Modeling and Querying Multimodal Urban Transportation Networks

BY

JOEL ALEXANDER BOOTH
B.A., Kalamazoo College, 2005

DISSERTATION

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2011

Chicago, Illinois

ACKNOWLEDGMENTS

I would like to thank all of my friends, family, coworkers, advisors, committee members, office staff, and everyone else who helped me through this. Without the guidance and support you provided, this dissertation would not exist. I would also like to acknowledge the support provided by NSF DGE-549489, NSF ITR-0326284, NSF III-COR 0812258, NSF IIS IIS-0133123, NSF ALT-0536968, NSF IIS 0905593, and University of Illinois at Chicago's University Fellowship.

JAB

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

DB	Database
DBMS	Database Management System
ITA	Intelligent Traveler's Assistant
ITS	Intelligent Transportation System
NL	Natural Language
NLDB	Natural Language Interfaces to Databases
NLI	Natural Language Interface
NLP	Natural Language Processing
SW	Semantic Web
TRANQUYL	The Transportation Query Language

SUMMARY

This dissertation presents a comprehensive and coherent approach to modeling and querying multimodal urban transportation networks. The time-dependent graph model captures the spatio-temporal aspects of real world transportation systems needed in order to effectively plan trips through the network. The temporal aspects of the model (e.g., transit schedules, current speed of roads) are modeled probabilistically. From the graph model, a relational model is developed in order to facilitate the generation of a new query language—the TRANsportation QUerY Language (TRANQUYL). This language allows for succinct and intuitive expression of complex trip planning queries.

In order to facilitate use of TRANQUYL by lay-users, a natural language interface was developed to translate English language queries into TRANQUYL. The approach was shown to be robust with regards to grammatical and vocabulary variation, and capable of translating long, complex queries. The system incorporates self referential language, allowing the user to make references to previously known places of interest as well as generic resources. Much of this functionality is accomplished through the use of ontologies. The efficacy of the system was shown through an intrinsic study.

CHAPTER 1

INTRODUCTION

Imagine arriving in a new city where you know very little about the transportation system. You know where you are staying and maybe one or two other points of interest, however you have not had a chance to learn where other resources such as a bank are located, nor have you learned the intricacies of the public transportation system. Whether you are on vacation or have recently moved, you will want to go out into the city. You may wish to go out and explore by wandering without direction, or wait and study the necessary maps and timetables first, but regardless of your strategy, you will need access to a great deal of information to make informed decisions. This dissertation represents an approach that can provide transportation information to people—more specifically, information on how to travel through a transportation network efficiently.

This research is important and timely as our ability to address transportation problems (e.g., congestion) are rapidly approaching physical and financial limitations. We are no longer able to "build our way out" of the problem; space for more roads is scarce and too expensive. Luckily, technology has progressed to a point where we can use it to overcome some of these limitations. Intelligent Transportation Systems (ITS) initiatives, such as those supported by the Intellidrive initiative of the US Department of Transportation [United State Research and Innovative Technology Administration, 2011] are working to apply these technical solutions to transportation problems.

One way in which we can alleviate congestion is by making more efficient use of the existing system, and in order to do that the users need to be able to make informed choices. The rise of personal GPS navigation systems, smart phones, and online route planners (e.g., Google Maps) demonstrate the need for such information. Unfortunately, these systems are still lacking in many ways. They generally only support finding a route to a single destination with few if any constraints. They ignore the very real problem of how uncertain the information is in a transportation network: transit schedules are often unreliable and speeds change on roads due to reoccurring congestion.

Going beyond these simple systems, the motivation for this dissertation comes from the larger research context of developing an Intelligent Traveler's Assistant (ITA) [Dillenburg et al., 2002]. This would be a single, handheld device (or software system running on a smartphone) that would be capable of guiding a user through their transportation planning needs. This means they need to be able to issue queries that are more complex than "how do I get from point A to point B?". The fact that the device may be used in different contexts; such as walking or driving; planning ahead in a quiet environment; or making a decision while in a crowded train station, means that users need a flexible modality of input. How the user issues these queries may vary with the current context. In the settings where typing or pointing on a map, especially on a small screen, is difficult the use of spoken natural language input is very useful.

This dissertation advances the state of the art and brings the community closer to being able to develop such an Intelligent Traveler's Assistant. There are three primary contributions in this dissertation,

1. A model for multimodal[1] urban transportation networks
2. A query language for trip planning
3. A natural language interface

Each of these components fills a key role in my attempt to provide a coherent way to model and query heterogeneous data for the purpose of planning trips in multimodal urban transportation environments. An overview of how these components fit together to form a cohesive system is found in Figure 1. Note that the two components in dashed lines (WordNet and TRANQUYL DBMS) are not implemented as a part of this dissertation. WordNet is a resource developed externally [Fellbaum, 1998], and the DBMS was deemed outside the scope of this dissertation (discussed in more detail later). Additionally, the implementation of a voice-to-text system was also beyond the scope of this dissertation; the NLI operates via text input.

The network model is based on a time-dependent multigraph, which is shown to have a relational equivalent. Using those relations, I define a query language—the TRANsportation QUerY Language (TRANQUYL)—that allows the succinct writing of trip based queries. For the purposes of this dissertation, consider a trip to be a route between, at minimum, two vertices

[1]Here, and in the remainder of the dissertation, multimodal refers to multiple modes of transportation (e.g., bus, train, pedestrian) rather than multiple modes of input/output in a computer system.

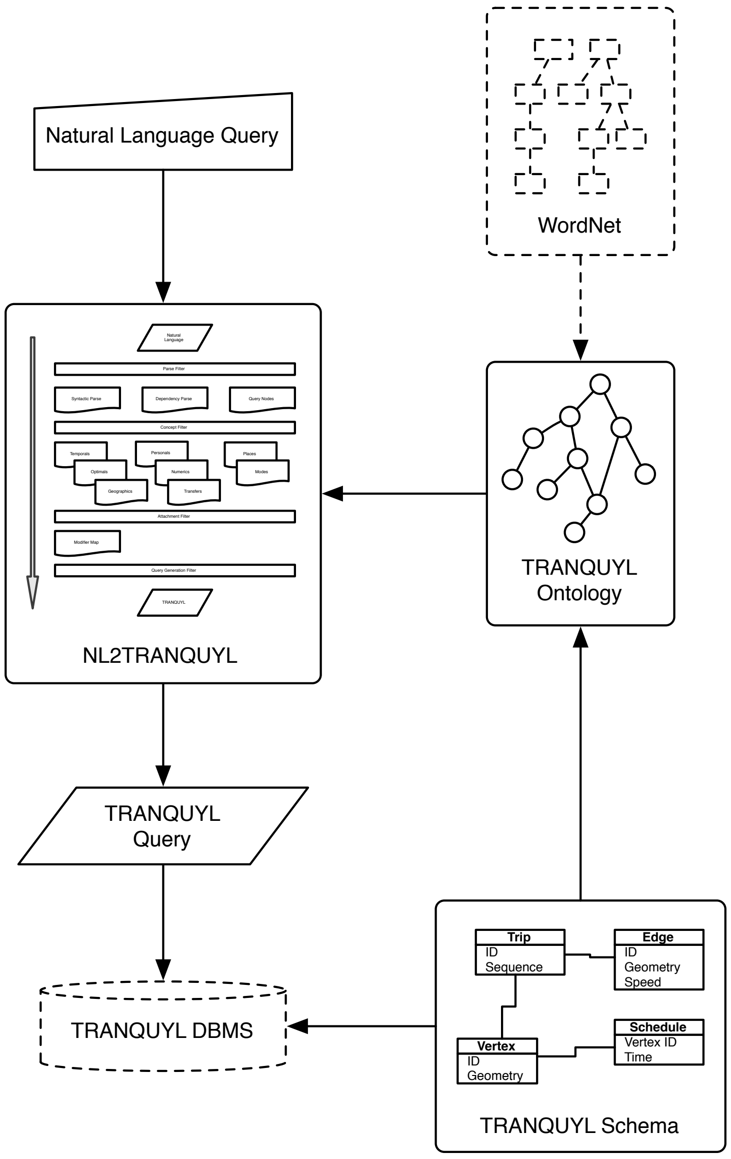


Figure 1. Dissertation components overview

subject to some set of constraints. The model and query language both incorporate spatial and temporal attributes, as well as the probabilistic nature of the temporal attributes. I define the syntax, semantics, and provide a wide range of example queries. Please note that I do not provide a software implementation of the database system; it was determined to be beyond the scope of the research. However, I do present a discussion on query processing and some issues related to implementation.

After discussing the issues related to modeling and querying, I present a natural language interface (NLI) to the query language, named NL2TRANQUYL. Given natural language input, the system effectively translates the query into an appropriate TRANQUYL output. The approach is such that it accepts a wide range of grammar and vocabulary. It strives to distinguish between personal references (e.g., my bank) and generic terminology (e.g., an ATM). It accomplishes this in part by using an ontology as a mediator between the natural language input and the TRANQUYL schema.

The following queries, in English, demonstrate the type of language the NLI can accommodate. Furthermore, they demonstrate types of concepts that I wish to model.

- Arriving by 6:00pm with 95% certainty, find the cheapest route to my hotel.
- Which bicycle route to 600 N. Michigan Ave. allows me to pass through a park?
- What is the fastest transit route from my house to my office that stops at a bank, grocery store, and pharmacist and arrives before 9:00pm for less than $6.50?

