A Data Model for Trip Planning in Multimodal Transportation Systems*

Joel Booth, Prasad Sistla, Ouri Wolfson†, Isabel F. Cruz‡
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois, USA
jbooth3@uic.edu, sistla@cs.uic.edu, wolfson@cs.uic.edu, ifc@cs.uic.edu

ABSTRACT

This paper introduces the problem of modeling urban transportation systems in a database where certain aspects of the data are probabilistic in nature. The transportation network is composed of multiple modes (e.g., automobile, bus, train, pedestrian) that the user can alternate between. A trip – a path between an origin and destination subject to some constraints – is the central concept. How these trips and the network can be represented as both a graph and relational model, as well as the requirements for querying are the main contributions of this paper. A set of operators are defined to work over these transportation concepts and they are integrated within a SQL-like syntax to express queries over the uncertain transportation network. Additionally, the paper shows how this model can be integrated within other moving objects and spatio-temporal data models, and how these graph-based queries can be processed.

1. INTRODUCTION

Urban transportation systems are large, complicated, and often difficult to utilize effectively. In recent years we have seen a growing number of resources that provide online maps, directories, location based services, and route planners that attempt to bring the necessary information to users. However, these systems do not provide a comprehensive solution to transportation information systems. Part of the difficulty in developing fully integrated systems is the heterogeneous nature of the data and the lack of a coherent data model that can be used effectively. In this paper, we propose a method to integrate the key aspects of spatio-temporal, moving objects, and graph-based databases to facilitate trip planning in urban transportation networks.

*This research was supported in part by NSF Award DGE-0549489.
†Additional support by NSF IIS awards 0847680, 0513736, 0326284.
‡Additional support by NSF Awards ITR IIS-0326284, IIS-0513553, and IIS-0812258

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

To motivate the scope of the paper, we begin by understanding what types of queries we would like to ask in order to determine the requirements for the data model itself. For example:

- Find a route that will get me home by my designated time and with 90% certainty.
- Using public transportation, find a route that lets me stop at a grocery store for 30 minutes on my way home such that I arrive by 7:00pm.

Immediately we see that routes, or trips, are the primary focus. These trips have spatial and temporal constraints as well as being subject to uncertainty. This paper focuses on the constructs, and their semantics, needed to express such queries. The goal is to provide a powerful and easy to use architecture for working with trips. Additionally, we introduce a flexible approach to query processing. We do not focus on querying for locations, nearest objects, and other classes queries for which there has been extensive research.

The remainder of the paper is organized as follows: first we introduce the concept of a multimodal urban transportation and how it is represented as a graph. Next we discuss existing network database modeling languages. Then we defineour relational network model. The relational model introduces a set of high-level transportation relations, operators that act over them, and their semantics. Examples of how these operators and relations can be used follow. Finally, we discuss how the model may be re-integrated with spatio-temporal models, the problem of query processing, and additional related work before concluding.

2. URBAN TRANSPORTATION NETWORKS

The urban transportation network has both static and dynamic components. The physical structure of the network does not change – buildings, roads, lakes, and train tracks remain constant (ignoring long term construction). However, the position of people and vehicles, status of the roads, departure times of busses, and similar components change continuously in real time.

The transportation network itself is composed of numerous routes that correspond to some physical paths. The most obvious paths are roads that can carry automobiles, busses, and in some cases pedestrians. There are also railroad tracks that carry trains. These routes are all labeled with unique identifiers (e.g., Red Line train, #12 bus, Roosevelt Rd., Interstate 94). It is possible for the routes to overlap on the same physical path (e.g., the #12 bus runs on Roosevelt Rd.).

A key factor in urban transportation is the presence of multiple modes of transportation. The modes considered in this paper are listed in Table 1. It is important to note that pedestrian is a specific mode of transportation. We do not consider air, water, or long range intercity travel at this time. These modes could likely be modeled in the same manner, but are beyond the scope of the current research.

Mode	Medium	Availability
auto	road network	always
pedestrian	⊆road network walkways	always
bus	bus routes	scheduled
urban rail	rail network	scheduled
suburban rail	rail network	scheduled

Table 1: Modes of transportation

We introduce the concept of a trip, which is a path from an origin to a destination through the network. In addition to these two defining characteristics there are four types of constraints that can be placed on a trip and are listed in Table 2. Facilities are resources such as banks, ATMs, and grocery stores that users may wish to include on a trip. While it is possible to include a specific gas station, often it is sufficient to include any one – hence we treat them as homogeneous classes. Treating the facilities as homogeneous classes does not preclude the user from including a specific place; visiting any pharmacy may be an option, but one can only retrieve one's clothing from a specific dry cleaners.

Class	Types of Constraints
Modal	The modes that are allowed and disallowed
Physical	Restrictions on the physical path the trip follows. E.g., avoid a specific neighborhood, do not include bridges, stay within a 5 block radius.
Facility	Constraints on the facilities that must be present on the route as well as the order they occur in.
Temporal	Starting and ending time of the trip. Time spent at facilities.

Table 2: Types of constraints on trips

Route finding, in general, attempts to find a path between points such that it is optimal according to some criteria. Many systems will calculate the shortest path either by distance traveled or duration. Users may be interested in additional types of optimizations, such as the number of inter-modal (e.g., bus to train) transfers they must take. Potential optimization criteria include distance, duration, cost, number of transfers, and the amount of walking.

Trips are subject to some level of uncertainty due to the dynamic nature of transportation systems. We assume that we have knowledge of the current speeds on links in the network as well as the expected arrival and departure times for public transportation. Being a real-world system, this information is unreliable and likely partially accurate; therefore, we describe it probabilistically. This allows us to determine a quantitative measure of the uncertainty of the result generated by a query.

3. GRAPH MODEL

We begin by defining the graph model used for representing the transportation network for querying trips. For illustrative purposes, Figure 1 presents a small example graph that will be referred to throughout this section.

We define a *transportation network* to be a tuple $U = (M, F, L, G)$ where M is a set of *modes*, F is a set of *facilities*, L is a set of attributes, and G is a labeled, directed, multigraph. The set M=*{pedestrian, auto, bus, urban rail, suburban rail}* corresponds to the *modes* of transportation available in the network. The set of *facilities* F represents the classes of facilities (e.g., grocery store, fast food restaurant) available on the transportation network. The set L denotes the *attributes* (e.g., length, name, mean speed) for the edges in the network.

Mode	Attributes
pedestrian	name, geometry
auto	name, mean speed (μ), geometry, speed variance (σ)
urban rail, suburban rail	name, mean speed (μ), run id, speed variance (σ), departure time, geometry
bus	name, run id, path departure time

Table 3: Mode attributes

For each $m \in M$ we define a set $edge\_attributes_m \subseteq L$ that describe attributes of the edges in the graph of that *mode*. Different modes have different attributes as enumerated in Table 3. Each edge has values for the attributes specified by the *mode* of the edge. Similarly, we define a set of attributes $vertex\_attributes \subseteq L$ that describe the attributes of vertices. Vertices are not associated with a mode, therefore all vertices have the same attributes $vertex\_attributes = \{name, geometry, facilities\}$. These attributes will be discussed in greater detail in the remainder of this section.

