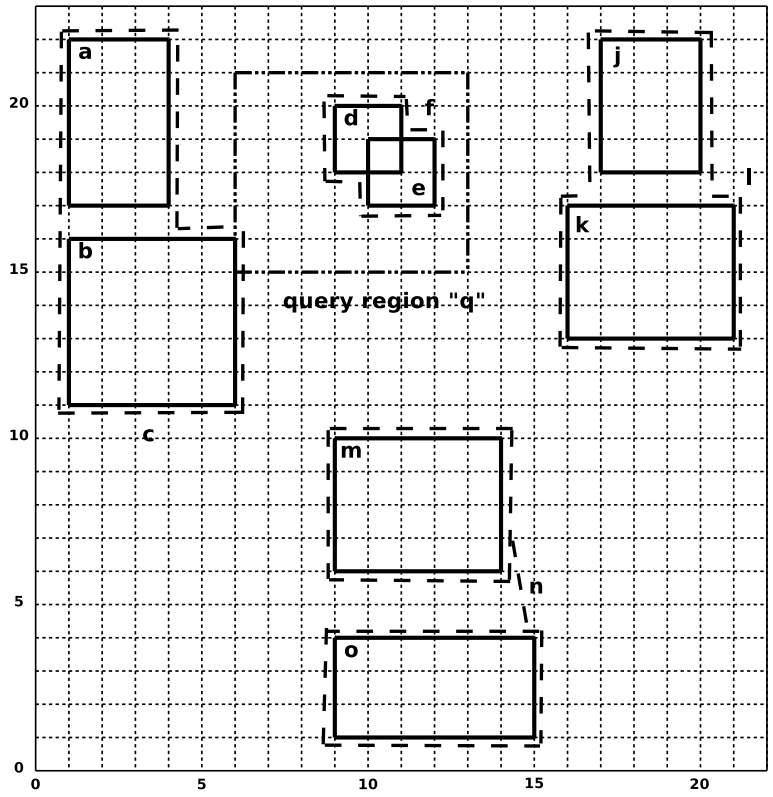


Figure 7.5: Area for pruning mechanism

Pruning cannot be done for either atom.

- *Now consider node* 6 *and composite* SPOT *atom* $(\{(id_3, t_1), (id_4, t_2)\}, l, [0.7, 1])$. *The query region does not intersect the composite region* l*, thus the condition of line 3 does not hold, because the* NC *condition is true. Thus we can directly can prune* (id_3, t_1) *and* (id_4, t_2).

- *For node* 7*, the composite* SPOT *atom* $(\{(id_2, t_1)\}, f, [0.9, 1])$ *is contained in the query region* q*. This triggers the third condition of* NC*, (*$0.7 \geq 0.9$ *is false), hence this node is pruned.*

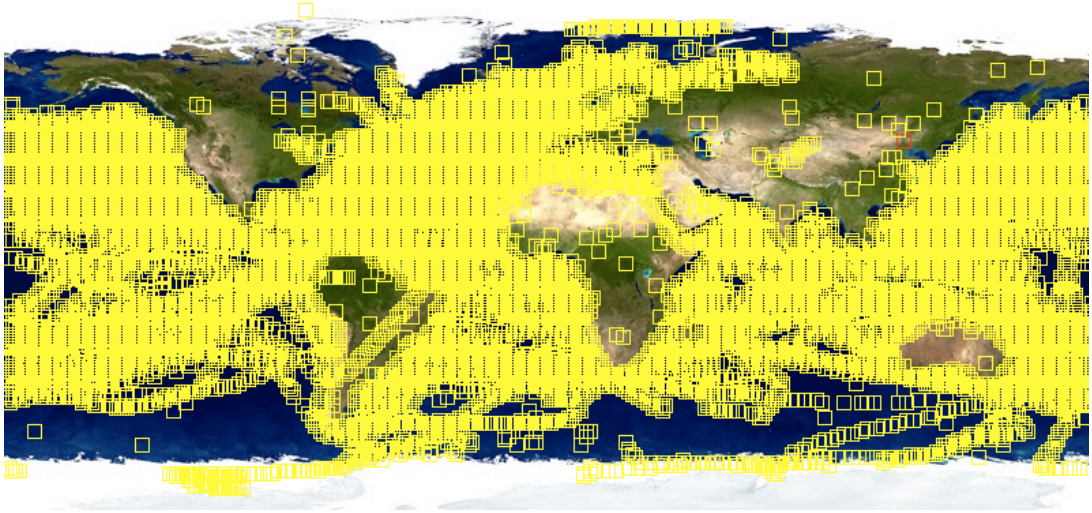


Figure 7.6: A visualization of the DoD Ship location dataset. Each rectangle represents a separate atom. Notice how many of the rectangles are clearly erroneous, putting the ship on land or in the middle of the ocean with no path to or from that location. For instance, the red rectangle in north-eastern China is clearly an error.

Cautious Selection

While the SPOT tree was introduced with optimistic selection as an example, it is also useful for cautious selection. Since the optimistic answer set is a subset of the cautious answer set for any query, any answer pruned from an optimistic query must also be pruned from a cautious query. Therefore the entire SPOT tree methodology can be directly applied to cautious query answering.

7.3.4 SPOT Tree: Experimental Results

Tests of the SPOT tree were conducted with real world ship location data provided by the United States Department of Defense. The ships tracked in the dataset were non-military, and the data sources were all public-sector. The data was not necessarily

reliable: there was error from the GPS systems used to derive the ship locations, a lack of consequences for inaccurate reporting, a lack of compliance to specified data schemas, and potential software bugs in parsers and data extraction algorithms. The locational data provided gave a latitude and longitude for a ship's location, the ship's ID, and a time stamp. These tuples were converted to SPOT atoms by using the specified ship ID and time as id and t, and centering a 10×10 rectangle r around the specified latitude and longitude to create the atom $(id, r, t, [0.3, 0.8])$. The probability interval $[0.3, 0.8]$ was chosen to allow three disjoint locations to be reported for the same ship at the same time point, and to ensure that no tuple is trusted more than 80% of the time (due to the myriad of potential error sources in the system). Figure 7.6 is a visualization of the dataset.

In the first batch of experiments, we examined the scalability of SPOT trees with this dataset. Figure 7.7 shows performance of SPOT trees as the number of atoms increases to 250,000. Three different versions of SPOT trees were tried, depending on the algorithm used to check $\langle id, t \rangle$ pairs not pruned by the tree. The fact that the three curves are nearly identical suggests that the tree is capable of pruning substantial numbers of potential pairs. The SPOT trees created for these experiments attempt to minimize the size of the regions in the MBRs.

In a second experiment, we test the SPOT tree with various implementations of the $chooseBestSubTree$ function. The flavors we test are:

- Random SPOT Tree: $chooseBestSubTree(nd, atm)$ chooses a random child of nd for insertion.

- Balanced SPOT Tree: $chooseBestSubTree(nd, atm)$ chooses a child of nd with minimal depth.