In Chapter 2, I provide a discussion of work on trip planning services and usability; spatio-temporal, moving objects, and graph databases; the semantic web and modeling with ontologies; natural language interfaces; and issues of context and understanding as they pertain to natural language systems. Each of these topics played an important role in shaping the direction and shape of the research as I am taking a multidisciplinary approach to solving a multidisciplinary problem. As a side effect, I claim that this work furthers the integration of the fields of natural language processing, the semantic web, databases, and transportation modeling.

The structure of this dissertation is as such: following the related work, in Chapter 3 I present the graph model In Chapter 4, the TRANQUYL query language, followed by the natural language interface (NLI) in Chapter 5. Chapter 6 contains an evaluation of TRANQUYL, the NLI, and the details of a small, web-based demo that uses NL2TRANQUYL. Finally, I end the dissertation in Chapter 7 with a brief conclusion where once again I highlight key contributions and outline potential future directions for this research.

CHAPTER 2

RELATED WORK

2.1 Trip Planning Services and Usability

The transportation informations available to users today generally fall into two categories: form-based and map-based. Many transportation agencies [Bay Area Rapit Transit Planner, 2011; Regional Transit Authority Trip Planner, 2011; Washington Metropolitan Area Transit Authority Trip Planner, 2011] 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). Some, such as TransitGenie [Biagioni et al., 2009] use realtime transit information. If a valid trip can be constructed the user is presented with an itinerary for its execution.

The second common class of planning tools have map-based graphical user interfaces [Google Maps, 2011; Mapquest, 2011]. 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 additional stops along the way in the query is cumbersome if possible at all.

While menu based systems can be used to easily browse for information, their search capacity is predefined and inflexible [Cohen, 1992; Grasso et al., 1998]. They also require physical interaction with the device, which is difficult in eye-busy/hands-busy situations; such as while driving, where a physical input modality may not be appropriate. Additionally, various studies [Cohen, 1992; Grasso et al., 1998; Kaufmann and Bernstein, 2008] show that NL is generally better at expressing certain questions/commands, such as those requiring negation, quantification, and temporal information; additionally, using NL reduces task completion time.

2.2 Databases

This work draws heavily from work done on moving objects, spatial, temporal, and graph databases. In this section I introduce the key literature in each of the areas and offer insight into how this work differs from the literature.

2.2.1 Spatio-Temporal and 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.* [Pelekis et al., 2004] have compiled an extensive review of many of the most important models. Similarly, graph databases have been studied for a long time. Yannakakis provides a good summary of many of the older approaches for graph-theoretic methods in databases [Yannakakis, 1990]. I will highlight some of the key models, but note that my model is neither that of a pure spatio-temporal or graph database—instead, it is situated somewhere in the middle and draws from both.

A model that deals extensively with uncertain queries is the Moving Objects Spatio-Temporal (MOST) model [Sistla et al., 1997; Sistla et al., 1998]. MOST is a data model based on the 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.

Vazirgiannis and Wolfson [Vazirgiannis and Wolfson, 2001] built a model to develop a rudimentary system for modeling road networks and vehicles on them. The roads are modeled as polylines and lack many of the transportation-based attributes required for our approach. The vehicles are modeled as vectors that are assumed to remain on these polylines. The model is better suited for modeling individuals in a transportation network, not the network itself. There is no good way to distinguish between the different modalities, model schedules, find paths, or write the specific queries we are interested in. In general, we find that transportation networks cannot be represented by these geometric primitives and moving objects alone.

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 [Erwig and Schneider, 2002b; Erwig and Schneider, 2002a; Güting et al., 2000] as well as road networks [Güting et al., 2006]. 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. These types natively support temporal evolution. Non-temporal types can be "lifted" and therefore used in temporal queries as well.

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. Our proposed model is influenced by this separation of routes and junctions from their geometry. At this time, the STQL network model is restricted to only the road network, whereas we must consider multiple interacting networks. Additionally, the functionality for finding shortest paths is rather limited, there is no model of temporal uncertainty, and there is no explicit support for resources in the network.

POSTGIS [POSTGIS, 2011] is an open source extension to the PostgreSQL object-relational database [Stonebraker et al., 1990; PostgreSQL, 2011], developed through the Open Source Geospatial Foundation (OSGeo) [The Open Source Geospatial Foundation, 2011]. It supports the Open Geospatial Consortium (OGC)'s [Open Geospatial Consortium, 2011] Simple Features Specification for SQL. Functionality includes support for spatial types (e.g., lines, points, regions), spatial predicates and operators, and spatial indexing. It is not designed for moving objects and therefore does not handle uncertainty.

2.2.2 Graph Query Languages

Querying of graph databases has been studied extensively. Angles and Guitierrez performed a comprehensive survey [Angles and Gutierrez, 2005], but we discuss a number of important models here as well. Dar and Agrawal [Dar and Agrawal, 1993], and Cruz and Norvell [Cruz and Norvell, 1989] 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. Languages such as G [Cruz et al., 1987], G+ [Cruz et al., 1988], and QDB* [Angelaccio et al., 1990] were designed to allow easier query writing for graph queries.

Dar, Agrawal, and Jagadish go on to describe algorithms for optimizing these generalized transitive closure queries [Dar et al., 1991]. Other have studied the problem of optimizing recursive queries [Ordonez, 2005; Houtsma and Apers, 1992], which is important given that for some queries there is an exponential or infinite number of possible paths. However, through optimization of the query plan it is possible to prune the search space dramatically – making efficient computation possible. We address the issues of complexity and expressivity again in Sections 6.1.1 and 4.4.

Güting proposed an object-relational graph database model, GraphDB [Güting, 1994], that supports many similar queries to our proposed language. It too builds on the select, from, where query structure common to SQL. Once again, this model does not address uncertainty. Sheng et al. define an object-oriented query language GOQL [Sheng and Özsoyoglu, 1999] and an algebra to deal with paths and sequences. This language was primarily defined for use with multimedia presentation graphs.

More recently, there has been work on modeling time-dependent graphs. For transportation network modeling, previous work focuses primarily on time-expanded networks where there is one edge per time-frame being modeled [Köhler et al., 2002; Ding and Güting, 2004]. George and Shekhar [George and Shekhar, 2008] proposed a method for aggregating the edges in order to reduce storage space and improve processing efficiency.

Tangentially related to graph databases are RDF [World Wide Web Consortium, 2011b] and the SPARQL query language [Prud'hommeaux and Seaborne, 2011]. An RDF schema is defined by a list of triples of form (subject, predicate, object), and these triples can be visualized as a graph. SPARQL is used to issues queries over RDF, and therefore is tangentially related to graph query languages. However, at this time the most closely related work is used for modeling generic social networks [San Martin and Gutierrez, 2009].

2.3 Natural Language Interfaces

NLIs have a long history in the NLP and databases communities. My work continues in that tradition, but addresses a new domain (trip planning) and attempts to address some of the many problems in previous NLIs: limited vocabularies, strict grammars, and poor understanding of the system's limits have long plagued NLIs [Androutsopoulos et al., 1995]. The proliferation of in-car and portable GPS units is prompting a renewed interest in NLIs. People are unable to type or use a mouse while driving, which limits system input to either times where the vehicle is stopped (which is enforced by some systems) or by voice.

Modern NLIs have taken a number of approaches to improve usability: interactive query formation [Li et al., 2005], clarification dialogs [Kaufmann et al., 2006], the use of WordNet to boost vocabulary [Lopez et al., 2006], and statistical methods to improve disambiguation of ambiguous terms [Meng and Chu, 1999]. NLIs have been increasingly applied to ontologies [Wang et al., 2007; Bernstein et al., 2005; Bernstein et al., 2006], especially in the domain of question answering [Garcia et al., 2006; Lopez et al., 2006]. In some cases, only shallow NLP techniques are used and partially formed queries (i.e., sentence fragments) are allowed [Kaufmann et al., 2007]. This relies on keywords alone rather than grammatical relations.

In many ways, NL2TRANQUYL is situated between the aforementioned works. I use WordNet as a straightforward method of expanding the vocabulary, but also allow for other word forms (e.g, colloquial terminology) to be captured in the TRANQUYL ontology. The use of the ontology resource in my manner is unique in the literature I have seen. It guides NL2TRANQUYL in interpreting what the concepts are and how they are related. In most systems the software is either querying the ontology or simply using it for an additional lexicon.

Similarly, the system functions with fully grammatical sentences as well as some ungrammatical sentences and fragments. This is because I make minimal use of the constituency parse[1], and focus on heuristics and the dependency parse[2] which is generally more stable across changes in syntax. I also consider the work distinct as the intended users of NL2TRANQUYL are not experts who are using an NLI to access their data, but rather, average people accessing information in a method that is familiar to them.

Other related work concerns transportation applications proper, such as those studying the best type of interface between a driver and his/her intelligent vehicle, e.g. the best combination of input / output modalities [Bernsen and Dybkjær, 2002; Buehler et al., 2003; Coletti et al., 2003; Geutner et al., 2002; John et al., 2004; Salvucci, 2005]. Of those the closest to the work is the VICO project [Coletti et al., 2003; Geutner et al., 2002], aimed at developing an in-car assistance system for accessing tourist information databases and obtaining driving assistance. However they focused on speech understanding rather than understanding of complex queries that are in turn translated into a query language.