We define the graph as $G = (V, E, \Psi)$ where V is the set of labeled vertices and E is a set of labeled edges. Each edge is defined as a 4-tuple (v_1, v_2, m, ϕ_m) where v_1, v_2 are the endpoints of the edge, m is the *mode* label of the edge, and ϕ_m is a function that maps the attributes defined for the mode to values. For each vertex v and vertex attribute x, Ψ specifies the value of attribute x for vertex v. First we introduce the properties of vertices, and then in the following paragraphs we define the edges on a per-mode basis. Note that this defines a single unified graph. Edges of multiple modes may be incident on the same vertices, and in fact this is how the transfer between modes is modeled. This is illustrated in Figure 1.

All vertices have only three attributes (*name*, *geometry*, and *facilities*) as defined by *vertex_attributes*. The *name* represents some real-world name of the vertex. For example, it may be the name of the intersection of streets, the name of the train station, etc. The *name* value may be *null* if there is no appropriate name available. The *geometry* of a vertex represents its real-world geometry (e.g., the x,y coordinate on a map[1]). Finally, *facilities* represents the set, $f \subseteq F$, of

[1]A simple x,y coordinate is unlikely to suffice for a true

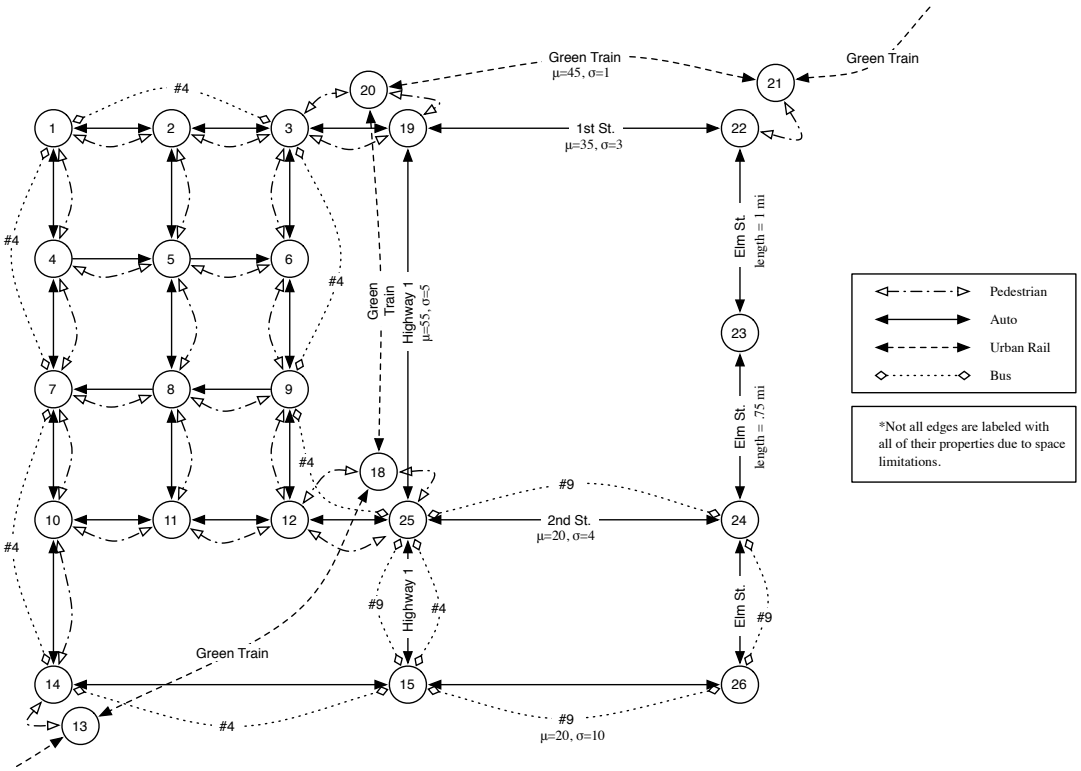

Figure 1: Example network

classes of facilities (e.g., grocery store, bank) present at the vertex. We assume that the facility is available at the given point in time if it is present in the set.

Now we introduce the edges in the network. We begin by describing the edges for the *pedestrian mode*. Intuitively these edges correspond to links, or segments, along which pedestrians may walk. They represent the segments between two intersections. In other words, the vertices of the pedestrian mode represent intersections. This includes many roads, additional pedestrian pathways (e.g., a path through a park) and importantly the segments of sidewalk between bus stops. The edges have a *name* that corresponds to the real world name of the segment (e.g., "Taylor St.", "Lakefront Walk"). Like for vertices we define a *geometry* that represents the real world segment (e.g., a piecewise linear function that represents a series of x,y coordinates where the endpoints correspond to the coordinates of the vertices). Finally, we assume there exist global parameters $\mu_{pedestrian}$ and $\sigma_{pedestrian}$ that represent the mean speed and its variance for the pedestrian mode.

For edges of *auto mode* the attributes are defined the same way as that for *pedestrian* edges. The edges themselves represent segments of the road system. These are generally the segments of roads between intersections or highway ramps. It also includes the segments between the last intersection and the endpoint of a road (e.g., dead-end, cul-de-sac). Vertices will exist where the intersections or endpoints occur. In other words, the coordinates of the endpoint vertices must correspond to the endpoints of the geometry. We assume that the speed associated with the edge is uncertain, but can be modeled with some standard, bounded[2] distribution with mean μ and variance σ. We call μ the expected speed, and if not specified otherwise, the speed associated with a link is assumed to be the expected speed.

For the *rail modes*, both urban and suburban, we have the same *name, length, geometry,* μ, and σ attributes as discussed previously. The *name* would correspond to the train route (e.g., "Blue Line", "South Suburban District"). The edges correspond to the segments of rail between consecutive train stations. The train stations act as vertices. We also define a *run id* attribute that denotes a specific run (e.g., the fifth southbound "Green Line" train for the day). For each pair of vertices there will be as many edges as there are runs. The *departure time* describes the expected time at which the link will be entered. We define a derived attribute *duration* that is equal to the expected amount of time spent on the edge (i.e., the length of the edge divided by the expected speed). We define the *arrival time* of an edge to be the *departure time* plus the *duration* of the edge. For any pair of edges e_1, e_2 with the same *name* and *run id*, and where the endpoint of e_1 is the start-point of e_2, the *departure time* of e_2 must be greater than or equal to the *arrival time* of e_1.

Finally, we discuss the edges of the *bus mode*. These edges have all of the attributes of the *rail modes*, except for the

representation of a vertex. The actual transportation system has objects (e.g., roads and bus stops) that have extended spatial regions. Because vertices are assumed to exist as connected points between modes, they must have a spatial extent. This is why we define a generic *geometry*.

[2]There are no infinitely long tails of minimal mass.

geometry, μ, and σ; instead it has an additional *path* attribute. The *path* maps the *bus* edge to the sequence of *auto* edges that represent the sequence of road segments between the two consecutive bus stops. This is to model the fact that busses run on the same physical roads as cars, and that the bus stops may not have a one-to-one correspondence with road intersections. For example, imagine that a bus stops on one road, makes a turn, and then stops on a different road. This is a single *bus* edge, but it corresponds to multiple *auto* edges. The existence of the *path* attribute obviates the need for a *geometry* attribute, since the *geometry* is taken from the auto (or road) network. An additional restriction is that the (coordinates of the) vertices at the endpoints of the edge must reside on the first and last line segment of the path *geometry*. The bus stops are represented as vertices. To illustrate this point, in Figure 1 the #4 bus has edges that map to more than one auto edge in the network. The *name* attribute denotes the name of the bus (e.g., "#8", "Loop"). Like for the *geometry*, the *path* attribute obviates the need for distinct speeds attributes as they can be derived from the corresponding *auto* edges.