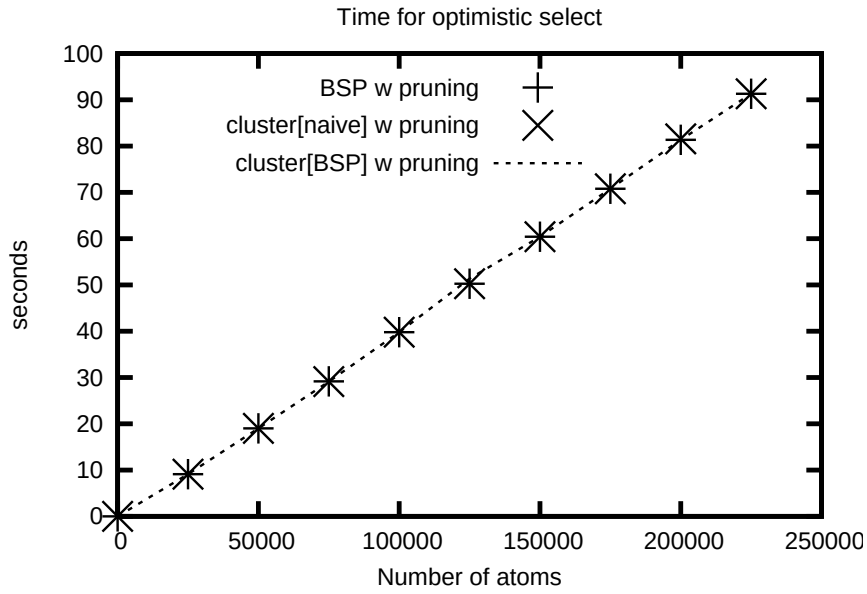


Figure 7.7: Scaling experiments with SPOT trees.

- ProbMin SPOT Tree: $chooseBestSubTree(nd, atm)$ chooses a child of nd that has minimal change to its probability bounds after update with atm.

- RegionMin SPOT Tree: $chooseBestSubTree(nd, atm)$ chooses a child of nd that has minimal change to its region after update with atm.

We test these against the control case where no tree is used in both optimistic and cautious selection. In these experiments, a random 3×3 region r_q is chosen, and both a cautious and an optimistic query of the form $(?id, r_q, ?t, [0.9, 1])$ is run on a random subset of the database. The timing information is averaged over 300 trials. The results are shown as Figures 7.9 and 7.8.

In these experiments, the SPOT tree techniques outperform non-SPOT tree techniques. Of further interest is the fact that the ProbMin insertion heuristic performs best when the number atoms is less than about 400, and RegionMin does best when the number of atoms is greater than about 400. As expected, the random SPOT tree

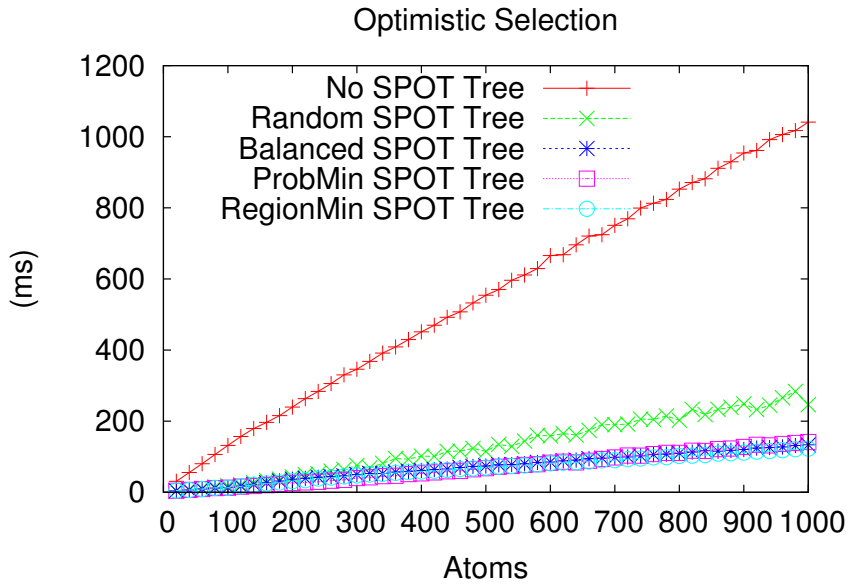


Figure 7.8: Experiments showing the effectiveness of the SPOT tree with optimistic queries of the form $(?id, r, ?t, [0.9, 1])$.

performs worst among SPOT tree techniques.

7.4 Inscribing and Circumscribing Convex Regions

Another approach to improving query efficiency relies on the linear programming aspect of consistency checking algorithms. The space of solutions to the linear programs involved in the SPOT framework is bounded and convex, and we will see this to allow us many pre-computation options. However, not all techniques produced here will require pre-computation: simply by saving solutions to previous computations of the linear program, one can increase the performance of later computations. This section details these techniques.

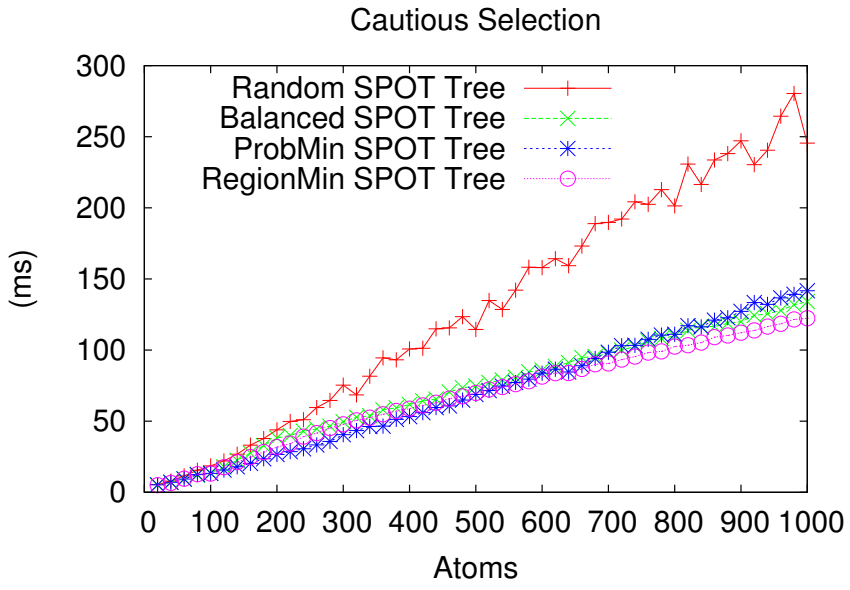


Figure 7.9: Experiments showing the effectiveness of the SPOT tree with cautious queries of the form $(?id, r, ?t, [0.9, 1])$.

7.4.1 Formalism

In this section we will use $n-tuples$ with $n = |\mathcal{L}|$ to represent the probability assignment of an interpretation for each location $L \in \mathcal{L}$ for a fixed $\langle id, t\rangle$ pair. We write $\overline{v}$ for such an $n-tuple$ where $\overline{v}_L$ is the component of $\overline{v}$ for $L \in \mathcal{L}$. Let $\mathcal{P}(\mathcal{S}, id, t) = \{\, \overline{v} \in [0,1]^n \mid \overline{v}$ is a solution to $LC(\mathcal{S}, id, t)\,\}$, which is a polytope in $[0,1]^n$. Clearly, every $\overline{v} \in \mathcal{P}(\mathcal{S}, id, t)$ corresponds to an interpretation $I \in \mathcal{I}(\mathcal{S})$ such that for each $L \in \mathcal{L}$, $I(id, L, t) = \overline{v}_L$, i.e., the $L-component$ of $\overline{v}$.