Additionally, there have been projects focused on multimodal (in terms of input) navigation of urban areas [Johnston et al., 2002; Gruenstein, 2008; Gruenstein and Seneff, 2007], however they have largely dealt with finding resources rather than planning trips or itineraries. The

[1]Constituency (a.k.a., phrase structure or structural) parses break a sentence into its constituent components. The result is a tree of more and more granular concepts–moving from phrases (e.g., noun phrases, prepositional phrases, verb phrases) to smaller constituent components (e.g., adjectival nouns, compound nouns) until single base types are reached (e.g., noun, verb, adjective). The representation only captures syntactic relationships related to how the sentence is organized. Figure Figure 6 contains an example constituency parse.

[2]A dependency parse is another form of syntactic parse. The parse output takes the form of a graph where words are nodes and the relationships, or dependencies, between them are modeled as labeled edges (e.g., adjectival modifier, noun modifier, cardinal number). This parse form may be more appropriate for extracting relationships between words. Figure Figure 7 contains an example dependency parse.

Let's Go project [Raux et al., 2005; Black et al., 2011] is a telephone-based information system for the Pittsburgh area bus system that answers questions about the current status of busses and bus routes. The system is dialogue-based, but is restricted to a much narrower range of question types.

Finally, I would like to point out some key differences to perhaps the most well known and relevant related work—the Air Travel Information System (ATIS) [Hemphill et al., 1990] corpus and related projects. The set of potential constraints on air travel is generally much smaller than that in urban transportation. Airline passengers must select an origin and destination, a departure time, and possibly a constraint on the number of layovers if possible. The only real optimization metrics are the cost and duration of the trip. The actual path taken is largely irrelevant to the user (e.g., preferring to fly over Montana rather than Nebraska) while the layovers are generally considered a nuisance rather than a point of interest to specify.

In an urban environment, the path you take may actually be relevant depending on the user's wishes. It is not unreasonable to wish to include a park on your walk home from work, or avoid a neighborhood known for heavy congestion. You may wish to minimize cost, duration, or number of transfers depending on not only your destination, but also how you plan to travel. The temporal constraints can be much more granular as you are not restricted to a handful of departures each day.

2.4 Context and Understanding

There have been many approaches for utilizing textual, or linguistic, context for interpreting ambiguous language [Jurafsky and Martin, 2009]; however, there have been far fewer projects that attempt to utilize non-linguistic context for such purposes. Much of the use of non-linguistic context has been for automatic speech recognition.

Coen *et al.* [Coen et al., 1999] use non-linguistic context in a NLI for an intelligent room. Users are able to issue commands that either return information (e.g., current room status, web search) or change the properties of the room (e.g. temperature, lighting, TV settings). The room possesses a set of speech recognition grammars that can be used for different tasks. The grammars that are active (i.e. being listened for) depend on where in the room the user is located as well as what he or she is doing. Kray and Porzel [Kray and Porzel, 2000] look at how spatial context can be used in NLIs. They are interested in answering questions such as "what building is this?" as well as provided user-centric spatial information like "you are behind the castle". There have been other attempts [Roy, 2005] at connecting linguistic and non-linguistic context in a variety of domains. However, much of the work focuses on connecting perception to language (e.g., learning what 'red' means in terms of sensor input). Flischman and Hovy [Fleischman and Hovy, 2006] utilize non-linguistic context in what they call a *situated* natural language interface (NLI). The interfaces allows a user to interact with a virtual environment for military training exercises. Spoken commands are used to direct the behavior of agents within the simulation. The speech recognition component uses the user's most recent sequence of actions to better estimate the probability for utterances. The premise is that in certain contexts (actions or sequences of actions) certain language usage is more probable. Interpreting deixis[1] like proposed in this dissertation still remains largely unexplored.

2.5 Semantic Web and Ontologies

In this research I use ideas from the Semantic Web in order to facilitate the translation of natural language queries into TRANQUYL queries. The Semantic Web is a concept envisioned by Berners-Lee [Berners-Lee et al., 2001]. Web pages, as they exist generally, are designed to be read and understood by humans. The information is presented visually and it is assumed that the human reader can understand the words and their relationships to one another. The idea behind the Semantic Web is to attach machine interpretable semantics to the information on the page so that a computer can make sense of it.

This concept is best illustrated with an example. Currently you are often able to browse local phone directories, maps, and physicians' websites. You may also track your personal schedule using an application either online or on your local computer. Performing a task such as finding a doctor who is available for an appointment next week would require you to access each of these resources manually. There is no way to simply issue a command to your computer to do so. All of the information is available online, but it in a purely syntactic form. You, as a human, are able to read the displayed text, but the web browser you use has no understanding of what the text means. All of the information is treated as words or symbols with no meaning.

[1]Deixis is a linguistic phenomena in which the meaning of a word depends on its context. Examples include pronouns like *he* and places like *here*, which can only be interpreted in a given context.

The semantic web aims to attach meaning to this text so that the browser could complete the task for the user.

In order to make things interpretable by a machine, one must explicitly define the meaning (semantics) of the information on the page. For the above example, the computer must understand that a doctor has a schedule, and the times in the schedule correspond to appointments. The appointment is a time when the person is allowed to visit the doctor. If there was an explicit model for how these concepts are related, and the information on the page were annotated with the relevant concepts in the model, it would be possible to draw the necessary conclusions about the problem as a whole. This is in fact what the Semantic Web aims to do, and the formal model used is an ontology: a formal conceptualization of a domain of knowledge.

In the ontology model, *classes* represent real world things (people, places, objects, activities, etc.) and have *properties* (age, number, location etc.). The semantic relationships between these classes are defined which allows *semantic inferencing.* Data are stored as instances of classes within the ontology. When queries are executed the system is able to take advantage of the semantic inferencing which is not possible in traditional syntax-based databases.

Ontologies are often visualized as graphs where the concepts are nodes and the relationships are directed arcs. The underlying implementation is independent of the theoretical groundings. RDF (Resource Description Framework) and RDFS (RDF Schema) [World Wide Web Consortium, 2011b] provide an XML-based approach to describing the relationships between concepts and their properties. OWL (Web Ontology Language) [World Wide Web Consortium, 2011a] is another language for describing ontologies that builds in semantics for reasoning and is based on description logic [Baader et al., 2007]. In most applications OWL is serialized into XML/RDF before it is used in production. A more detailed discussion of these languages may be found in Antoniou and van Harmalen [Antoniou and van Harmelen, 2004].

At this time, I should mention WordNet [Fellbaum, 1998], which is an annotated lexical resource that connects words according to a number of different relationships. For example, it captures that a car is a type of automobile; and that a wheel is a part of an automobile. WordNet is not an ontology in the strictest sense of the term. Or at least, it was not initially designed using the principles espoused by the Semantic Web community. None the less, it does provide a formal model for how words are interrelated in terms of hypernymy, hyponymy, meronymy, and metonymy. In this dissertation I use WordNet to extend the vocabulary that NL2TRANQUYL recognizes. I am not the first to propose such a method. Budanitsky and Hirst [Budanitsky and Hirst, 2006] compiled a comprehensive discussion of techniques for determining lexical semantic relatedness. They also implement and compare five algorithms and compare the results against human performance, as well as their performance within larger NLP frameworks.

2.6 Transportation Models

Transportation networks have long been modeled as graphs. Going back to the 1950s, the Chicago Area Transportation Study (CATS) modeled road networks using graphs for travel demand forecasting (i.e., determining future road usage). The "four-step" model they proposed used a graph to connect zones in the region, where each zone was some aggregation of blocks or neighborhoods. The graph itself did not include all of the roads, instead including only major thoroughfares. This approach, and many built upon it, are well studied in transportation planning [Hensher and Button, 2007].

Over the years, the simple graph models used in the CATS model have been expanded significantly. Pallottino and Scutellà provide a discussion of many of the now classical approaches as well as more modern innovations [Pallottino and Scutella, 1997]. Recently, a significant amount of research has been published on modeling multimodal networks. Many of these focus on how to compute shortest paths subject to various constraints. Many focus on the problem of scheduling (non-probabilistic) for public transportation [Lozano and Storchi, 2001; Ziliaskopoulos and Wardell, 2000; Lozano and Storchi, 2002]. Newer works add in constraints for number of transfers [Zografos and Androutsopoulos, 2008; Bielli et al., 2006; Ayed et al., 2008].

While there are subtleties that vary across the models, there are a few constants. In each, the graph edges represent some path in the network; whether it is a single link of road or series of links. The vertices represent the intersections of those paths and points of transfer. How many, and which, attributes are assigned to the edges and vertices vary. Travel time, length, capacity, mode, turn restriction, and many more are common. My model, described in the next chapter, builds on many of these common themes. It differs in that it also models the resources (e.g., homes, stores, banks) that the network connects. I feel that this is an important and significant improvement over the discussed models. Additionally, the attributes in the discussed models are not probabilistic in nature.

In addition to the graph models that are used for shortest paths problems, the transportation field has worked on developing ontologies to model the functionality of the transportation network. In Section 5.2 I will present my own model, which differs from the previous literature in some important ways.