Having provided technical definitions of edges and vertices, and before introducing concepts built on their aggregation, we reiterate a few important points. Vertices correspond to street intersections, bus stops, train stations, parking lots, the intersection of a footpath and a street, etc. Bus stops will exist either along an *auto* edge or at one of its endpoints. Train stations may exist along a *bus* or *auto* edge, or separated and only accessible via a connecting *pedestrian* edge. Any vertex that has incident edges of more than one *mode* signifies that a change between modes may occur at that vertex. There are no special transfer edges.

While we say that x,y coordinates or piece-wise linear functions can be used to define the geometries, in reality it may actually be more complex. For the purposes of this abstract model this is a reasonable approximation.

We define a *leg* to be a sequence of alternating vertices and edges starting and ending with a vertex where all of the edges have the same *name*, *mode*, and if available, *run id*. For each edge in the leg, its start vertex is same as the vertex preceding the edge in the leg; the end vertex of an edge is same as the vertex following the edge in the leg. We define the *departure time* of a leg to be the *departure time* of its first edge; similarly, we define its *arrival time* to be the *arrival time* of its last edge. A *trip* is a sequence of *legs*, where the beginning vertex of each succesive leg in the sequence is same as the end vertex of the preceding leg, such that the *departure time* of each subsequent *leg* is greater than or equal to the *arrival time* of the previous *leg*. We define the *departure time* and *arrival time* of a *trip* to be the *departure time* and *arrival time* of the first and last *leg* respectively.

We define a *transfer* to be a vertex shared by two different *legs* in the same trip. The *transfer* is *intermodal* if the *modes* of the incoming and outgoing *legs* are different, and *intramodal* if they are the same.

In addition to the *duration* of an edge, we can define a similarly derived attribute for any leg, trip, or even vertex. In the case of a vertex it represents the length of time spent at that vertex. This is of most interest when the vertex is a *transfer*.

Figure 2 illustrates a trip composed of four *legs*. It begins at vertex O and *transfers* from the *urban rail* to the *pedestrian mode* at vertex 13. At vertex 14 it transfers to the *bus mode* and travels to vertex 7. Once there, it *transfers modes* back to *pedestrian* and continues to the destination at vertex 9. Note that the *rail* and *bus* edges have been labeled with their specific *run id* in addition to their *name*. All of the transfers in this trip are intermodal. An example of an intramodal transfer would be moving from the #4 to the #9 bus at vertex 15 in Figure 1.

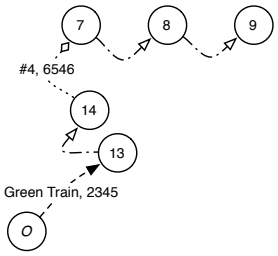

Figure 2: Example trip

4. EXISTING NETWORK LANGUAGES

Before defining the relational implementation of the graph model, we introduce a highly related work from which we draw a great deal of inspiration, as well as listing what we have determined to be the important limitations that our model addresses. Güting *et al.'s* Spatio-Temporal Query Language (STQL) provides an extremely rich data model and query language for modeling moving objects in both open areas [9, 10, 13] as well as road networks [14].

They define both datatypes and operators for spatial and temporal concepts. These types include *point*, *line*, and *region* – all of which have corresponding *moving* conceptualizations. More specific *graph point* and *graph line* types are provided for circumstances where the objects are constrained to a network (e.g., roads). This network itself has an explicit representation built from a set of *routes* (roads) and *junctions* (intersections). Note that this representation is not graph based, although a graph could be generated from the model.

Table 4 illustrates how these types can be used to describe the objects in a transportation system. Each of these objects would likely have additional attributes that can be described with traditional relational types (e.g., *int, real, string, boolean*).

Object	**Type**
person	*moving point*
house	*point*
park	*region*
rain storm	*moving region*
river	*line*
car	*moving graph point*
construction zone	*moving graph line*

Table 4: STQL types used for transportation

In order to exploit the powerful type system, a set of spatial and temporal operators are defined to operate over them. Spatial predicates such as **touches, overlaps, inside**, and **crosses** allow the user to determine how spatial objects are

related. Operators such as **duration** allow the user to query how long a moving object moves while **at** allows them to retrieve a value for a specific time. For a complete list of such operators we refer the readers to the primary literature [9]; only the general intuition of the operators is necessary to understand our work.

While STQL enables rich description of some transportation systems, it is still not sufficient for our applications. Most importantly, we find that:

- The STQL network model is restricted to only the road network. Multi-modality is a requirement in most urban planning scenarios, and for our purposes must be included in the model at the ground level.
- There are no explicit types for trip-related concepts. A shortest path is represented as a *moving point* or *line*. There are no explicit transfers, legs, or their concatenation. In order to properly model transportation systems all kinds of transportation concepts need to be modeled. For example, we must be able to express the path from point a to b involving the minimum number of transfers.
- The **trip** and **shortest_path** operations in STQL are limited to finding a shortest (distance) path between two points with no further restrictions. Ideally, the computation of paths should allow for intermediate stops as well as constraints based on mode choice, spatial, and temporal properties. The criteria chosen for optimization may mean that these are not all simple shortest paths problems.
- There is no explicit concept of a *facility* (e.g., grocery store, florist) in the STQL model. This must be present in order to allow the full range of trip-based queries we are interested in. Facilities must be compatible with the trip generation aspects of the language.
- Uncertainty is not addressed by the model. This is an important drawback because when dealing with transportation systems there is always some level of uncertainty inherent in the knowledge of the system state.

In Section 5 we show how we can use these types and predicates in conjunction with our work on querying trips, and in Section 8 we discuss how our trip model can be integrated back into the STQL type system.

5. THE RELATIONAL NETWORK MODEL

In this section we define the relations and operators needed to implement our graph model. The work focuses on querying trips subject to various constraints – including that of uncertainty.

5.1 Notation

In the remainder of the paper we use a specific notation for relations and operators. A relation is defined as:

```
name(attribute1:type, attribute2:type, ...)
```

where `name` is the name of the relation, `attribute1` and `attribute2` are attributes of the relation and `type` is a data type. These relations may not always be in first normal form. We refer to the value of an attribute for a relation or variable as `r.attribute` where `r` is the relation or variable, and `attribute` is an attribute of that relation. Relations and relation variables are written in `fixed width font`.

When not included in a relation definition or query example, all data types (e.g., *int, real, string, point, line*) are written in *italic font*. The entire text for relation definitions and example queries is written in a `fixed width font`.