Next, for a query $Q = (?id, r, ?t, [\ell, u])$, we define

$$\mathcal{Q}(r, \ell, u) = \{\, \overline{v} \in [0,1]^n \mid \sum_{L \in r} v_L \geq \ell \text{ and } \sum_{L \in r} v_L \leq u\,\}.$$

$\mathcal{Q}(r, \ell, u)$ is also a convex polytope.

In our theorems we will typically deal with convex regions in an n-dimensional

space that are either included in or that include $\mathcal{P}(\mathcal{S}, id, t)$. Our results hold for any regions that satisfy the inclusion criteria; however, in the implementation, in order to get the best results, we will be using included regions that are as large as possible and including regions that are as small as possible. Hence we simply use the terminology everywhere that $\mathcal{R}_1$ is *inscribed* in $\mathcal{R}_2$, or $\mathcal{R}_2$ *circumscribes* $\mathcal{R}_1$, just in case $\mathcal{R}_1 \subseteq \mathcal{R}_2$.

The following Corollary follows from Theorem 1.

Corollary 5. *Given a* SPOT *database* $\mathcal{S}$ *and query* $q = (?id, r, ?t, [\ell, u])$*, for each pair* $\langle id, t \rangle$ *in* $\mathcal{S}$*,*

i) $(id, r, t, [\ell, u]) \Subset \mathcal{S}$ *iff* $\mathcal{P}(\mathcal{S}, id, t) \cap \mathcal{Q}(r, \ell, u) \neq \emptyset$.

ii) $\mathcal{S} \models (id, r, t, [\ell, u])$ *iff* $\mathcal{P}(\mathcal{S}, id, t) \subseteq \mathcal{Q}(r, \ell, u)$.

For a set of locations r and a point $\overline{v}$ in the n-dimensional space, we define the *probability mass* in $\overline{v}$ w.r.t. r as $p(\overline{v}, r) = \sum_{L \in r} \overline{v}_L$. For a convex region $\mathcal{R}$ and a set of locations r, we define $inf(\mathcal{R}, r) = \{\overline{v} \in \mathcal{R}$ such that $p(\overline{v}, r)$ is minimum $\}$, and $sup(\mathcal{R}, r) = \{\overline{v} \in \mathcal{R}$ such that $p(\overline{v}, r)$ is maximum $\}$. In the following, we will write $p(inf(\mathcal{R}, r))$ (resp., $p(sup(\mathcal{R}, r))$), for denoting the minimum (resp., maximum) value of the probability mass in $\mathcal{R}$ w.r.t. r.

Next, given a set $V = \{\overline{v}_1, \ldots, \overline{v}_k\}$ of k points in the n-dimensional space, we will denote the convex envelope of V as $convEnv(V) = \{ \sum_{i=1}^{k} \alpha_i \overline{v}_i \mid \overline{v}_i \in V,\ \alpha_i \in [0, 1],$ and $\sum_{i=1}^{k} \alpha_i = 1\}$.

The following theorem provides a simple way of checking if a convex region intersects with, or is contained in, a query region. It states that these two relationships can be checked by considering only the convex envelope of two appropriate points in the region, or just an appropriate numerical interval.

Theorem 17. *For a convex region* $\mathcal{R}$ *and query region* $\mathcal{Q}(r, \ell, u)$*,*

i) $\mathcal{R} \cap \mathcal{Q}(r, \ell, u) \neq \emptyset \Leftrightarrow convEnv(\{inf(\mathcal{R}, r) \cup sup(\mathcal{R}, r)\}) \cap \mathcal{Q}(r, \ell, u) \neq \emptyset \Leftrightarrow [p(inf(\mathcal{R}, r)), p(sup(\mathcal{R}, r))] \cap [l, u] \neq \emptyset$.

ii) $\mathcal{R} \subseteq \mathcal{Q}(r, \ell, u) \Leftrightarrow convEnv(\{inf(\mathcal{R}, r) \cup sup(\mathcal{R}, r)\}) \subseteq \mathcal{Q}(r, \ell, u) \Leftrightarrow [p(inf(\mathcal{R}, r)), p(sup(\mathcal{R}, r))] \subseteq [l, u]$.

Inscribed and circumscribing regions for the polytope $\mathcal{P}(\mathcal{S}, id, t)$ will be used to answer selection queries under the cautious and optimistic semantics.

7.4.2 Cautious Semantics

The following theorem provides two sufficient conditions which will be exploited for answering cautious queries. Specifically, for a given pair $\langle id, t \rangle$, when one of these two conditions is satisfied no optimization programs like that in Theorem 1 have to be solved. In that case, we say that the pair $\langle id, t \rangle$ is *pruned* via these sufficient conditions. The first condition in the theorem ensures that the pair belongs to the answer, whereas the second condition ensures that it does not.

Theorem 18. *Let $\mathcal{S}$ be* SPOT *database and $q = (?id, r, ?t, [\ell, u])$ a query. For each pair $\langle id, t \rangle$ in $\mathcal{S}$, let $\mathcal{R}_{ins}(\mathcal{S}, id, t)$ and $\mathcal{R}_{cir}(\mathcal{S}, id, t)$ be two convex regions such that $\mathcal{R}_{ins}(\mathcal{S}, id, t) \subseteq \mathcal{P}(\mathcal{S}, id, t) \subseteq \mathcal{R}_{cir}(\mathcal{S}, id, t)$.*

i) If $\mathcal{R}_{cir}(\mathcal{S}, id, t) \subseteq \mathcal{Q}(r, l, u)$ then $(id, r, t, [\ell, u])$ is in the cautious answer *to q.*

ii) If $\mathcal{R}_{ins}(\mathcal{S}, id, t) \not\subseteq \mathcal{Q}(r, l, u)$ then $(id, r, t, [\ell, u])$ is not in the cautious answer *to q.*

By Theorem 17, the two conditions in the above theorem can be checked by using the intervals $\mathcal{I}_{\mathcal{R}} = [p(inf(\mathcal{R}, r)), p(sup(\mathcal{R}, r))]$, where $\mathcal{R} \in \{\mathcal{R}_{ins}, \mathcal{R}_{cir}\}$. We now consider some specific kinds of regions and show conditions equivalent to that of

Theorem 18 for these specific regions, which are obtained by rewriting the numeric interval $\mathcal{I}_{\mathcal{R}}$ for the considered cases.

Hyper-rectangles, also called boxes, are probably the most common kind of objects used for bounding regions. For example, minimum bounding rectangles (MBRs) are used in R-trees for spatial indexing [7, 8, 44]. When we consider box regions, the interval $\mathcal{I}_{\mathcal{R}}$ can be easily related to the box sides. Given an n-dimensional space, a box $\mathcal{B}$ is a (convex) region defined by the Cartesian product $I_1 \times I_2 \times \cdots \times I_n$, where for each interval I_j with $j \in [1..n]$, $I_j \subset \mathbb{R}$. We will denote by $\ell(\mathcal{B}, L)$ and $u(\mathcal{B}, L)$ the lower and upper bounds of $\mathcal{B}$ on the dimension corresponding to location L.