There are high level ontologies such as SUMO [Suggested Upper Merged Ontology, 2011] and OpenCyc [OpenCYC, 2011] that contain some information about transportation systems, but the focus of the ontologies are general knowledge and are therefore much too broad for the purposes of this dissertation. When I began my research in this field, there were very few transportation specific ontologies. Those that existed focused too heavily on the mechanics on the transportation network [Fonseca et al., 2000; Lorenz et al., 2005; Benslimane et al., 2000]. My research is not focused on how the transportation system is built or organized, beyond the minimum necessary to guide users through it. Modeling how freight interacted with the road network and shipping ports is far beyond my scope, and would only serve to introduce noise into the model.

Because these preexisting models were too focused on the minutiae of the transportation network—rather than how a user interacts with it—I chose to develop my own ontology from the ground up. Additionally, it ties in some spatial and temporal attributes and relationships missing from the transportation specific ontologies. This is done to facilitate the use of natural language, which was not a concern of the mentioned ontologies.

Over the last couple years, others have begun to look at ontologies in order to support trip planning. By the time these works were published, the ontology development for this dissertation was complete or nearly so; however they still deserve discussion here. Houda et al. designed an ontology to support user travel planning [Houda et al., 2010]. Their ontology carries many similarities to my own in that it contains information about route selection preferences and potential resources in the network. Before that, Niaraki and Kim proposed an ontology-based personalized route planning system [Niaraki and Kim, 2009]. The ontology is used to make decisions on which optimization and selection criteria are to be used in route planning. It does not model the resources in the network or the user's interaction with them. Once again, these ontologies were not designed to facilitate the use of natural language.

CHAPTER 3

THE MULTIMODAL URBAN TRANSPORTATION NETWORK AND ITS MODEL

3.1 A High Level Network Description

3.1.1 Network Description

Consider the transportation network to be the collection of *pathways* (e.g., roads, railways, sidewalks) and the resources they connect (e.g., homes, schools, banks, parks). The network is *multimodal* in that it has more than one *mode* of transportation (e.g., pedestrian, automobile, bus). The multimodal pathways interact with one another and the resources in the network in a complicated manner that brings about many interesting modeling considerations.

TABLE I

MODES AND PATHWAY TYPES

	Road	Freeway	Bike Lane	Sidewalk	Railway
Auto	X	X			
Bus	X	X			
Pedestrian				X	
Bike	X		X		
Train					X

First, consider the interaction between the physical pathways and the vehicles or people that travel on them. Table I lists the five modes and five pathway types as well which modes are allowable on the types. The different pathways do not overlap spatially, except at intermodal crossings or intersections. Illustratively, a railway cannot use the same space as a road, except at a railroad crossing. However, certain modes are allowed to use multiple pathways. There is no separate pathway for the bus, because it drives on the same roads as automobiles. Bicycles may have separate bike lines in some areas, but in general they are required to ride on the same roadways as the automobiles and busses, while they are generally prohibited from using freeways. This division of pathways and modes is biased towards North American networks, with the most consideration towards Chicago. There are other transportation networks with slightly different configurations. The proposed model could readily be modified to accommodate a variety of transportation systems with no changes to the fundamental modeling paradigm.

While the pathways of different types do not share the same physical space, there are designated points of intersection to allow for people to change between different pathways and modalities. Call the point that links two different pathways together a *connector*. Furthermore, break this down into *interpathway* and *intrapathway* connectors[1]. An example of the former would be a freeway interchange (where the two intersect) while the latter would be a freeway onramp (to go from the road to freeway or vice versa). The pathways may be labeled with

[1]Throughout this dissertation, a number of definitions rely on the prefixes *inter* and *intra*. These follow their standard definition of 'between' and 'within', such as interstate commerce (between states) and intrastate commerce (within a single state). The prefixes to denote the difference of between modes versus within a single mode.

a *name*. The collection of pathway segments with the same name, in the case of roads and freeways, are known as *streets*.

The connectors may possess different *control mechanisms*, such as stop signs, traffic lights, roundabouts, and turn restrictions. The controls determine how, and importantly, when a user may move from one section of pathway to another.

It is necessary to consider how the different modes act. When using the automobile, pedestrian and bike modalities, the user is free to go where they please on their allowable pathways. Users riding public transportation (i.e., busses and trains) must follow predetermined *routes*. A route is a directed path along the necessary type of pathway. In order to change between routes, a user must *transfer*. Once again, further subdivide a transfer into a *intramodal transfer* and *intermodal transfer*. The former occurs when a change is made from one route to another within the same modality (e.g., a user changes from the #12 to #8 bus at a bus stop). An intermodal transfer occurs when a user moves between modes (e.g., moving from a bus to a train, a pedestrian getting into an automobile). The distinction between modes, pathways, and routes is important as different attributes of the transportation network are tied to these different sources.

For the public transportation aspect of the network there is another complication: schedules. Unlike the other modalities, busses and trains run on a limited schedule that the user does not have control over. For the purposes of this model, assume that schedules are specified as a series of *departure times* (and consequently *arrival times*) at *transit stops* on a route. The collection of departure and arrival times for all of the transit stops on a route is known as its *schedule*.

Of course, in many transportation networks the published times are not always adhered to exactly – busses and trains are frequently either early or late. This introduces the problem of uncertainty into the network. The second source of uncertainty is congestion on the roads and freeways. Automobiles are not always able to travel at the posted speed limit.

The final component of the transportation network are the facilities or *resources* that it provides access to. The banks, grocery stores, school, parks, residences, and other places of interest are the driving force behind the transportation network—these are the places that people want to go to. Therefore, it is imperative to include such resources in the transportation model. We find that these resources can be divided into two components: specific resources and a *class* of homogenous resources. For example, Mom's Generic Bakery at 100 N. Main St. versus the entire collection of bakeries. I feel that this is a reasonable delineation as there are times where users are interested in both: needing to pick up your dry cleaning requires visiting a specific store, while picking up a loaf of bread for dinner on the way home can be satisfied by any bakery. The resources are accessible along some point on one or more pathways—either at connectors or *pseudo-connectors*. A pseudo-connector is a point along a pathway that does not contribute to the topology of transportation network, rather it acts as a point of access for a resource.

3.1.2 The Problem of Trip Planning in the Network

Define the problem of *trip planning* to be finding a path or route between two locations, subject to a set of *constraints* and an *optimization*. The most simplistic trip would be "the shortest path from point A to point B". This would correspond to the query, "Find the shortest way to my home from my office". However, I am interested in more than finding the simple shortest path.

First, given a *trip* (the path, constraints, and optimization), divide it into smaller components. Each trip is composed of a series of *legs* and *transfers*. A leg is the unit of travel between transfer points. Recall that a transfer occurs either when changing modes (intermodal) or when changing between public transportation routes (intramodal). A transfer necessarily occurs at a connector, but not all connectors are transfers (e.g., turning at an intersection is not a transfer). Therefore, the path for any trip can be viewed as an alternating sequence of legs and transfers. Note that a transfer takes time, therefore need to consider the *minimum transfer time* when calculating the duration of a trip.

Subdividing the trip path into legs is important as a trip may utilize multiple modes of transportation. Additionally, a trip may include intermediate points that a user wishes to visit. For example, "Find the shortest way to home from work that also stops by a bakery". In this case the trip must include an intermediate vertex that has a resource of type 'bakery'. Such an intermediate point is called a *stop*. I claim that a stop involves a transfer or at minimum a *pseudo-transfer*. If the user is traveling on public transportation, they must disembark from the bus or train to make the stop and complete the task. After the stop is completed they may continue on their way. Similarly, if not using public transportation the user must cease their walking, driving, or bicycling in order to complete their task before continuing on their way. Because the user may continue on their previous transit route or non-transit mode, there is a specification for a pseudo-transfer that takes this into consideration.

In addition to the intermediate stops, a trip may include various spatial and temporal constraints. The temporal constraints are intuitive to anyone who travels: specifying departure and arrival times, and knowing or bounding the travel time of the trip as a whole. These constraints may also apply to specific legs or stops of a trip: spending a certain length of time at a stop or limiting how long you walk. As discussed before, there is a great deal of uncertainty in transportation networks and this uncertainty presents itself when dealing with temporal constraints. When specifying that the trip ends by 6:00pm, the reality is that it is difficult to ensure that such a trip is absolutely feasible given the current network state. For this reason, a user may wish to place a *certainty constraint* on the trip—ensuring that given all of the temporal constraints for the trip, all of the constraints are met with the specified likelihood.

Spatial constraints are also a key aspect of trip planning. Specifying the inclusion, or exclusion, of a region might be the most common example. A user may wish to pass through a park or avoid a specific neighborhood. Cities, counties, and other politically demarcated extents; as well as geographic features such as rivers could also be relevant to the planning.

The final type of constraint are those applied to the modes, pathways, connectors, and transfers. The most obvious is limiting the modes allowed in a trip; similarly the user may wish to limit which pathways are allowed, be it a specific road or an entire class (e.g., highways). Limiting the number of transfers or minimizing the number of signalized intersections is also important. If the information is available, users may wish to place constraints on fuzzier concepts such as the safety or comfort of the modes or pathways.

Optimizations are the final constraint on the trip, and I separate them from the previous constraints for a reason. It is unlikely that a user is interested in just any path, but rather some ‘best’ path. It may be the shortest length, minimal duration or least expensive, but it is generally optimal according to some metric. The optimal path is selected such that all of the other constraints are ensured. Only one optimization criteria is allowed for a given trip.