We define an operator to be a function, denoted as follows:

operator(par1type, par2type, ...) → result

where the name is written in **bold face font**. If a parameter is optional it is listed inside square braces (e.g., [mode]). If a parameter can be of more than one type, the list of allowed types is written as: par1type|par2type|... All of the operators defined in this paper take a single tuple of the relation type defined by the parameter as input. If the output of the operator is a single tuple of a relation it will be written in lower case `fixed width font` while if it returns one or more tuple of a relation it will be written in upper case. If it returns a type value, it will be written in *italic font*.

5.2 A Relational Network

The transportation network graph can be described by relations representing vertices and edges. All of the vertices are modeled the same, but the edges are mode-dependent. We provide the relation for pedestrian edges here. The edges for the remaining modes can be defined similarly using the attributes listed in Section 3.

```
vertex(name:string, geometry:point,
  facilities:[set of string])

pedestrian_edge(name:string, geometry:line,
  origin:vertex, destination:vertex)
```

This representation has been used extensively for graph-based databases. Spatial types can be used to model the geometries while the temporal types can be used to model the departure times.

5.3 Querying for Trips

Our query structure builds on the standard "select, from, where" structure of SQL. We retain the same base syntax and structure but extend it in two important ways.

First, to query trips we introduce an operator

ALL_TRIPS(`vertex`, `vertex`) → `TRIP`

that accepts two vertices (the first being the origin, and the second the destination) as input and returns a relation of type `trip` that is defined as:

```
trip(id:int, origin:vertex, destination:vertex
  path:[sequence: l1:leg, ..., lk:leg])
```

In other words, this operator returns a nonmaterialized relation of all possible trips between the origin and destination vertices. It acts on the network graph as defined by the `vertex` and `edge` relations. We postpone the discussion of processing this operator until Section 9.

In addition to this operator and relation we introduce three new clauses that allow further specification of the parameters of the trip:

`WITH [ORDERED] STOP_VERTICES` A set of vertices (that may be ordered) to be included in the trip.

`WITH MODES` A list of the modes to be allowed in the trip.

`OPTIMIZE` A criteria by which the trip is optimized (e.g., distance, time, reliability), which is specified with the `MINIMIZE` or `MAXIMIZE` keyword.

`WITH CERTAINTY` A specified minimal probability that the trip can be executed as specified.

With these new clauses we have defined a generic query structure:

```
<SELECT *>
<FROM ALL_TRIPS(origin, destination)>
<WITH STOP_VERTICES>
<WITH MODES>
<WITH CERTAINTY>
<WHERE>
<OPTIMIZE>
```

The query structure allows for the full description of trips in an urban transportation system in a relational-like syntax. To better understand this structure, we present the following straightforward example:

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, bus
WITH CERTAINTY .8
WHERE FINISHES(t) <= 5:00pm
MINIMIZE LENGTH(t)
```

This query finds paths from a vertex `work` to a vertex `home` where only the pedestrian and bus modes are allowed. The `WHERE` clause specifies that the trip must finish by 5:00 pm and all of the constraints must be met with probability greater than or equal to .8. After these constraints have been met, the shortest path (by length) is selected.

The clauses are discussed in greater detail in Section 5.4 while the **finishes** operator, and others, are introduced in Section 5.5. Note that some of the not yet introduced operators will be used in the section on clauses. We provide detailed technical query semantics in Section 6.

5.4 Clauses

5.4.1 With Stop Vertices

The optional `WITH STOP_VERTICES` clause allows the specification of variables that range over vertices that are to be included in the trip. Inclusion of the `ORDERED` keyword indicates that the variables must be included in the order that they are listed. If the `ORDERED` flag is not present, the variables may be included in any order. Not specifying an order may dramatically increase the computational complexity of processing the query. If the clause is not included the trip will be calculated between the specified origin and destination only. An example of a query using the clause is as follows:

```
SELECT *
FROM ALL_TRIPS(home, work) AS t
WITH STOP_VERTICES v1
WITH MODES pedestrian, auto
WHERE  "movie theater" IN v1.facilities
MINIMIZE LENGTH(t)
```

In this query, a vertex possessing a movie theater in its set of facilities must be visited during the trip. We may specify a trip with two ordered stop vertices where both have a minimal duration as follows:

```
SELECT *
FROM ALL_TRIPS(home, the_movie_theater) AS t
WITH ORDERED STOP_VERTICES v1, v2
WITH MODES pedestrian, bus, urban rail
WHERE  "atm" IN v1.facilities
  AND DURATION(v1) > 5min
  AND "pizza place" IN v2.facilities
  AND DURATION(v2) = 60min
MINIMIZE DURATION(t)
```

5.4.2 With Modes

The `WITH MODES` clause simply specifies the list of allowed modes for the trip. Many users may only be interested in some subset of the modes available (e.g., only public transportation, only a personal automobile). If the clause is not used it is assumed that all modes in the network are allowed.

5.4.3 With Certainty

The `WITH CERTAINTY` clause specifies the minimal probability with which the where clause must be met. For example, in the following query we require that the total duration of the trip must be less than or equal to 35 minutes with a probability greater than or equal to .9:

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, urban_rail
WITH CERTAINTY .9
WHERE DURATION(t) <= 35min
MINIMIZE LENGTH(t)
```

Only the temporal aspects of the trip are subject to uncertainty. Spatial constraints are guaranteed to be satisfied (i.e., probability of 1.0).

5.4.4 Optimize

The `OPTIMIZE` clause allows the specification of the criteria by which the trip should be optimized (e.g., shortest length, least duration, fewest transfers). We allow the use of `MINIMIZE` and `MAXIMIZE` keywords.

The optimization is applied such that any conditions in the `WHERE` and `WITH CERTAINTY` clauses are met. That is, only trips that meet the criteria of the `where` clause with the required level of certainty are considered. Given the following partially defined query:

```
SELECT *
FROM ALL_TRIPS(origin, destination) AS t
...
OPTIMIZE
```

The `OPTIMIZE` clause would be `MINIMIZE DURATION(t)` to find the trip with the shortest duration. Generally, the optimization will always be a minimization, but there may be situations for which some maximization is appropriate.

When optimized by a temporal operator the *expected* value will be optimized. Only one optimality criteria may be chosen for each query. This restriction is made because allowing more than one optimization criteria on path computation is NP-hard [11].

5.5 Operators

In addition to **ALL_TRIPS**, we define four sets of operators that can be used in conjunction with the transportation relations: structural, temporal, spatial, and cost operators.

5.5.1 Structural

The first set of operators are used to manipulate the structural components of the relations. A `trip` is composed of components that can be accessed using the following operators:

intermodal_transfers(`trip`) → `TRANSFER`
intramodal_transfers(`trip`) → `TRANSFER`
all_transfers(`trip`) → `TRANSFER`
legs(`trip`) → `LEG`
edges(`trip`|`leg`) → `EDGE`
vertices(`trip`|`leg`|`edge`) → `VERTEX`

We define separate operators for intermodal (e.g., bus to rail), intramodal (bus to bus), and all transfers as the distinction may be important in certain circumstances. We define two transportation relations, `leg` and `transfer`, as follows:

```
leg(id:int, route:string, mode:string,
  origin:vertex, destination:vertex, run:int,
  path:[sequence: e1:edge, v1:vertex, ...,
  ek:edge, vk:vertex])

transfer(id:int, initial_mode:string,
  final_mode:string, location:vertex)
```

Each of these relations captures the information necessary to represent their corresponding concept in the graph model. `Trips`, `legs`, and `edges` can also be concatenated into a new trip using the **concatenate** operator, defined as:

concatenate(`trip`|`leg`|`edge`,
`trip`|`leg`|`edge`) → `trip`

Finally, we introduce a set of operators that access the count of various transportation sub-relations within transportation relations:

num_intermodal_transfers(`trip`) → *int*
num_intramodal_transfers(`trip`) → *int*
num_all_transfers(`trip`) → *int*
num_legs(`trip`) → *int*
num_edges(`trip`|`leg`) → *int*
num_vertices(`trip`|`leg`) → *int*

Specific operators for this task have been included because they correspond to common criteria by which a trip may be optimized. This avoids an unnecessary nested query that uses the `COUNT` operator and provides for less ambiguous semantics.