Corollary 6. *Let $\mathcal{S}$ be* SPOT *database $\mathcal{S}$ and $q = (?id, r, ?t, [\ell, u])$ a query. Let $\mathcal{B}_{ins}$ and $\mathcal{B}_{cir}$ be two boxes such that $\mathcal{B}_{ins} \subseteq \mathcal{P}(\mathcal{S}, id, t) \subseteq \mathcal{B}_{cir}$. Then,*

i) if $[lower(\mathcal{B}_{cir}), upper(\mathcal{B}_{cir})] \subseteq [\ell, u]$, then $(id, r, t, [\ell, u])$ is in the cautious answer *to q, and*

ii) if $[lower(\mathcal{B}_{ins}), upper(\mathcal{B}_{ins})] \not\subseteq [\ell, u]$, then $(id, r, t, [\ell, u])$ is not in the cautious answer *to q, where*

- *$lower(\mathcal{B}) = \sum_{L \in r} \ell(\mathcal{B}, L)$, and*
- *$upper(\mathcal{B}) = \sum_{L \in r} u(\mathcal{B}, L)$.*

Theorem 1 gives us an exact method for computing cautious answers to a selection query. This method entails that, for each $\langle id, t \rangle$ pair in the database, two optimization problems must be solved in order to decide if the pair belongs to the answer. On the other hand, Theorem 18 gives us a strategy that can be exploited better when the database is not updated frequently. Assume that, for each pair $\langle id, t \rangle$ in the database and location L in $\mathcal{L}$, the lower and upper bounds $\ell(\mathcal{B}_{ins}, L)$, $u(\mathcal{B}_{ins}, L)$, $\ell(\mathcal{B}_{cir}, L)$ and

$u(\mathcal{B}_{cir}, L)$ for the boxes $\mathcal{B}_{ins}$ and $\mathcal{B}_{cir}$ are known. Then, applying Corollary 6, the cautious answer to a query will be computed in constant time.

Box regions belongs to the class of regular polytopes. A non-regular polytope can be easily described by specifying a set of points such that its convex envelope results in the polytope. We use inscribed regions specified by the convex envelope of a set of points in our experiments. The following corollary identifies the pruning conditions for these kinds of regions.

Corollary 7. *Let $\mathcal{S}$ be a* SPOT *database and $q = (?id, r, ?t, [\ell, u])$ a query. Given two sets of points V_1, V_2 such that $convEnv(V_1) \subseteq \mathcal{P}(\mathcal{S}, id, t) \subseteq convEnv(V_2)$,*

i) if $[lower(V_2), upper(V_2)] \subseteq [\ell, u]$ then $(id, r, t, [\ell, u])$ is in the cautious answer *to q, and*

ii) if $[lower(V_1), upper(V_1)] \not\subseteq [\ell, u]$ then $(id, r, t, [\ell, u])$ is not in the cautious answer *to q where*

- *$lower(V) = \min_{\overline{v} \in V} p(\overline{v}, r)$, and*
- *$upper(V) = \max_{\overline{v} \in V} p(\overline{v}, r)$.*

7.4.3 Optimistic Semantics

Analogously to the case of cautious selection, the following theorem provides pruning conditions for answering optimistic queries by exploiting inscribed and circumscribing regions.

Theorem 19. *Let $\mathcal{S}$ be* SPOT *database $\mathcal{S}$ and $q = (?id, r, ?t, [\ell, u])$ a query. For each pair $\langle id, t \rangle$ in $\mathcal{S}$, let $\mathcal{R}_{ins}(\mathcal{S}, id, t)$ and $\mathcal{R}_{cir}(\mathcal{S}, id, t)$ be two convex regions such that $\mathcal{R}_{ins}(\mathcal{S}, id, t) \subseteq \mathcal{P}(\mathcal{S}, id, t) \subseteq \mathcal{R}_{cir}(\mathcal{S}, id, t)$. Then,*

i) $(id, r, t, [\ell, u])$ *is in the* optimistic answer *to* q *if* $\mathcal{R}_{ins}(\mathcal{S}, id, t) \cap \mathcal{Q}(r, \ell, u) \neq \emptyset$

ii) $(id, r, t, [\ell, u])$ *is not in the* optimistic answer *to* q *if* $\mathcal{R}_{cir}(\mathcal{S}, id, t) \cap \mathcal{Q}(r, \ell, u) = \emptyset$

Reasoning as in the previous section, we derive pruning conditions for optimistic selection for inscribed and circumscribing regions consisting of boxes or convex envelopes of sets of points. For boxes, the pruning conditions of Theorem 19 can be checked considering the intersection between the probability interval stated in the query and the interval $[lower(\mathcal{B}), upper(\mathcal{B})]$, with $\mathcal{B} \in \{\mathcal{B}_{ins}, \mathcal{B}_{cir}\}$, introduced in Corollary 6. In case the (inscribed or circumscribing) region is specified by a set V of points, the pruning conditions can be checked by using interval $[lower(V), upper(V)]$ introduced in Corollary 7.

7.4.4 Computing Inscribed and Circumscribed Regions

In this section we introduce some strategies for incrementally computing inscribed and circumscribed regions during the lifetime of the database. In fact, we assume that initially there are no inscribed or circumscribing regions. These can be constructed using answers to queries evaluated with the naive algorithms when no pruning is possible by means of the current available regions. Let $q = (?id, r, ?t, [l, u])$ be a query, and let $\overline{s}_o$ be a solution of the linear program in Theorem 1.i) when q is evaluated under the optimistic semantics. Let $\overline{s}_\ell$ and $\overline{s}_u$ be the solutions of the optimization programs in Theorem 1.ii) when q is evaluated under the cautious semantics. Solutions $\overline{s}_\ell$ and $\overline{s}_u$ yield, respectively, values ℓ' and u'.

Consider an $\langle id, t \rangle$ pair. If the cautious answer to q is asked and the answer is '*yes*', then $\overline{s}_\ell$ and $\overline{s}_u$ can be added to the set of points V that specifies the inscribed region $convEnv(V)$. On the other hand, if the answer to q is '*no*', there are two cases. First,

if $u' > u$ (resp., $l' < l$) we can add $\overline{s}_u$ (resp., $\overline{s}_l$) to the set of points V constructing an inscribed region. Second, if $u' \leq u$ (resp., $l' \geq l$), then the inequalities $\sum_{L \in r} v_L \leq u$ (resp., $\sum_{L \in r} v_L \geq l$) describe a hyperplane bounding the polytope $\mathcal{P}(\mathcal{S}, id, t)$. In the case that $r = \{L\}$, this inequality represents the upper bound $u(\mathcal{B}, L)$ (resp., the lower bound $\ell(\mathcal{B}, L)$) on the dimension L for a bounding box $\mathcal{B}$.

Similarly, if the optimistic answer to q is asked and the answer is '*yes*', then $\overline{s}_o$ can be added the set of points V that specifies the inscribed region. Whereas, if the optimistic answer to q is '*no*', then the equations $\sum_{L \in r} v_L \geq l$ (or $\sum_{L \in r} v_L \leq u$) describe a hyperplane bounding the polytope $\mathcal{P}(\mathcal{S}, id, t)$.

Circumscribing regions can be also obtained exploiting the concept of *composite atom* introduced in [38]. Indeed, a composite atom ca for a set $\mathcal{S}$ of SPOT atoms is such that its interpretations subsume the interpretations of $\mathcal{S}$. As interpretations corresponds to points of the polyhedron associated with $\mathcal{S}$, a composite atom represents a bounding region.