3.2 A Graph Model

3.2.1 The Transportation Network as a Graph

Define a multimodal transportation network, $U = (n, M, P, C, A, R, F)$ where,

n is a graph

M is a set of modes

P is a set of pathway types

C is a set of connector types

A is a set of attributes

R is a set of routes

F is a set of resources

The set of modes, M is a set of labels that specify the modes of transportation in the network (i.e., automobile, train, bus, pedestrian, bicycle). Similarly, P is the set of labels specifying the pathway types present in the network (i.e., road, freeway, bike lane, sidewalk, railway). And C is the set of connector types—both intramodal and intermodal—found in the network. A is the set of attributes that describe the network. For each $m \in M$, $p \in P$, and $c \in C$; define subsets of the attributes, A_m, A_p, and A_c to be the attributes that correspond to each mode, pathway type, and connector type. All of the attributes are enumerated in Table II–Table IV and will be discussed shortly—as soon as the remaining basic graph concepts have been introduced.

TABLE II

PATHWAY ATTRIBUTES AND THEIR TYPES

All Pathways	geometry:line, speed:distribution, resources:set, name:string, id:int, width:real
Road	has_toll:boolean, is_one_way:boolean, has_parking:boolean, time:period, num_lanes:int, has_shoulder:boolean
Freeway	has_toll:boolean, num_lanes:int, time:period, has_shoulder:boolean
Bike Lane	is_protected:boolean, is_independent:boolean
Sidewalk	
Railway	

Define the network as a directed multigraph $n = (V, E)$ where V is a set of labeled vertices and E is a set of labeled edges. Each edge is defined as a 4-tuple (v_1, v_2, p, ϕ) where v_1 and v_2 are the start and end points of the edge, p is the pathway type, and ϕ is a function that maps the attributes defined for the pathway type (i.e., A_p) to values. Each vertex is defined as a 3-tuple (v, c, ψ) where v is the vertex, c is a connector type label, and ψ is a function that maps the attributes defined for the connector type (i.e., A_c) to values. A mode is a tuple (m, Ω)

TABLE III

MODE ATTRIBUTES AND THEIR TYPES

Automobile	
Pedestrian	
Bike	maximum_speed
Train	
Bus	

TABLE IV

CONNECTOR ATTRIBUTES AND THEIR TYPES

All Connectors	resources:set, geometry:point, id:int
Connectors	delay:distribution, has_stop_sign:boolean, has_traffic_light:boolean, turn_restrictions:set
Pseudo-Connectors	on_edge:edge, offset:real

where m is a mode label and Ω is a function that maps the attributes defined for the mode type to values.

The road and freeway edges have a label for *time* that the other modes do not have. This is because the *speed* of those pathways are dependent on the time of the day due to congestion. Assume that bike lanes, sidewalks, and railways are not effected by congestion the same way roads and freeways are. Therefore, for each time interval in the modeling period, there will be a set of edges for the roads and freeways in the graph that represent the current conditions of the pathway rather than the idealized condition (i.e., what the road is supposed to be like). The bike lane, sidewalk, and railway edges do not suffer from this duplication effect as the speed of the links does not change during the day[1].

Define subnetworks n_{roads}, $n_{freeways}$, $n_{bikelanes}$,$n_{sidewalks}$,and $n_{railways}$ to be the networks comprised of pathways of the given type. In the case of the roads and freeways, their time-expanded edges are collapsed (i.e., only one edge between vertices, ignoring temporal information).

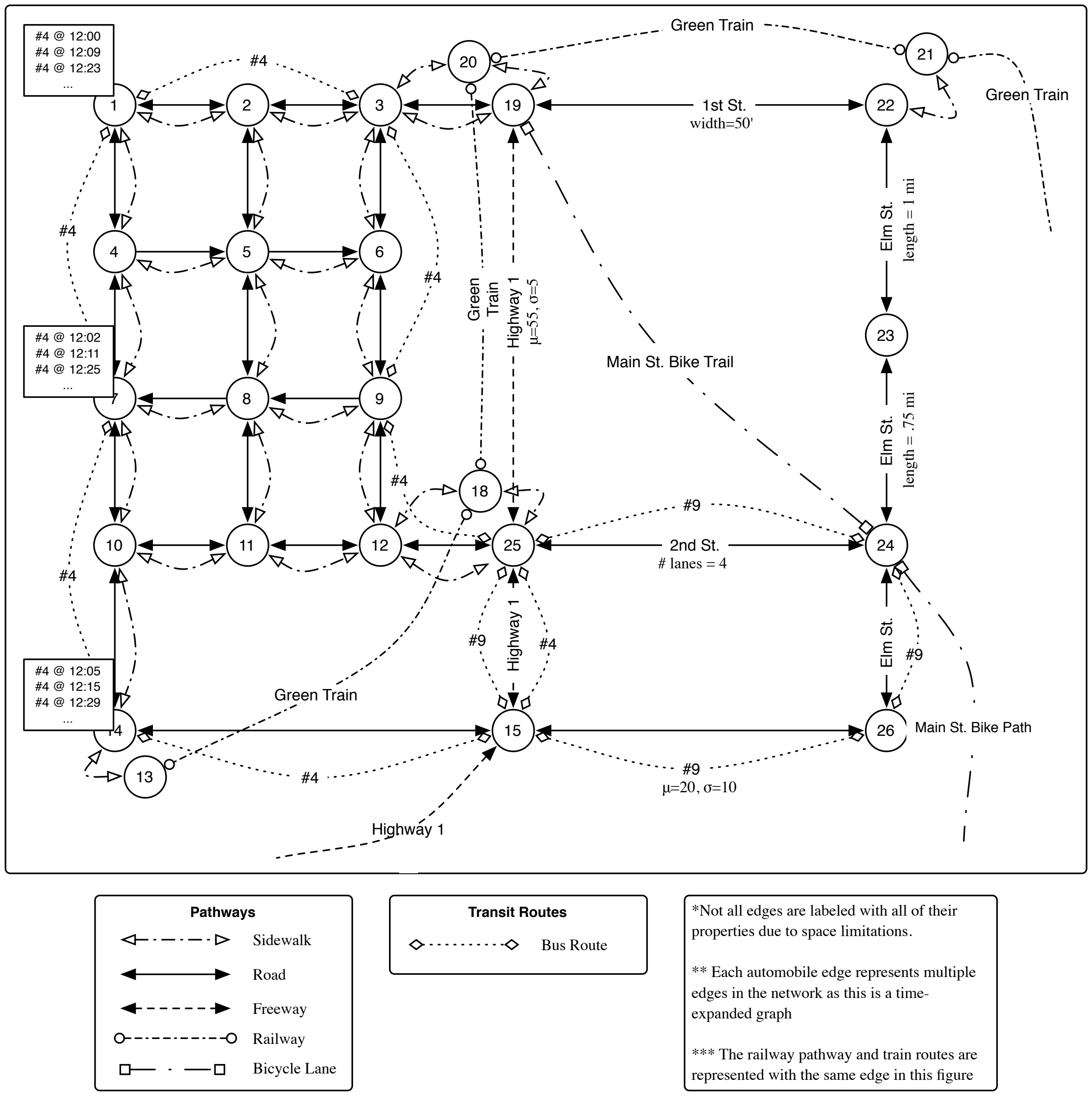
To continue with attributes, the *geometry* captures the actual geometry of the edge (i.e., where in 2d space the path exists). I model it as a line (or a piecewise combination of lines), even though in the real world a road cannot be entirely represented as a line (i.e., there is width). To compensate, add an attribute *width*. Each edge also has a *speed* attribute that denotes the speed at which a user can travel along the edge. We model it as a distribution, as

[1]This follows the standard time-expanded representation of time-dependent networks. It is possible to define an equivalent time-aggregate representation.

not only can the speed change throughout the day, but the reliability of the speed information is generally less than perfect. All edges are labeled with a *name* representing the name of the path and a unique *id* number for identification. Finally, each edge has a *set of resources* that are accessible at some point on the edge. More details on resources will follow shortly.

Some of the pathway types do have their own attributes. For the bike lanes, define two boolean attributes: *is_protected* and *is_independent*. There are a number of different types of bike lanes, and these attributes attempt to capture them. Generally, the bike lane is a shared portion of the road where automobiles are also present. In some areas, the lanes are protected from automobiles by a physical barrier. There may also be bike lanes that are entirely independent of the road system (e.g., specific commuter paths, leisure trails). The roads and freeways have attributes to capture whether or not they are toll roads, if they have a shoulder, and how many lanes they possess. Users may be interested in this for either cost or safety reasons. Roads have additional properties to denote the presence of on-street parking and whether it is a one-way road.

Before continuing, Figure 2 contains a simplified example graph for a very small network. It is meant to aid in the reader's intuitive understanding of the model rather than containing every detail. It shows how multiple pathways connect to a single connector. Note that the road and freeway pathways show only a single edge between connectors—in the model each of these edges is represented multiple times as it is a time-expanded graph. Similarly, the railway pathway and train route have been combined.



*Not all edges are labeled with all of their properties due to space limitations.

** Each automobile edge represents multiple edges in the network as this is a time-expanded graph

*** The railway pathway and train routes are represented with the same edge in this figure

Figure 2. An example network graph

At this time, there is only one defined mode-specific attribute: a *maximum speed* for bicycles. It is assumed that bike lanes are correctly labeled with their speed, which reflects how fast bicyclists travel, but bicycles are also allowed on roads that may have speed limits higher than bicyclists can be expected to reach. For this reason, define a global maximum speed for the mode so even if a portion of a roadway with a 55mph speed limit is used, more accurate durations of trips can be calculated if the trip involves a bicycle.