5.5.2 Temporal

The second set of operators are to access the temporal properties of the relations. The operators return the *expected* value of the uncertain variable.

arrival(`trip`) → *instant*
departure(`trip`) → *instant*
begins(`trip`) → *instant*
finishes(`trip`) → *instant*
duration(`trip`, [mode]) → *real*

Arrival and **departure** return the time at which a vertex, or transfer on a given trip is reached and left respectively. **Begins** and **finishes** return the time at which a component of the given trip beings and ends respectively. The **duration** operator returns the length of time spent on a component. The **duration** operator takes an optional mode parameter that restricts the calculation of the duration to only the portion spent on the given mode. For example, to calculate the duration of the trip spent on a bus.

5.5.3 Spatial

In addition to the temporal aspects of the transportation relations, we may also access their spatial attributes and corresponding geometries:

origin(`trip`|`leg`|`edge`) → *point*
destination(`trip`|`leg`|`edge`) → *point*
geometry(`trip`|`leg`|`edge`) → *line*
geometry(`transfer`|`vertex`) → *point*
length(`trip`|`leg`|`edge`, [mode]) → *real*

Similar to the **duration** operator, the **length** operator takes an optional mode operator that restricts the calculation of the length to only the portion spent on a given mode.

5.5.4 Cost

The final category of operators deals with the measures of fiscal or monetary cost of a trip. At this time we define only a single operator

cost(`trip`|`leg`, [context]) → *real*

that takes either a `trip` or `leg` as input and returns a *real* representing the cost (e.g., $3.50). We include an optional, undefined type for the *context* of the `trip` or `leg`. This may represent the time of day, any special discounts applied, use of a monthly pass, etc. that would influence the cost. This is important because cost, unlike length, is not a monotonic function of the edges in the path. We do not address this issue of context at this time, rather we simply introduce the syntactic component necessary to utilize cost information. Once again, we postpone the discussion of processing until Section 9.4.

6. SEMANTICS

The semantics of a query is defined as follows. First, from the non-materialized relation returned by the **ALL_TRIPS** operator, select the tuples that correspond to trips from *origin* to *destination* using only the specified modes to obtain a set $\mathcal{F}$. Let X be the set of variables specified in the `STOP_VERTICES` clause. For each such trip t, define a binding ρ for `STOP_VERTICES` to be a function that maps X to the set of stop vertices on the trip t. We say that such a binding ρ is *proper* if either the `STOP_VERTICES` clause is unordered, or it is ordered and the following condition holds: for every pair of distinct variables $x, y \in X$ such that x appears before y in the `STOP_VERTICES` list, it is the case that the vertex $\rho(x)$ is identical to, or appears before, the vertex $\rho(y)$ in t. From the set $\mathcal{F}$, we select those trips t such that there exists

a proper binding ρ and the `WHERE` condition of the query is satisfied by the pair (t, ρ) with probability greater than or equal to the value specified by the `WITH UNCERTAINTY` clause. From the resulting trips, we further select those for which the value specified by the `OPTIMIZE` clause is optimal where expected values are used for the temporal operators.

Now, we define the probability of satisfaction of the where condition with respect to a pair (t, ρ) where t is a trip in $\mathcal{F}$ and ρ is a proper binding for the variables in X with respect to t. Let ϕ be the `WHERE` condition of query. Now we formally define the probability of satisfaction of ϕ with respect to the pair (t, ρ). First observe that ϕ is a boolean combination of atomic conditions. We say that an atomic condition is *temporal* if it involves a temporal operator specified in Subsection 5.5.2. The satisfaction of a non-temporal condition by the pair (t, ρ) has no uncertainty in it. Thus, for each non-temporal atomic condition, we determine whether it is satisfied or not by the pair (t, ρ). We replace every occurrence of such an atomic condition in ϕ, respectively, by *TRUE* or by *FALSE* depending on whether the atomic condition is satisfied by (t, ρ) or not. After such replacement, we simplify the resulting formula. Let ϕ' be the resulting formula. If ϕ' is equivalent to *TRUE* then the probability of satisfaction of ϕ with respect to (t, ρ) is defined to be 1. If ϕ' is equivalent to *FALSE* then the probability is defined to be 0. If neither of the above conditions is satisfied, we define the probability of satisfaction of ϕ with respect to (t, ρ) as follows.

First observe that every atomic condition in ϕ' is temporal. The trip t is given by a sequence of legs. The binding ρ maps some of the variables in X to transfer points in t and some to intermediate vertices on some leg. Note that the transfer point is the end point of a leg and the start point of the subsequent leg. If a variable in X is mapped to an intermediate point on a leg in t then we split that leg at that point into two legs. By doing this, we get a trip t' so that all points in X are mapped to transfer points in t' by ρ. We define the probability of satisfaction of ϕ by the pair (t, ρ) to be the probability of satisfaction of ϕ' by (t', ρ) that we define as follows.

Let the trip t' be the sequence of legs of the trip: $(L_1, L_2, ..., L_n)$. For each $i = 1, ..., n$, let u_i be the end point of the leg L_i. Note that $u_1, ..., u_{n-1}$ are the transfer points and u_n is the terminal point. For each i, $1 \leq i \leq n$, let Y_i, Z_i be the departure time and duration time of leg L_i respectively. Observe that all these variables are random variables. Let $\vec{Y}$ and $\vec{Z}$ denote the vectors of random variables $(Y_1, ..., Y_n)$ and $(Z_1, ..., Z_n)$ respectively. Now let $f_{\vec{Y},\vec{Z}}(y_1, ..., y_n, z_1, ..., z_n)$ represent the joint density function of the random variables in $\vec{Y}, \vec{Z}$ where y_i, z_i are variables denoting the values of the random variables Y_i, Z_i respectively. We also let $\vec{y}, \vec{z}$ represent the sequences of variables $y_1, ..., y_n$ and $z_1, ..., z_n$, respectively. Let g be the product $f_{\vec{Y},\vec{Z}}(\vec{y}, \vec{z}) \cdot dy_1 \cdot dy_2 \cdot ... \cdot dy_n \cdot dz_1 \cdot ... \cdot dz_n$. Note that g denotes the probability that Y_i lies in the interval $[y_i, y_i + dy_i]$ and Z_i lies in the interval $[z_i, z_i + dz_i]$ for $i = 1, ..., n$.