7.4.5 Multiple Inscribed Regions for Cautious Selection

Inscribed regions are used for cautious selection in order to prune those $\langle id, t \rangle$ pairs which do not belong to the answer of a given query. Although the concept of minimal bounding region has been extensively studied and applied in several contexts, such as spatial indexing and data mining, we are not aware of any work where inscribed region are used for similar issues. In working with bounded and bounding regions for a given object, a natural question arises: what is the *best* region to use? To answer this question we need to take into account both the geometry of inscribed (or circumscribing) regions and a parameter for measuring the effectiveness of this region. Boxes and ellipsoids are commonly used; their effectiveness is measured by the volume of the

region. For instance, minimum volume bounding rectangles are used for spatial indexing and as basic component of R-trees, whereas in the data mining area, minimum volume ellipsoids are used for identifying data outliers.

A nice property of ellipsoids is that there is a unique maximum volume ellipsoid inscribed in a convex polytope. This is not the case for other common objects. For instance, there may be more than one maximum volume box inscribed in a convex region (as an example, consider maximum area rectangles in a regular pentagon). Although the intuition could say that increasing the volume of an inscribed region results in a more effective pruning strategy, the following example shows that this is not always true.

Example 29. *Let $\mathcal{S}$ be the* SPOT *database*

$$\{(id, \{L_1\}, t, [0.2, 0.5]), (id, \{L_2\}, t, [0.1, 0.6]), (id, \{L_1, L_2\}, t, [0.2, 0.8])\},$$

and $\mathcal{L}$ the set of locations $\{L_1, L_2, L_3\}$. The query is $q = (?id, \{L_1, L_2\}, ?t, [0.4, 0.8])$. Let E_1 and E_2 be two ellipsoids inscribed in $\mathcal{P}(\mathcal{S}, id, t)$ such that E_1 has maximum volume. The projection into the subspace $\{L_1, L_2\}$ of E_1 and E_2 is shown in Figure 7.10. The projection of the query region $\mathcal{Q}(\{L_1, L_2\}, 0.4, 0.8)$ is represented by the area between the two parallel oblique lines.

Figure 7.10 also shows the segments I_1 and I_2 representing, respectively, the pruning intervals $[p(inf(E_1, \{L_1, L_2\})), p(sup(E_1, \{L_1, L_2\}))]$ and $[p(inf(E_2, \{L_1, L_2\})), p(sup(E_2, \{L_1, L_2\}))]$ (see Theorems 17 and 18). It is easy to see that, although ellipsoid E_2 has a volume smaller than that of E_1, it is associated with a pruning interval I_2 whose length is greater than that of I_1, thus using E_2 results in a more effective pruning strategy for the query $q = (?id, \{L_1, L_2\}, ?t, [0.4, 0.8])$. On the other hand, ellipsoid E_1 results better than E_2 when the query $(?id, \{L_1\}, ?t, [0.4, 0.8])$ (or

$(?id, \{L_2\}, ?t, [0.4, 0.8]))$ is considered, since the projection of E_1 on axis L_1 (or L_2) is greater than the projection of E_2 on the same axes.

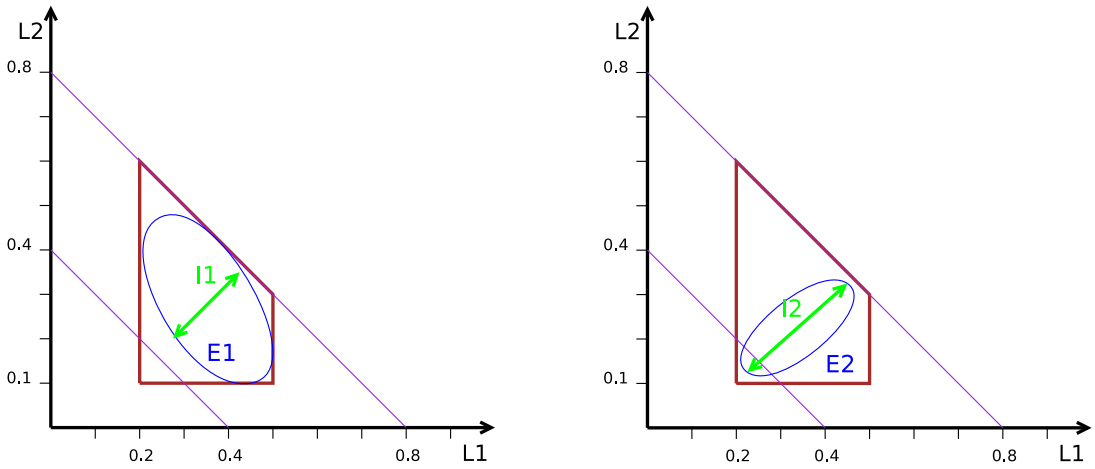


Figure 7.10: Inscribed ellipsoids for cautious selection.

The example above suggest that the optimality of an inscribed region for pruning depends on the locations considered in the query: an optimal inscribed region may become sub-optimal by changing the projection on different locations in the query. Moreover, in order to obtain more effective pruning, the quantity to be maximized is the length of the pruning intervals, not the volume of the inscribed regions.

Several inscribed regions can be used together to obtain more efficient pruning strategies. The following theorem shows how to combine them in order to maximize the pruning.

Theorem 20. *Let $\mathcal{S}$ be* SPOT *database $\mathcal{S}$ and $q = (?id, r, ?t, [\ell, u])$ a query. For each pair $< id, t >$ in $\mathcal{S}$, let $\{\mathcal{R}_1, \ldots \mathcal{R}_k\}$ be a set of convex regions such that $\mathcal{R}_i \subseteq \mathcal{P}(\mathcal{S}, id, t)$ with $i \in [1..k]$. Then, the greatest interval for pruning atoms $(id, r, t, [\ell, u])$ which do not belong to the* cautious answer *to q is*

$$[\,\ell', u'] = [\min\{p(inf(\mathcal{R}_i, r)) \mid i \in [1..k]\}, \max\{p(sup(\mathcal{R}_i, r)) \mid i \in [1..k]\}\,].$$

7.4.6 Algorithms

The first algorithm we introduce is the pre-computation algorithm (Algorithm 15), which creates $(V_{id,t}, csa_{id,t})$ for each $\langle id, t \rangle$. $V_{id,t}$ is a set of solutions to $LP(\mathcal{S}, id, t)$ that serves as the inscribed region. $csa_{id,t}$ is a composite atom (Definition 49) that serves as an enclosing region. Composite atoms define a circumscribing region for the satisfying interpretations and therefore a region circumscribing the solutions of $LC(\mathcal{S}, id, t)$.

Algorithm 15 For SPOT database $\mathcal{S}$, compute the inscribed region $V_{id,t}$ (represented as k separate solutions to $LP(\mathcal{S}, id, t)$) and the enclosing regions $csa_{id,t}$ (represented as a composite SPOT atom (Definition 49)), and return both as $(V_{id,t}, csa_{id,t})$.

Create empty composite atom $csa_{id,t}$.
for $sa \in \mathcal{S}$ **do**
  $csa_{id,t} =$ UpdateCSA(csa, sa) (Algorithm 11).
end for
for i from 1 to k **do**
  Compute random solution $\bar{v}$ to $LP(\mathcal{S}, id, t)$, add $\bar{v}$ to $V_{id,t}$
end for
return $(V_{id,t}, csa_{id,t})$.

Pruning according to the inscribed region is accomplished via Algorithm 16. This algorithm applies the insights of Corollary 7: it returns OUT according to part (ii) of that corollary.

We know how to update composite regions and when they are applicable for pruning via Theorem 12 and Algorithm 11.