Next are the attributes for connectors, $A_{connectors}$, listed in Table V. Similar to edges, each vertex is labeled with a point representing its geometry, an integer identifier, and a set of resources accessible at the connector. Earlier I discussed the difference between connectors and pseudo-connectors; the former control the connectivity of the transportation network, while the latter simply provide access points to resources. The pseudo-connector's *on_edge* attribute captures which edge it lies on, and the *offset* captures the distance along the edge. This is simply to capture where the pseudo-connector exists. The true connectors have attributes to capture how they effect the transportation network. There are boolean attributes for whether or not the connector possess a stop sign or traffic light. Additionally, there may be a set of turn restrictions, where each restricted turn can be modeled as a pair of edges, one incoming and one outgoing from the connector, (e_1, e_2) that cannot join to make a path. Each connector also has a delay distribution to capture any delays passing through the connector may present (e.g., an intersection with a stop sign is likely to add 20 seconds to your trip).

Finally, I define the set of resources, F. Each resource has values for the attributes $A_{resource}$, defined in Table V. The integer identifier is unique to each resource. The *name* attribute

TABLE V

RESOURCE ATTRIBUTES AND THEIR TYPES

Resources	id:int, name:string, class:string, access_point:connector, accessible_period:period

corresponds to the name of the place (e.g., Mom's Coffee Shop) while the *class* captures the class of the resource (e.g., coffee shop, grocery store, bank, dry cleaner). The *access_point* is the connector (or pseudo-connector) where the resource is accessed. The *accessible_period* is the time period for which the resource is available—analogous to the hours of operation. The locations of the resources are cross-indexed as the edges and connectors have attributes that specify which resources they provide access to. This is important for making query processing efficient.

For ease of discussion, define derived sets of roads from the network. Define a road as a tuple $(name, path)$ where $name$ is the name of the road and $path$ is a path in n_{roads} where all of the edges have the same given value for $name$. For example, "Halsted Street" would be the sequence of edges in the subset of the network consisting of 'road' pathways that are labeled "Halsted Street". Using the freeway, sidewalk, and bike lane subnetworks, derive named sets of each in the same manner. The transit aspect of the network is slightly different and discussed next.

So far I have defined what is basis to the physical network: the pathways and their connectors. For the non-transit modes (i.e., pedestrian, bike, and automobile), these pathways define where the users may travel. However, recall that for the transit modes (i.e., busses and trains) the movement is restricted to specific routes. Each route $r \in R$ can be defined as a 4-tuple $(\nu, m, D, name)$ where ν is a collapsed path in n, m is a mode, D is a schedule, and $name$ is an identifier for the route. Say that ν is a collapsed path because the vertices correspond to the transit stops (at connectors), not all connectors are necessarily a transit stop (i.e., the bus does not stop at every intersection). Therefore, the vertices in ν correspond to the stops on the transit route, and the edges in ν correspond to edges, or a sequence of edges, in n.

The schedule for a route, d_r can be viewed as a set of tuples of form (c_r, δ) where c_r is the connector representing the transit stop and δ is a departure time distribution. The number of tuples in the set is equal to the numbers of departures for all of the stops on the route for the modeled time period (assume one day). For now, δ can be thought of the expected departure time (i.e., what it says on the schedule), and I will discus the distribution aspect in greater detail in Section 3.2.3.

From the schedule, I derive one more concept: a run. A run, κ_r^δ can be thought of as an instance of a route r that begins at time δ. That is, it is a sequence of connectors and their corresponding departures, beginning at the first connector on the route and ending at the final connector on the route. From a run, derive one last attribute: the arrival time. The arrival time is the time (or distribution of times) at which one arrives at a connector, having departed a previous connector. The time between the departure at one stop and the arrival at the next stop is the time it takes to travel between the two.

3.2.2 Trips in the Network Graph

We define a trip τ as a time-dependent alternating sequence of vertices and edges in n, written $\tau =< v_0, (e_0, t_0), v_1, (e_1, t_1), ..., v_{n-1}, (e_{n-1}, t_n), v_n >$, where (e_i, t_i) denotes the time at which the edge was entered, and $t_0 < t_1 < ... < t_n$. The times are associated with the edges as the time-expanded multigraph has labels for times and speeds on the edges for the non-transit modes, and for the transit modes the departure times at connectors can be viewed as the time the edge is entered.

We define two functions: $departure(\tau)$ and $arrival(\tau)$ that return the departure and arrival time respectively of the trip. The departure is the time at which the user enters the first connector and the arrival is the time at which they arrive at the last connector. We can also define a duration function, $duration(\tau)$ that returns the length (in time) of the trip. We discuss the problem of uncertainty in the next section. For now, assume that any value is the expected value of a distribution function.

Going back to the definitions in 3.1.2, it was shown that a leg is a sequence of vertices and edges, where the start and end vertex are transfers or pseudo-transfers. Define a leg $l =< v_0, (e_0, t_0), v_1, (e_1, t_1)..., v_{n-1}, (e_{n-1}, t_n), v_n >$ to be a sequence of alternating vertices and edges where each edge has the same mode label and, if present, is part of the same route. Therefore, a trip can be written $\tau =< \rho_0, l_0, \rho_1, l_1, ..., \rho_n >$ where ρ is a transfer and l is a leg. The transfer ρ is the last vertex in l_i and the first vertex in l_{i+1}, i.e., the shared vertex. Recall that a vertex is called an *intermodal transfer* if the joined legs are of different modes, and called an *intramodal transfer* if the joined legs are of the same mode but different routes. We call ρ_0 and ρ_n, the beginning and ending transfers, *null transfers.*

To demonstrate, Figure 3 depicts a trip through the network shown in Figure 2. The graph topology is the same, though minor spatial modifications were made for formatting purposes. The trip begins at origin vertex O and ends at destination vertex D, both marked in a double line. The first leg of the trip is of the mode *pedestrian*, the second *bus*, the third once again *pedestrian*, and finally the *train* mode is used. The three intermodal transfers are denoted by a bold line on vertices 14, 1, and 20. Bold lines are drawn to show the four legs.

Once again, define departure and arrival functions for both legs and transfers. The definition for the departure and arrival functions for the legs follows exactly from that of the trip. The arrival time of a transfer is equal to the arrival time of the previous leg. The departure time of the transfer is equal to the departure time of the subsequent leg. The time between these values may be non-zero. In Section 3.2 I mentioned that connectors, and therefore transfers which must occur at connectors, may have a delay attribute that captures the expected delay for traveling through that connector (e.g., the time it takes to exit the train and walk to the street). Therefore, like for the trip, I can define the duration of both legs and transfers.

In addition to legs being separated by a transfer as described above, I also introduced the notion of a pseudo-transfer. These pseudo-transfers capture what happens when a user stops at a vertex in the trip. Recall the example of a user beginning a trip, stopping at a bakery, and then resuming the trip. These pseudo-transfers occur at pseudo-connectors (i.e., the locations of