Observe that the arrival time of the trip at transfer point u_i and duration at u_i are random variables given by the expressions $Y_i + Z_i$ and $Y_{i+1} - Y_i - Z_i$ respectively. Now we transform the condition ϕ' into an equivalent formula by replacing temporal operators by linear combinations of variables in $\vec{y}$ and in $\vec{z}$ as follows. If variable $x \in X$ is mapped to the transfer point u_i then we replace the term $arrival(t, x)$ by $y_i + z_i$, the term $departure(t, x)$ by y_{i+1}, and the term $duration(t, x)$ by $y_{i+1} - y_i - z_i$. We also replace $begins(t)$ by y_1 and $finishes(t)$ by $y_n + z_n$. Similarly other terms in ϕ' are replaced by the variables in $\vec{y}$ and $\vec{z}$. Let ϕ'' be the resulting formula. Intuitively, ϕ'' specifies the possible combination of values of the random variables in $\vec{Y}$ and $\vec{Z}$ that satisfy the conditon ϕ'. To the formula ϕ'' we need to add additional conditions requiring that each transfer in the trip is feasible. Consider the transfer point u_i. In order for the transfer at u_i to be successful there should be a sufficent gap between the arrival time of leg L_i and the departure time of L_{i+1}. We assume that this required gap is given by a positive constant d_i which is associated with the transfer point u_i. Now, the condition for successful transfer at u_i is given by the atomic condition $y_{i+1} - y_i \geq d_i$. Now, for each $i = 1, ..., n-1$, we add the above condition as a conjunct to ϕ''. Let ψ be the resulting formula. Now consider the $2n$-dimensional space $\mathbb{R}^{2n}$ represented by the $2n$ variables $\vec{y}, \vec{z}$ where $\mathbb{R}$ is the set of real numbers. Let S be the region of points in $\mathbb{R}^{2n}$ that satisfy the condition ψ. We define the probability of satisfaction of ϕ by the pair (t, ρ) to be the value of the definite integral of g over the region S.

Example: Let t' be a trip having two legs L_1, L_2. Let ϕ' be the formula $begins(trip) \geq 8 \wedge finishes(trip) \leq 10$ which requires the trip to start after 8 and finish before 10; all units of time are in hours. Now we have four random variables Y_1, Y_2, Z_1, Z_2 and a joint density function $f_{\vec{Y},\vec{Z}}(y_1, y_2, z_1, z_2)$. Using the above construction, we get ϕ'' to be $y_1 \geq 8 \wedge y_2 + z_2 \leq 10$. Let d_1 be 0.1. The transfer condition at the single transfer point is given by $y_2 - y_1 - z_1 \geq 0.1$. The formula ψ is the conjunction of ϕ'' and the above transfer condition.

7. EXAMPLE QUERIES

In addition to our graph relations, for these examples we will define an additional relation using a spatial type:

```
neighborhood(area:region, name:string,
  population:int)
```

We also use the **distance** and **intersects** operators from spatio-temporal modeling. We assume there are two vertices, `home` and `work`, in the following examples. While any arbitrary vertices can be used, these are chosen for their clear meaning.

1. Find a trip home from work, using public transportation, that minimizes the number of intermodal transfers made.

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, bus, urban_rail
MINIMIZE NUM_INTERMODAL_TRANSFERS(t)
```

2. Find the fastest way to work from home that passes a pharmacy within 2 Km of the office, leaving at 8:00. The trip may use a personal automobile in addition to walking.

```
SELECT *
FROM ALL_TRIPS(home, work) AS t
WITH STOP_VERTICES v1
WITH MODES pedestrian, auto
WHERE BEGINS(t) = 8:00 AM
```

```
  AND "pharmacy" IN v1.facilities
  AND DISTANCE(GEOMETRY(v1), GEOMETRY(work)) < 2 Km
MINIMIZE DURATION(t)
```

3. With a certainty greater than or equal to .75, find the least expensive trip home from work that uses public transportation and visits a pharmacy and then a florist (spending at least 10 minutes at each)

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH ORDERED STOP_VERTICES v1, v2
WITH MODES pedestrian, bus, urban_rail
WITH CERTAINTY .75
WHERE "pharmacy" IN v1.facilities
  AND "florist" IN v2.facilities
  AND DURATION(v1) > 10min
  AND DURATION(v2) > 10min
MINIMIZE COST(t)
```

4. Find a trip home from work using public transportation that minimizes my walking and does not go through Lincoln Park, stops at a pizza place and spends 45 minutes there.

```
SELECT *
FROM ALL_TRIPS(work, home) AS t, neighborhood AS e
WITH STOP_VERTICES v1
WITH MODES pedestrian, bus, urban_rail
WHERE NOT INTERSECTS(GEOMETRY(t), e.area)
  AND e.name = "Lincoln Park"
  AND "pizza" IN v1.facilities
  AND DURATION(v1)  = 45 min
MINIMIZE LENGTH(t, pedestrian)
```

5. Find the fastest automobile-based trip home from work that stops at a preselected vertex with the name "UIC".

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH STOP_VERTICES v1
WITH MODES auto
WHERE v1.name = "UIC"
MINIMIZE DURATION(t)
```

6. Find a trip home from work that arrives by 7:00 PM with certainty greater than or equal to .8, and spends the least time possible on busses.

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, bus, urban_rail
WITH CERTAINTY .8
WHERE FINISHES(t) <= 7:00 PM
MINIMIZE DURATION(t, bus)
```

7. Find a trip home from work that leaves by 5:00 PM and arrives by 7:00 PM with certainty greater than or equal to .8. It must spend the maximum amount of time possible at a pizza place along the way.

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH STOP_VERTICES v1
WITH MODES suburban rail, pedestrian
WITH CERTAINTY .8
WHERE BEGINS(t) >= 6:00 PM
  AND FINISHES(t) <= 8:00 PM
  AND "pizza" IN v1.facilities
MAXIMIZE DURATION(v1)
```

8. TRIPS AND OTHER MODELS

Our proposed model should not be considered entirely in a vacuum. We have presented a novel approach to modeling and generating multimodal trips for a transportation network, but it would also be interesting to see how these concepts could be exported to other models. Given our previous discussion of STQL (Section 4) the first step would be to see how it and our model could be related.

Despite different underling models, both data models are based on a network representation. If STQL were extended with even rudimentary support for multiple modes (e.g., labels on edges), we could export our trip relations to appropriate STQL data types. For example,

2trajectory(`trip|leg|edge`) → *graph line*
2mgpoint(`trip|leg|edge`) → *moving graph point*
2gpoint(`vertex|transfer`) → *graph point*

would allow for at least a minimal translation of our concepts.

Implementing probabilistic link speeds, explicit timetables, runs, stops, transit routes, and other concepts would be more difficult and are far beyond the scope of this paper. At the same time, it would be interesting to explore additional syntactic constructs needed given the probabilistic aspect. Perhaps some integration with other moving objects systems (see Section 10.3) would yield interesting results.

9. QUERY PROCESSING

Given the expressiveness of our model, the query processing becomes an important and non-trivial consideration. Rather than develop a single, specific algorithm for query processing we view the solution as a *framework*.

We first discuss the processing of the **ALL_TRIPS** operator and returned relation. Depending on the specific combination of the number, and type, of visits, the optimization criteria, the modes allowed, etc., certain algorithms may be more efficient than others in calculating a path. Secondly, we discuss how it is possible to process the probabilistic information in order to implement the uncertainty operators. Finally, we briefly discuss a number of general processing steps that can be applied to most queries and the problem of determining the cost of a trip.

9.1 Trip Algorithms

A wide range of path algorithms supporting route queries have been developed in computer science, transportation science, and operations research. Some of these account transportation-specific constraints such modal transfers, schedules, and cost computation. All of these algorithms utilize a graph model. The cases each of these algorithms handle can be expressed in a canonical form in our query language. Once such a canonical form is recognized, we will invoke the appropriate algorithm. This produces a framework for incorporating algorithms that will be developed in the future.

We propose identifying the canonical forms for specific algorithms by the constraints specified in different clauses. Remember that our generic query structure is:

```
<SELECT *>
<FROM ALL_TRIPS(origin, destination)>
<WITH STOP_VERTICES>
<WITH MODES>
<WITH CERTAINTY>
<WHERE>
<OPTIMIZE>
```

In the following subsections we show how this structure can be used by enumerating three classes of queries and a corresponding algorithm that can be used to process them. These three classes of queries are not sufficient to cover all potentially expressible queries at this time; rather, they are used to demonstrate how the framework operates. The algorithms listed are not necessarily the only algorithm that work for the specified class.

9.1.1 Simple Shortest Path

With the following constraints, we can define a class of queries that are expressible as a simple shortest path problem:

> `WITH STOP_VERTICES` must be empty
> `WITH MODES` may have any values
> `OPTIMIZE` is the minimization of the sum of some numeric edge attribute (e.g., length, duration)

An example of a query meeting these constraints would be

```
SELECT *
FROM ALL_TRIPS(home, work) AS t
WITH MODES pedestrian, bus, urban_rail
MINIMIZE DURATION(t)
```

The constraints are such that the graph is limited to the edges of certain modes (all edges for disallowed modes can be ignored in computing the path) and the edges have a numeric weight value. There is a single origin and destination vertex. A query in this form can be calculated in $O(e+v^2)$ time (or better depending on the data structures used to store the graph) using a version of Dijkstra's algorithm that has been modified to operate on a multigraph.

9.1.2 One or More Ordered Stop Vertex

As a simple extension to the simple shortest paths form, we consider the case where there is one or more ordered stop vertex in the trip. We can express these constraints as:

> `WITH ORDERED STOP_VERTICES` one or more stop vertices for which a facility has been specified
> `WITH MODES` may have any values
> `OPTIMIZE` is the minimization of the sum of some numeric edge attribute (e.g., length, duration)

An example of a query meeting these criteria would be

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH ORDERED STOP_VERTICES v1, v2
WITH MODES auto
WHERE "wine_store" IN v1.facilities
  AND "hotel" IN v2.facilities
MINIMIZE DURATION(t)
```

We can express the problem as a sequence of shortest paths. A sketch of the algorithm is as follows:

input: origin, destination, ordered list of stop vertices (with facilities), and network graph G
output: a shortest path

1. Generate a new graph, G', with vertices corresponding to the origin and destination.
2. In G, calculate the shortest path from the origin to all vertices with the facility specified by the first stop vertex. For each shortest path found add a vertex and edge to G' where the edge weight is the length of the shortest path.
3. If there are more stop vertices remaining go to step 4. Else, compute the shortest path from the last set of stop vertices to the destination, adding the corresponding vertex and edge to G'. After this go to step 5.
4. Consider the next stop vertex specification. Compute the shortest path from all of the last set of stop vertices to all of the vertices that have the facility specified by the next stop vertex. Add a corresponding vertex and edge to G'.
5. In the graph G' compute the shortest path from the origin to the destination. Reconstruct the corresponding, complete shortest path in G and terminate.

Each invocation of the single source shortest path algorithm has complexity $O(e + v^2)$. There are $O(mv)$ such invocations where m is the number of stop vertices. Thus, thus the overall complexity is $O(mv(e + v^2))$.

9.1.3 Transfer Constrained Multimodal Paths

As an example of this frame work, consider the algorithm of Lozano and Storchi [15] for computing shortest paths in multimodal transportation networks. A key point of the algorithm is that it computes the shortest trip based on some cumulative edge weight subject to a maximum number of intermodal transfers. The class or queries handled by this algorithm can be described canonically as:

> `WITH STOP_VERTICES` must be empty
> `WITH MODES` may have any values
> `WHERE` includes a constraint on the number of transfers
> `OPTIMIZE` is the minimization of the sum of some numeric edge attribute (e.g., length, duration)

and an example of such a query is:

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, bus, urban_rail
MINIMIZE NUM_ALL_TRANSFERS(t)
```

9.2 Uncertainty

Another major query processing consideration is introduced by the uncertainty operators. In this section we present a simplification to the processing mentioned in Section 6. We assume that the duration time of travel on the links in the network are modeled as a bounded (i.e., no infinitely long tails of minimal density) probability distribution function that is defined by some mean and variance.

In Section 6, we gave a definition of the probability of satisfaction of the where condition with respect to a trip t' and an evaluation ρ. This definition uses a joint probability density function f over random variables representing departure times on each leg of the trip and the duration times of each leg.

Now assume that these random variables are independent; this is often reasonable because the travel times of separate trains or busses are independent. As a consequence, the joint density function can be written as the product of the density functions of each of the random variables. Thus, the assumption that we know a joint probability distribution on a larger set of variables can be eliminated.

9.3 General Processing

One can easily disallow a mode, or edges/vertices whose geometry intersects a region. To do so, the edges are simply ignored by the path algorithms.

Similarly, the opening/closing times of facilities can be taken into consideration. If we have a relation that relates facilities and vertices, and it also maintains the periods of time during which the facility is open, then we can ignore the closed facilities at the appropriate times during path computations.

9.4 Cost Processing

As mentioned in Section 5.5.4, calculating the cost of a trip is non-trivial. Unlike measures such as length, the cost of a trip is not a strictly monotonic increasing function of the edges. Attributes of the trip, transportation system, and even the user will affect how the cost is calculated.

In the case of optimization by cost alone, the calculation will be trivial (i.e., the entire trip will be pedestrian). In more interesting cases, such as when there is a constraint on both trip cost and duration, the processing will probably be NP-complete. In general, one must consider individually simultaneous optimization of cost and another criterion. This discussion is beyond the scope of this paper.

10. RELATED WORK

Before concluding it is important to discuss a number of related works. Each addresses some subset of the problems we cover: route planning, database models, and uncertainty.

10.1 Transportation Information Systems

The transportation informations available to users today generally fall into two categories: form-based and map-based. Many transportation agencies [2, 18, 21] provide web sites that allow users to plan a trip using the public transportation system. They tend to allow the specification of time constraints, mode constraints (some include information for the auto network as well), preferences for walking distance, and how the trip should be optimized (e.g., duration vs. number of transfers). If a valid trip can be constructed the user is presented with an itinerary for its execution.

A wide range of algorithms [1, 3, 15] supporting these route queries have been developed to account for the problems with modal transfers, schedules, and cost computation. The algorithms utilize a graph model extend well known shortest path algorithms (e.g., Dijkstra's), and have polynomial running times. Pending further investigation, the fields of Transportation Science and Operations Research will likely yield more algorithms that fit within our query processing framework.

The second common class of planning tools have map-based graphical user interfaces [12, 16]. Users may enter their origin and destination via either a form or by clicking points on a map. These generally do not allow for constraints on time, or the use of public transportation. Unlike most form-based planners, some map-based sites allow for the insertion of multiple stops along the trip and may include some real-time traffic information.

Being end user systems, these planners do not provide the full functionality we are looking for. The underlying database is not open to custom queries, one cannot generate trips subject to a wide number of constraints, and there is no concept of uncertainty. Including facilities in the query is cumbersome if possible at all.

10.2 Graph Query Languages

Querying of graph databases has been studied extensively. Dar and Agrawal [7], and Cruz and Norvell [6] have presented algorithms and analysis of the computation of generalized transitive closure, and aggregative closure respectively. These algorithms build on the basic concept of transitive closure to allow queries for maximum capacity path, critical path, most reliable path, shortest path, bill of materials, and "connecting flights" among others.

Dar, Agrawal, and Jagadish [8] go on to describe algorithms for optimizing these queries. This is especially important given that for some queries there is an exponential or infinite number of possible paths. This clearly violates the need for polynomial time algorithms. However, through optimization of the query plan it is possible to prune the search space dramatically – making efficient computation possible.

These approaches do not model uncertainty in any way. While it may be possible to specify multimodal graphs and facilities in the models, they are not included as explicit constructs available for direct manipulation. We introduced a graph model that is both highly expressive as well as being tailored to an important domain.

10.3 Moving Objects Databases

In the last fifteen years there has been an ever growing interest in, and demand for, databases for moving objects. Pelekis *et al.* [17] have compiled an extensive review of many of the most important models. We have already explained the works of Güting *et al.*, possibly the most comprehensive model for spatial and temporal types in detail earlier in the paper. There is one additional model that we should introduce as it deals extensively with uncertain queries: the Moving Objects Spatio-Temporal (MOST) model [19, 20].

MOST is a data model based on the authors' Future Temporal Logic and is designed to model the current and near-future positions of moving objects. While *point, line,* and *region* types are supported in the spatial model, only point types may be moving. Points are represented as vectors that capture their current position and velocity. Queries may be issued over the current moment or times in the near future. Future queries estimate the position of points based on their last known position and velocity.

Predicates like `SOMETIME_MAYBE` and `DEFINITELY_ALWAYS` have been introduced to handle the uncertainty from a linguistic perspective. For example, these predicates could be applied to a query determining whether or not a point passes through a specific region during a time period. In the underlying model, the exact position of an object is known at the moment the record is updated. As time passes, the level of uncertainty grows until an update is received. There is a tradeoff between the accuracy of the database and the frequency of updates.

Because we are not working with traditional moving objects in our work, we simplified the problem of uncertainty down to a single clause in the query. If we were to extend the work to integrate better with moving objects databases (as discussed in Section 8) it would be important to consider new uncertainty predicates.

11. CONCLUSIONS AND FUTURE WORK

We introduced the concept of multimodal transportation networks, their properties, and how they can be modeled as a graph. We developed a relational model based on that graph that supports operations on transportation-level concepts. We introduced syntax to allow powerful, expressive queries on the trips; and methods for working with uncertainty. We considered the problem of query processing as a framework rather than a single rigid algorithm. Depending on the structure of the query we can select an appropriate algorithm to process the query, which in turn allows for more flexible extensions to the system. We explored existing spatio-temporal models and showed that our work can leverage existing type and operator framework. We also demonstrated that our trip-based framework can be exported back into such a type system.

In the future we would like to explore the potential implementation of the model in an actual RDBMS. One consideration is the development of a graphical user interface for querying the multimodal trips. Graphical query languages have been studied before [4, 5] and may provide a foundation for defining the relationship between the graphical and relational representations of the query. GUI-based mapping programs and GIS systems may provide further inspiration in how the interface should look. One potential approach is that the trip is presented as a timeline with the origin and destination as endpoints. Users can drop stops on the timeline in the order in which they want them filled. Properties for each stop and leg can be specified by the use of contextual menus and highlighting regions on a map. A small number of global properties can be specified through a persistent box with options. Providing a graphical query language that has a direct correspondence to the underlying syntactic query representation would allow novice and expert users alike to develop applications using the model.

We will continue to explore additional classes of queries and appropriate algorithms for them in our processing framework. In the current work the queries only cover a query made before a trip is executed. Another approach will be to explore how queries could self-update as the trip itself is executed, thus allowing for unforeseen changes.

12. REFERENCES

[1] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A graphical query language with recursion. *Software Engineering*, 16(10):1150–1163, 1990.

[2] Bay Area Rapit Transit Planner. http://www.bart.gov/.

[3] M. Bielli, A. Boulmakoul, and H. Mouncif. Object modeling and path computation for multimodal travel systems. In *European Journal of Operational Research*, volume 175, pages 1705–1730, December 2006.

[4] M. P. Consens, I. F. Cruz, and A. O. Mendelzon. Visualizing queries and querying visualizations. *SIGMOD Rec.*, 21(1):39–46, 1992.

[5] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A Graphical Query Language Supporting Recursion. In *ACM SIGMOD International Conference on Management of Data*, pages 323–330, 1987.

[6] I. F. Cruz and T. S. Norvell. Aggregative closure: An extension of transitive closure. In *International Conference on Data Engineering (ICDE)*, pages 384–391, Washington, DC, USA, 1989.

[7] S. Dar and R. Agrawal. Extending sql with generalized transitive closure. *IEEE Trans. on Knowl. and Data Eng.*, 5(5):799–812, 1993.

[8] S. Dar, R. Agrawal, and H. V. Jagadish. Optimization of generalized transitive closure queries. In *International Conference on Data Engineering (ICDE)*, pages 345–354, Washington, DC, USA, 1991.

[9] M. Erwig and M. Schneider. Spatio-temporal predicates. *IEEE Trans. on Knowl. and Data Eng.*, 14(4):881–901, 2002.

[10] M. Erwig and M. Schneider. STQL: A spatio-temporal query language. *Mining Spatio-Temporal Information Systems*, 2002.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, page 214. W. H. Freeman, 1979.

[12] Google Maps. http://maps.google.com/.

[13] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.

[14] R. H. Güting, T. de Almeida, and Z. Ding. Modeling and querying moving objects in networks. *VLDB Journal*, 15(2):165–190, 2006.

[15] A. Lozano and G. Storchi. Shortest viable path algorithm in multimodal networks. In *Transportation Research Part A: Policy and Practice*, volume 35, pages 225–241, March 2001.

[16] Mapquest. http://www.mapquest.com/.

[17] N. Pelekis, B. Theodoulidis, I. Kopanakis, and Y. Theodoridis. Literature review of spatio-temporal database models. *Knowl. Eng. Rev.*, 19(3):235–274, 2004.

[18] Regional Transit Authority Trip Planner. http://tripsweb.rtachicago.com/.

[19] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *International Conference on Data Engineering (ICDE)*, pages 422–432, 1997.

[20] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 310–320. Springer, 1998.

[21] Washington Metropolitan Area Transit Authority Trip Planner. http://www.wmata.com/tripplanner_d/tripplanner.cfm.