Algorithm 17 puts all these pieces together and performs cautious selection with pruning. This is the algorithm tested in the following experiments. Notice that one important variable will be the size of $V_{id,t}$, or the number of solutions provided to the pruning algorithm. We will see this to have an impact on the algorithm's performance.

Algorithm 16 For cautious query $q = (?id, r, ?t, [\ell, u])$, set of solutions V to $LP(\mathcal{S}, id, t)$, for some pair $\langle id, t\rangle$, $pruneCautious$ returns OUT if $(id, r, t, [\ell, u])$ is not in the query's cautious answer set, and $UNKNOWN$ otherwise.

pruneCautious($q, V = \{\bar{v}_1, \ldots, \bar{v}_n\}$)
  Let $(?id, r_q, ?t, [\ell_q, u_q]) = q$ be the query.
  for $\bar{v}_i \in V$ **do**
    If $p(\bar{v}_i, r_q) \notin [\ell_q, u_q]$ then **return** OUT
  end for
  return $UNKNOWN$.

Algorithm 17 Computes the set of $\langle id, t\rangle$ pairs that are in the cautious answer to the query q on SPOT database $\mathcal{S}$ using precomputed information $\{V_{id,t}\}$.

pruneCautiousSelect($q, \{(V_{id,t}, csa_{id,t}\}, \mathcal{S}$)
  Let $(?id, r_q, ?t, [\ell_q, u_q]) = q$ be the query.
  $ANS = \emptyset$
  for $\langle id, t\rangle$ **do**
    if $NC(csa_{id,t}, (id, r_q, t, [\ell_q, u_q]))$ **then**
      if pruneCautious($q_r, V_{id,t}$) = IN **then**
        $ANS = ANS \cup \{(id, r_q, t, [\ell_q, u_q])\}$
      else
        Check if $(id, r_q, t, [\ell, u]) \Subset \mathcal{S}$ via linear programming.
      end if
    end if
  end for

7.4.7 Experiments

Algorithms Used

We produced a prototype implementation within a SPOT framework for experimentally testing the effectiveness of our pruning techniques. In our implementation, we took a database $\mathcal{S}$ and ran Algorithm 15 to create our pre-computed data $\{(V_{id,t}, csa_{id,t})\}$. Each $V_{id,t}$ contains a set of k solutions to $LC(\mathcal{S}, id, t)$ for an $\langle id, t\rangle$ pair. The larger k is, the larger the inscribed region will be and the larger the pruning capabilities of such a region. However, larger k also implies larger up-front costs for

computing the inscribed region.

Section 7.3 explains how SPOT trees can be used for optimistic selection. However, as those pruning techniques prune away potential answers to optimistic selection queries, and all answers to a cautious query are, by definition, also answers to the corresponding optimistic query, we can use these techniques to perform pruning for cautious queries also. We include these results in our experiments and label such data by "SPOT *tree*".

Random Artificial Data

In this experiment, we generate random artificial data and test the running times for optimistic and cautious selection. We use inscribed volumes represented by k solutions for pruning, and we vary k. Since this sort of pruning is performed once per $\langle id, t \rangle$ pair, we use only one such pair. SPOT atoms are generated via a random process. $\mathcal{L}$ is 100×100. The region r is a rectangle whose width and height are both randomly chosen integers between 1 and 20, while its x and y coordinates are chosen uniformly at random. For the probability bounds, a random draw was taken from $[0, 1]$ for u and another random draw was taken from $[0, u]$ for l. In order to ensure the database's consistency after all atoms are created, the lower bound of every atom is divided by the number of atoms.

The query's bounds $[\ell, u]$ are chosen in another random process: the smaller of two random draws from $[0, 1]$ is ℓ, while the large draw is u. Small queries use a randomly chosen 2×2 region, while large query regions are a randomly chosen 30×30 region.

We measured the running time of cautious and optimistic selection, varying the number of generated atoms and the number of solutions stored by the database.

Cautious selection queries show substantial benefit from pre-computing an in-

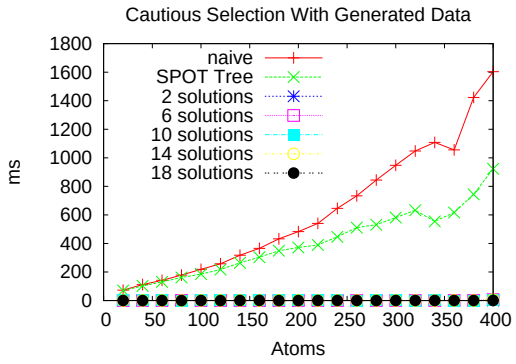


Figure 7.11: Cautious query times with randomly generated data and small query region. Data points are an average of 300 trials.

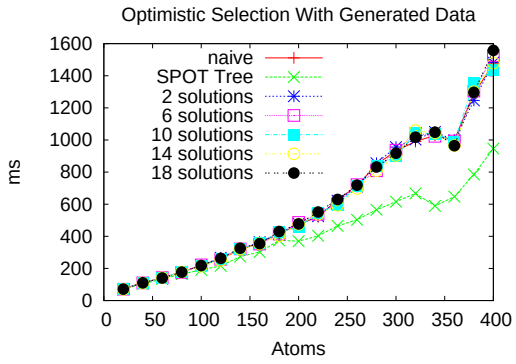


Figure 7.12: Optimistic query times with randomly generated data and small query region. Data points are an average of 300 trials.

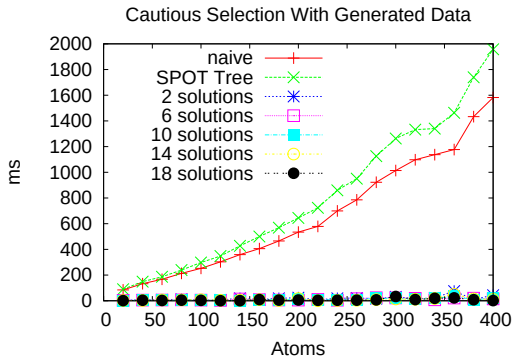


Figure 7.13: Cautious query times with randomly generated data and large query region. Data points are an average of 300 trials.

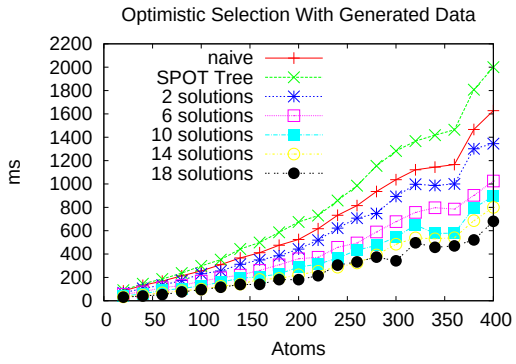


Figure 7.14: Optimistic query times with randomly generated data and large query region. Data points are an average of 300 trials.

scribed and a circumscribed region. The given $\langle id, t \rangle$ pair could be pruned immediately in almost every case, resulting in running times close to the x-axis whenever inscribed regions were stored. See Figures 7.11 and 7.13. This is partially due to the fact that for most of the randomly generated datasets, the cautious query returned an empty answer set. We note that while the SPOT tree is also able to provide pruning when the query region is small, the overhead of the tree actually makes it perform worse than naïve checking for large query regions. In these cases, the tree rarely prunes any $\langle id, t \rangle$ pairs.