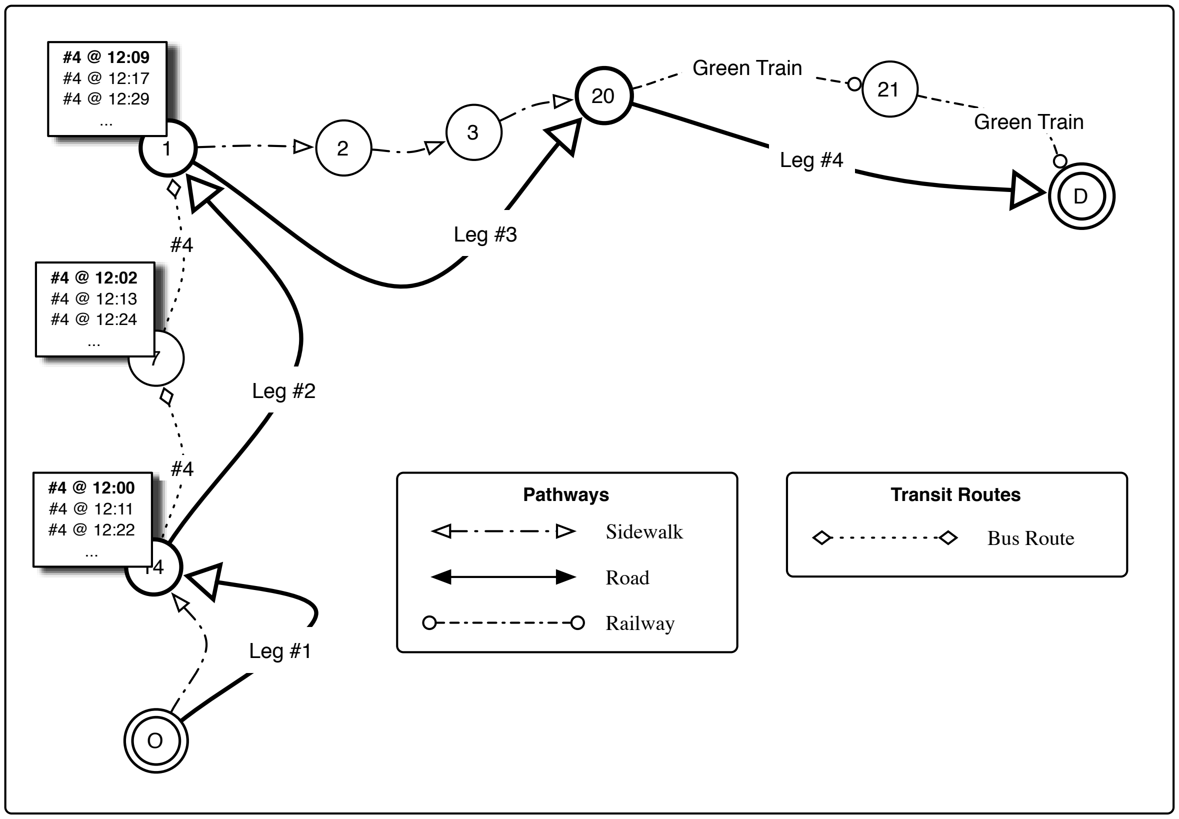


Figure 3. An example trip

resources). When using the non-transit modes, assume the user simply pauses at the necessary vertex, completes the task, and then resumes travel using the same mode. However long they are paused at the vertex is considered the *duration* of the stop. We do not model the fact that the user may have to get out of their car or dismount their bike—I treat it if as though there is no change of mode. If the user is riding transit, they may have to get off the bus or train and walk to the destination, and in that case there is a proper intermodal transfer. The third, and final, case is that the transit stops right at the necessary vertex (e.g., a bus stop at the bakery). In this case, call it a pseudo-transfer as the user must exit the transit vehicle, perform the task, and then reenter a transit vehicle. We call it is a pseudo-transfer because the users are not changing modes or routes, they are only changing runs.

3.2.3 Modeling Uncertainty in the Network Graph

Ideally, the exact speed of every edge in the network at every time and transit schedules would be known and perfectly reliable. Unfortunately, this simply isn't the case in the real world and therefore it is important that our model captures this uncertainty. In the previous sections I discussed potential sources of uncertainty: speeds on edges, delays at connectors, and schedules (i.e., departure times). We said that each of these could be modeled as some probability distribution. For our purposes, any bounded distribution that has a mean and variance is acceptable. We require that the distribution be bounded, as infinitely long tails of minimal mass are uninterpretable in our problem domain: negative speeds and times do not make sense.

Examples of such distributions are truncated normal distributions[1] and triangular distributions. For the purposes of this dissertation, let us assume a truncated normal distribution.

We can divide the transportation network into two components, transit and non-transit, for the discussion of uncertainty. The sources of uncertainty are different for each so they are presented separately. The interaction between the two components is then discussed. For the non-transit aspect (automobile, pedestrian, bicycle), the uncertainty lies in the speed of the pathways and the delays at connectors. Therefore, for any non-transit leg the uncertainty is the cumulative effect of the uncertainty on each edge and vertex.

Define the time it takes to traverse an edge e to be $duration(e) = \frac{e_l}{min(e_\mu, m_{maximum\_speed})}$, where e_l is the length of e and e_μ is the expected speed on e and $maximum\_speed$ is the maximum allowed speed for the mode (if defined). Recall that at this time it is only the bicycle mode that is restricted by a maximum speed. Similarly, define the $duration(v)$ to be the expected value (mean) of the delay distribution for the vertex. Therefore, given a leg, $l =< v_0, (e_0, t_0), v_1, (e_1, t_1) ..., v_{n-1}, (e_{n-1}, t_n), v_n >$, the $duration(l) = \sum_{i=0}^{n-1} duration(e_i) + \sum_{i=1}^{n-1} duration(v_i)$, for $e, v \in l$. Note that the duration does not include the delay for the first or last vertex. Those vertices correspond to transfers, and I will discuss how those are handled shortly.

The transit portion of the network handles uncertainty in a fundamentally different manner: schedules. The schedules, as previously defined, are listed as runs: a time dependent

[1]A truncated normal distribution is a distribution $X \sim N(\mu, \sigma^2)$ that lies within the interval $X \in (a, b), -\infty \leq a \leq b \leq \infty$.

sequence of vertices. Given our previous definitions, a transit leg l is a subsequence of a run for some route, ρ_r. The departure time at each vertex in the schedule is a distribution, and for a given run the subsequent departure time distributions are assumed to model any uncertainty experienced. Given the leg, $l =< v_0, (e_0, t_0), v_1, (e_1, t_1)..., v_{n-1}, (e_{n-1}, t_n), v_n >$, the $duration(l) = arrival(v_n) - departure(v_0)$.

Once the duration of both types of legs is defined, one can consider the duration of the entire trip. The trip is a sequence of transfers and legs, $\tau =< \rho_0, l_0, \rho_1, l_1, ..., \rho_n >$. We have already defined the durations of the legs. The duration of a transfer is the time it takes to transition between the two legs at the connecting vertex. There are four possible types of transfers:

1. null
2. intermodal
3. intramodal
4. stop/pseudo-transfer

The null transfers are the most straightforward. They represent the start and finish of the trip where no true transfers take place, therefore the duration of a null transfer is 0.

For the intermodal and intramodal transfers, it is known that it takes at least some time to make a transfer in the real world. Therefore, define a value for the minimum transfer time for both intermodal and intramodal transfers. Call them mt_{inter} and mt_{intra} respectively. For the stop at a resource (i.e., pseudo-transfer), there is a defined duration of the stop—specified by the user—denoted as $duration(stop)$. In each of the three cases, the values specify a minimum bound for the duration of the transfer. If the target mode of the transfer is non-transit (i.e., the user is transferring to pedestrian, walking, or biking), the upper bound is the same as the lower bound.

For the cases where the target mode is transit, the user is constrained by the transit schedule. Given two legs, l_1, l_2 and a connecting transfer ρ, where l_2 is a subsequence of run on route r, κ_r, one can show the bounds on $duration(\rho)$. It must be the case that $arrival(l_1) < departure(l_2)$. It is also the case that $departure(l_2) \geq arrival(l_1) + b$ where, b is either mt_{inter} or mt_{intra} depending on the type of transfer—the departure of the second leg cannot occur less than b time after the arrival of the first leg. Therefore, choose the departure time δ to be the earliest in the schedule such that $\delta \geq arrival(l_1) + b$; making the upper bound on the duration of the transfer $\delta - (arrival(l_1) + b)$. Intuitively, this is how long the person must wait for the next available bus or train after arriving at the stop, subject to the constraint that the bus or train arrive at least some minimum time b after the person arrived. We also note that one can calculate the probability that a transfer is successful, by calculating the probability that the inequality $\delta \geq arrival(l_1) + b$ is met. In conclusion, the duration of a trip is the sum of the durations of its legs and transfers, and the probability that the trip can be completed at the estimated arrival time is the probability that all transfers are made. More details will be presented in Section 4.4.1.4.

CHAPTER 4

TRANQUYL: THE TRANSPORTATION QUERY LANGUAGE

4.1 A Relational Representation

In this section I will demonstrate how the graph model can be translated into a relational model suitable for relational databases. First, I explain the notation scheme to be used in the remainder of the dissertation.

4.1.1 Notation

In the remainder of the dissertation I 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 dissertation 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*.

4.1.2 Relations

The model I am proposing is a relational model. They are well-known and compatible with many existing database systems. We are able to leverage a significant body of research on optimizations, and practitioners are familiar with the query structure for relational languages. Like all graphs, the heart of the model is defined as a set of vertices and edges. These can be stored in tables; first the edges are defined as,

```
road_edge(id:int, geometry:line, speed_distribution_id:int,
  resources:set, name:string, width:real, time:period,
  has_toll:boolean, is_one_way:boolean,
  has_parking:boolean, time:period, num_lanes:int,
  has_shoulder:boolean)

freeway_edge(id:int, geometry:line, speed_distribution_id:int,
  resources:set, name:string, width:real, time:period,
  has_toll:boolean,num_lanes:int, has_shoulder:boolean)

bike_edge(id:int, geometry:line, speed_distribution_id:int,
  resources:set, name:string, width:real, time:period,
  is_protected:boolean, is_independent:boolean)

sidewalk_edge(id:int, geometry:line, speed_distribution_id:int,
  resources:set, name:string, width:real, time:period)

rail_edge(id:int, geometry:line, speed_distribution_id:int,
  resources:set, name:string, width:real, time:period)
```

and then the vertices for both the connectors and pseudo connectors,

```
    vertex(id:int, resource:set, geometry:point,
      delay:distribution, has_stop_sign:boolean,
      has_traffic_light:boolean, turn_restriction:set)

    pseudo-vertex(id:int, resource:set, geometry:point,
      on_edge:int, offset:real)
```

where the pseudo-vertex can be considered a subtype of the vertex – i.e., anywhere a vertex can be used, a pseudo-vertex is also acceptable. We can store the resources in a relation,

```
    resource(id:int, name:string, class:string, access_vertex_id:int,
      accessible_period:period)
```

We also need to define a turn restriction,

```
    turn-restriction(e1_id:int,e2_id:int)
```

From the vertex and edge relations one can build the route and road relations.

```
    route(id:int, name:string, path:[sequence:<v1_id:int,e1_id:int,
      v2_id:int, e2_id:int, ..., vk_id:int>])
    road(id:int, name:string, path:[sequence:<v1_id:int,e1_id:int,
      v2_id:int, e2_id:int, ..., vk_id:int>])
```