In contrast, the optimistic experiments show speedup as the number of solutions defining $convEnv$ increases when the query region is large (Figure 7.14) but not when the query region is small (Figure 7.12). We posit this to be due to the fact that large query regions are more likely to contain enough points assigned non-zero probability to cause the probability mass of the query region according to $convEnv$ to be contained in the query interval. However, when the query region is small, it is substantially less likely that the query region will be assigned non-zero probability by a random solution.

7.5 Comments on Selection Algorithms

This chapter addressed the selection query for the SPOT framework. This probabilistic query will be of clear use in many of the environments where the SPOT framework may be deployed. Consider the example of the fleet of fishing ships, which want to travel as little as possible to find as many fish as possible. The fleet would benefit from knowing when and where fish will be with probabilistic guarantees. Or consider a military scenario, where opponents deliberately make tracking difficult. A com-

mander on the field clearly wants to be able to extract which enemies will be in range and when from the available probabilistic spatial temporal data. Supposing the data is uncertain enough that a single probability distribution function cannot be provided, the SPOT framework is the only one capable to answering these sorts of queries.

The algorithms presented in this chapter both involve pre-computation. The first pre-computes a tree structure called a SPOT tree, which is akin to the R-tree. The SPOT tree contains nodes with data bounding the set of interpretations satisfying their children. This data is stored in *composite atoms*, a new form of a SPOT atom. There are fast operations for updating these composite atoms to account for newly added children. Experimental results on a prototype implementation show algorithms based on SPOT trees to produce substantial speedup.

The second set of algorithms implements inscribing and circumscribing regions *bounding the solutions to the linear program*. In most common spatial settings, bounding boxes are put to substantial use. With these algorithms, there is both a *bounded* (inscribed) as well as a bounding (circumscribing) data structure, which may be used to speed up optimistic and cautious queries. In our experiments, these inscribed and circumscribing regions produce good results, and in particular they do well in some situations where little pruning can be accomplished by a SPOT tree.

These data structures are an important first step towards a fully function SPOT framework. They are a proof-of-concept showing that query procedures leveraging SPOT 's spatial temporal probabilistic structure are possible and can provide substantial speedup.

Chapter 8

Conclusion

In this thesis we have seen many algorithms and techniques for dealing with spatial probabilistic temporal (SPOT) databases. SPOT databases store probabilistic information of the form "Object id was in region r at time point t with a probability in the interval $[\ell, u]$". Despite previous work done on representing probabilistic spatial data [16, 45, 46, 36], this is the first work to handle the possiblity of not being provided with full probability distributions. In many domains, full PDFs for object locations are not available: one can only bound the probability of correctness. For instance, there are military prediction algorithms where vehicles are reported in regions according to probability intervals [34, 28]. Such domains require an abstract probabilistic logic such as SPOT .

There are two different potential formal semantics interpreting SPOT atoms. The time point semantics, intorduced in Section 3.3 (page 33) concerns the probabilities of an object being at locations at each time point, yet does not account in the semantics for an objects' movement contratints. Instead, such constraints are expected to be present in the data. For times when the movement constraints cannot be adequetely represented in the data, and there is external information available on what points are

reachable from where, there is the world-based semantics (Section 3.4 on page 36). These constraints use probability distributions over worlds to ensure that interpretations abide by the object's potential movement.

All sets of semantics admit polynomial time consistency checking algorithms. While it is relatively straightforward to construct a polynomial time consistency checking algorithm for the time point semantics (Section 4.1.1 on page 43), I was also able to introduce several alternative techniques to improve performance of consistency checking under the time point semantics. These included introducing a binary space partition to determine salient regions (Section 4.1.2 on page 44) and applying divide-and-conquer type techniques to clusters of related SPOT atoms (Section 4.1.3 on page 51).

The consistency checking under the world-based semantics is also a polynomial time proceedure. This is surprising, as the naïve representation of the world-based semantics is exponential. By using path probability variables, one can bypass the exponential number of worlds with polynomially many *path probability* variables. The proof of the equivalence of a path probability linear program to the world based semantics is described in Section 4.2.2. In Section 4.2.3 further efficiencies are introduced to the path probability formulation.

For times when the database is inconsistent, one may use the database revision techniques in Chapter 5 (page 79). Revision, like database repair, corrects inconsistent databases. To assure the adequecy of a revision strategies, all of the revision strategies are compared against the AGM axioms, which specify conditions necessary for a proper revision of a belief state [1]. Apart from standard subset based revision strategies (Section 5.5 on page 82), one can make changes to individual elements of a SPOT atom. For instance, Section 5.6 has an example proving there is no method

that modifies only the spatial component of SPOT atoms and also satisfies the AGM axioms. There is, however, a polynomial time method that revises the probability bounds of SPOT atoms to create a consistent database. In Section 5.8 (page 93) describes the probability revision method and proves it to satisfy the AGM axioms.

There are also methods for doing selection queries in SPOT databases. Selection queries tell which objects are in a given region at which times with a probability in a given range. They might, for instance, tell a cell phone company which cell phones will be in range of a given tower at which times with a probability greater than 90%. Several methods for computing selection queries are detailed. First Section 7.3 introduces the SPOT Tree, a new spatial probabilistic indexing structure. Modeled on the R-tree, where each node contains a box that "bounds" the data in its children, a SPOT tree node contains *composite atoms*, which bound the sets of interpretations entailed by the node's children. Just as the R-tree's bounding boxes serve to prune search, these bounds can be used to quickly eliminate many potential answers to various selection queries. Another method for computing selection precomputes both circumscribing regions as "bounds" for the set of interpretations and inscribed regions which are "bounded" by the set of interpretations. The inscribed and circumscribing regions provide further pruning oppurtunities, which produced substantial speedup in our experiments on cautious queries.

This thesis introduced SPOT databases. It investigated two separate semantics, and found consistency checking in both to be polynomial time. For inconsistenct SPOT database, such as might be encountered when using many disparate data sources, the thesis introduced several revisions strategies to repair the databases. The strategies were compared to the AGM axioms for belief revision. Also, some selection operations were considered from several different perspectives. I provide an

R-tree based indexing structure that sped up selection operations, and a pruning algorithm based on inscribed and circumscribing regions. Together, these contributions represent a significant first step towards a workerable spatial probabilistic temporal database framework of use in wide varieties of spatial probabilistic temporal settings.

Bibliography

[1] C.E. Alchourrón, P. Gärdenfors, and D. Makinson, *On the logic of theory change: partial meet contraction and revision functions*, Journal of Symbolic Logic **50** (1985), 510–530.

[2] David Applegate, William Cook, Sanjeeb Dash, and Monika Mevenkamp, *Qsopt library*, http://www2.isye.gatech.edu/ wcook/qsopt/index.html.

[3] M. Arenas, L. Bertossi, and J. Chomicki, *Consistent query answers in inconsistent databases*, Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (1999), 68–79.

[4] M. Arenas, L. Bertossi, and Jan Chomicki, *Answer sets for consistent query answering in inconsistent databases*, Theory and Practice of Logic Programming **3** (2003), 393–424.

[5] C. Baral, S. Kraus, and J. Minker, *Combining multiple knowledge bases*, IEEE TKDE **3** (1991), no. 2, 208–220.

[6] D. Barbara, H. Garcia-Molina, and D. Porter, *The management of probabilistic data*, Knowledge and Data Engineering, IEEE Transactions on **4** (1992), 487–502.

[7] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*, SIGMOD Conference, 1990.

[8] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel, *The X-tree: An Index Structure for High-Dimensional Data*, VLBD Conference, 1996.

[9] A. Bolognesi and P. Ciancarini, *Searching over metapositions in kriegspiel*, Computer Games 2004, 2004.

[10] Jean Bolot, *How to own a cellular network in your spare time, and what data mining has to do with it*, Talk at The University of Maryland, October 2007.

[11] G. Boole, *The laws of thought*, Prometheus Books (2003), 1854.

[12] H. Cao, O. Wolfson, and G. Trajcevski, *Spatio-temporal data reduction with deterministic error bounds*, The VLDB Journal The International Journal on Very Large Data Bases **15** (2006), 211–228.

[13] Roger Cavallo and Michael Pittarelli, *The theory of probabilistic databases*, Morgan Kaufmann Publishers Inc., 1987, pp. 71–81.

[14] B. L. Clarke and P. D. F. File, *A calculus of individuals based on"connection"*., Notre Dame J. Formal Logic **22** (1981), 204–218.

[15] A G Cohn and S M Hazarika, *Qualitative spatial representation and reasoning: An overview*, Fundamenta Informaticae **46** (2001), no. 1-2, 1–29.

[16] Xiangyuan Dai, Man Lung Yiu, Nikos Mamoulis, Yufei Tao, and Michail Vaitis, *Probabilistic spatial queries on existentially uncertain data.*, SSTD (Clau-

dia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, eds.), Lecture Notes in Computer Science, vol. 3633, Springer, 2005, pp. 400–417.

[17] N. Dalvi and D. Suciu, *Efficient query evaluation on probabilistic databases*, The VLDB Journal The International Journal on Very Large Data Bases **16** (2007), 523–544.

[18] A. Dekhtyar, F. Ozcan, R. Ross, and V. S. Subrahmanian, *Probabilistic temporal databases, ii: Calculus and query processing*, (2001).

[19] Alex Dekhtyar, Robert Ross, and V. S. Subrahmanian, *Probabilistic temporal databases, i: algebra*, ACM Trans. Database Syst. **26** (2001), 41–95.

[20] Curtis E. Dyreson and Richard Thomas Snodgrass, *Supporting valid-time indeterminacy*, ACM Trans. Database Syst. **23** (1998), no. 1, 1–57.

[21] T. Eiter, T. Lukasiewicz, and M. Walter, *A data model and algebra for probabilistic complex values*, Annals of Mathematics and Artificial Intelligence **33** (2001), 205–252.

[22] W. Faber, G. Greco, and N. Leone, *Magic sets and their application to data integration*, Journal of Computer and System Sciences **73** (2007), 584–609.

[23] R. Fagin, J. Y. Halpern, N. Megiddo, IBMAR Center, and C. A. San Jose, *A logic for reasoning about probabilities*, Logic in Computer Science, 1988. LICS'88., Proceedings of the Third Annual Symposium on (1988), 410–421.

[24] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi, *On the semantics of updates in databases*, PODS, ACM, 1983, pp. 352–365.

[25] Bettina Fazzinga, Sergio Flesca, Filippo Furfaro, and Francesco Parisi, *Dart: A data acquisition and repairing tool*, EDBT Workshops (Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, eds.), Lecture Notes in Computer Science, vol. 4254, Springer, 2006, pp. 297–317.

[26] Kenneth E. Foote and Donald J. Huebner, *Error, accuracy, and precision*, The Geographer's Craft Project, Department of Geography, The University of Colorado at Boulder, 1995.

[27] T. Hailperin, *Boole's logic and probability: a critical exposititon from the standpoint of contemporary algebra, logic, and probability theory*, North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands, 1986.

[28] Tom Hammel, Timmothy J. Rogers, and Bernie Yetso, *Fusing live sensor data into situational multimedia views*, Multimedia Information Systems, 2003, pp. 145–156.

[29] G. B. M. Heuvelink, *Error propagation in environmental modelling with gis*, Taylor & Francis, 1998.

[30] Frederick S. Hillier and Gerald J. Lieberman, *Introduction to operations research, 4th ed.*, Holden-Day, Inc., San Francisco, CA, USA, 1986.

[31] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, *Probview: a flexible probabilistic database system*, ACM Transactions on Database Systems (TODS) **22** (1997), 419–469.

[32] L.G.Khachiyan, *A polynomial algorithm in linear programming*, Soviet Mathematics Doklady **20** (1979), 191–194.

[33] M. R. Malek, A. U. Frank, M. R. Delavar, T. Technology, T. Tehran, and A. Vienna, *A logic-based foundation for spatial relationships in mobile gis environment*, Proceeding of 2nd Intenational symposium on LBS & Telecartography, Austria, Vienna (2006).

[34] R. Mittu, F. Segaria, S. Guleyopuglu, K. S. Barber, T. Graser, R. Ross, and J. Walters, *Supporting the coalition agents experiment (coax) through the technology integration experiment (tie) process*, (2003).

[35] Raymond T. Ng and V. S. Subrahmanian, *Probabilistic logic programming*, Information and Computation **101** (1992), no. 2, 150–201.

[36] J. Ni, C. V. Ravishankar, and B. Bhanu, *Probabilistic spatial database operations*, Advances in Spatial and Temporal Databases: 8th International Symposium, SSTD 2003, Santorini Island, Greece, July 24-27, 2003: Proceedings (2003).

[37] N. J. Nilsson, *Probabilistic logic*, Readings in Uncertain Reasoning (G. Shafer and J. Pearl, eds.), Kaufmann, San Mateo, CA, 1990, pp. 680–688.

[38] Austin Parker, Guillaume Infantes, John Grant, and V. S. Subrahmanian, *Spot databases: Efficient consistency checking and optimistic selection in probabilistic spatial databases*, Transactions on Knowledge and Data Engineering (2008).

[39] Austin Parker, Guillaume Infantes, John Grant, and VS Subrahmanian, *An agm-based belief revision mechanism for probabilistic spatio-temporal logics*, AAAI, 2008.

[40] Austin Parker, V.S. Subrahmanian, and John Grant, *A logical formulation of probabilistic spatial databases*, IEEE Transactions on Knowledge and Data Engineering **19** (2007), no. 11, 1541–1556.

[41] Austin Parker, Fusun Yaman, Dana Nau, and V.S. Subrahmanian, *Probabilistic go theories*, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), January 2007, pp. 501–506.

[42] Judea Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[43] H. Samet, *The quadtree and related hierarchical data structures*, ACM Computing Surveys (CSUR) **16** (1984), 187–260.

[44] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos, *The R+-tree: A Dynamic Index for Multi-Dimensional Objects*, VLBD Conference, 1987.

[45] Yufei Tao, Reynold Cheng, Xiaokui Xiao, Wang Kay Ngai, Ben Kao, and Sunil Prabhakar, *Indexing multi-dimensional uncertain data with arbitrary probability density functions*, VLDB '05: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 922–933.

[46] Yufei Tao, Xiaokui Xiao, and Reynold Cheng, *Range search on multidimensional uncertain data*, ACM Trans. Database Syst. **32** (2007), no. 3, 15.

[47] J. Zhang and M. F. Goodchild, *Uncertainty in geographical information*, CRC Press, 2002.

[48] W. Zhao, A. Dekhtyar, and J. Goldsmith, *Databases for interval probabilities*, International Journal of Intelligent Systems **19** (2004), 789–815.