We may also wish to have a relation that simply enumerates pathway types,

```
pathway(id:int, type:string)
```

Our selection of a truncated normal distribution for modeling speeds as well as departure and arrival times allows us to model the necessary information with only four real-valued parameters. Storage overhead is potentially reduced by maintaining a table of distributions and referring to them by their id attribute where needed.

```
distribution(id:int, mean:real, variance: real, min:real, max:real)
```

Using the distribution, one can build the transit schedules as a table of departures.

```
departure(route_id:int, vertex_id:int, departure_id:int)
```

From these base relations I define our higher level relations for legs, transfers, and trips. These relations are not stored on disk prior to the issuance of a query—they are only generated at runtime.

```
leg(id:int, path:[sequence:<v1_id:int,e1_id:int, t1:instant,
  v2_id:int, e2_id:int, t2:instant, ..., vk_id:int>])
transfer(id:int, trasfer_vertex_id:int, leg1_id:int, leg2_id:int)
trip(id:int, origin_vertex_id:int, destination_vertex_id:int,
  path:[sequence:<l1_id:int, l2_id:int, ... lk_id:int>])
```

4.1.3 Spatio-Temporal Data Types

In addition to the standard numeric and string data types, both spatial and temporal types are required. Rather than develop yet another overlapping type system, I take advantage of a rich spatio-temporal type system developed by Güting, *et al.*. Their Spatio-Temporal Query Language (STQL) provides a rich data model and query language for modeling moving objects

[Erwig and Schneider, 2002b; Erwig and Schneider, 2002a; Güting et al., 2000]. They define both datatypes and operators for spatial and temporal concepts. These types include *point*, *line*, *region*, *instant*, and *period*. 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.

4.2 TRANQUYL Defined

Building on the previously defined relations, in this section I define the Transportation Query Language (TRANQUYL). The language follows an SQL-like syntax and extends the familiar SELECT, FROM, WHERE query structure. The new clauses, along with many new operators, allow elaborate trip-planning queries to be written in a concise and readable manner. To illustrate, consider to following example query:

```
SELECT *
FROM All_Trips(vertex.5236, vertex.3573) AS t
WITH MODES pedestrian, bus
WITH CERTAINTY .8
WHERE ENDS(t) <= 5:00pm
MINIMIZE LENGTH(t)
LIMIT 1;
```

This query can be read as: “from the set of all paths between the two given vertices, find those that use only pedestrian or bus modes where the trip ends no later than 5:00pm with 80% certainty, and of those, select the one with the shortest path”. In the remainder of this section I formally define TRANQUYL. I begin by providing the language grammar in BNF, and then go on to explain how the new clauses are used and introduce the new operators. Then I provide the formal semantics for a trip query. Finally, I present an extensive set of example queries to demonstrate the flexibility of the language.

4.2.1 Syntax

The syntax of TRANQUYL follows the general form of standard SQL queries, with the changes to the clausal structure discussed in Section 4.2.2. Optional statements are denoted by the square brackets, variable types are specified when necessary, and keywords are written in uppercase.

```
<query> ::= <trip query> | <sql query>

<trip query> ::=
  <select-clause>
  [<with-modes-clause>]
  [<with-certainty-clause>]
  [<with-stop-vertices-clause>]
  [<where-clause>]
  <optimize-clause>
  [<limit-clause>];

<select-clause> ::=
  SELECT *
  FROM All_Trips(<vertex>,<vertex>)
  AS <trip-variable> [, <table-references>]

<table-references> ::= null | <table-name> AS <variable>, <table-references>

<with-modes-clause> ::= WITH MODES <modes-list>
<modes-list> ::= <mode> | <modes-list>, <mode>
<mode> ::= null | BUS | RAIL | PEDESTRIAN | BICYCLE | AUTOMOBILE

<with-certainty-clause> ::= WITH CERTAINTY <real>

<with-stop-vertices-clause> ::= WITH [ORDERED] STOP VERTICES <vertex-list>
<vertex-list> ::= null | <vertex-list>, <vertex-variable>

<optimize-clause> ::= MINIMIZE <opt-expression> | MAXIMIZE <opt-expression>
<opt-expression> ::= <tranquyl operator>(<variable>) | COUNT(<rows>)
```

```
<limit-clause> ::= LIMIT <int>


<where clause> ::= <where condition>
<where condition> ::= <atomic condition> |
                      <where condition> <boolean statement> <where condition>
<atomic condition> ::= <tranquyl condition> | <sql condition>

<tranquyl condition> ::=  <resource condition> |
                          <temporal condition> |
                          <spatial condition> |
                          <certainty condition> |
                          <count condition> |
                          <attribute condition> |
                          <cost condition>

<facility condition> ::= <string> IN <vertex-variable>.resources
<temporal condition> ::= <duration condition> | <bounding condition>
<duration condition> ::= <duration operator>(<variable>) <inequality> <duration>
<bounding condition> ::= <bounding condition operator>(<variable>)
                            <inequality> <time>
<certainty condition> ::= <likelihood operator>(<temporal condition>)
                            <inequality> <number>
<spatial condition> ::= <spatial predicate>(<spatial variable>,
                            <spatial variable>)
<count condition> ::= COUNT(<rows>) <inequality> <number>
<attribute condition> ::= <variable>.<attribute> <inequality> <value>
<cost condition> ::= <cost operator>(<trip variable>) <inequality> <number>

<boolean statement> ::=  [NOT] AND | OR
<value> ::= <numeric expression> | <string>
<number> ::= <int> | <real> | <numeric expression>
<inequality> ::= = | < | <= | > | >=
```

The <int>, <real>, and <string> refer to their standard type definitions. For the temporal conditions, <duration> is a length of time (e.g., 2 hours) and a <time> is a specific time (e.g., 4:15pm). A <numeric expression> is some algebraic expression that evaluates to a single value (e.g., $(2 + 6)/3$). The <rows> refers to a set of tuples, generally returned by an operator; while a <variable> is a variable that may refer to one or more tuples of any relation. The

<spatial predicate> and <spatial type> refer to those presented in Section 4.1.3. The types are generally returned by a spatial operator. The above syntax covers the entire language with a few caveats related to <sql query> and the <sql condition>. These will be discussed in greater detail in Section 4.2.2.5 when the <where clause> is presented in detail.

4.2.2 Clauses

As shown in Section 4.2.1, TRANQUYL adds a number of clauses to the standard SELECT, FROM, WHERE query structure. Here, I elaborate on how these clauses are used.

4.2.2.1 Select Clause

The <select-clause> is used to specify the selection from tables, or more appropriately, the selection from an operator that returns a non-materialized set of tuples, the opertator being **All_Trips**. To query for a trip, the **All_Trips** operator must be used. The selection from trips between two vertices is shown in this partially defined query,

```
SELECT *
FROM ALL_TRIPS(vertex1, vertex2) AS t
...
```

It is possible to include additional tables in the select statement. For example, a user may wish to refer to a table of neighborhoods in order to include additional constraints in the <where-clause>.

```
SELECT *
FROM ALL_TRIPS(vertex1, vertex2) AS t, neighborhoods AS n
...
```

4.2.2.2 With Modes Clause

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.

4.2.2.3 With Certainty Clause

The <with-certainty-clause> specifies the minimal probability with which the where clause must be met. For example, in the following query it requires 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(vertex1, vertex2) AS t
WITH MODES pedestrian, rail
WITH CERTAINTY .9
WHERE DURATION(t) <= 35min
MINIMIZE LENGTH(t)
LIMIT 1;
```

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

4.2.2.4 With Stop Vertices Clause

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(vertex1, vertex2) AS t
WITH STOP_VERTICES v1
WITH MODES pedestrian, auto
WHERE  "movie theater" IN v1.resources
MINIMIZE LENGTH(t)
LIMIT 1;
```

This query asks for a trip with two ordered stop vertices where both have a minimal duration as follows:

```
SELECT *
FROM ALL_TRIPS(vertex1, vertex2) AS t
WITH ORDERED STOP_VERTICES v1, v2
WITH MODES pedestrian, bus, rail
WHERE  "atm" IN v1.resources
  AND DURATION(v1) > 5min
  AND "pizza place" IN v2.resources
  AND DURATION(v2) = 60min
MINIMIZE DURATION(t)
LIMIT 1;
```

4.2.2.5 Where Clause

The <where-clause> is used to specify additional constraints on the trip. These constraints may be temporal, spatial, the specification of resources to be included at vertices, a bound on the number of transfers, or any number of things. The syntax was defined, with two minor caveats. The <sql query> and <sql condition> refer to queries and sub-queries written following standard SQL syntax—not a TRANQUYL trip query. This is to allow nested sub-queries that query over components of the `trip` relation (e.g., `VERTEX, LEG`) or access an additional table reference. These queries will return a value or variable usable in an <atomic condition> It also allows queries over relations such as `trip`. Example of these queries and sub-queries are shown in Section 4.2.5.

4.2.2.6 Optimize Clause

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-clause> and <with-certainty-clause> 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
LIMIT 1;
```

If the <optimize-clause> is `MINIMIZE DURATION(t)`, it would 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 [Garey and Johnson, 1979].

4.2.2.7 Limit Clause

We leverage the `LIMIT` keyword from SQL in order to specify the number of trips returned by the query. The **All_Trips** operator retrieves all of the paths between two vertices, but generally one is only interested in the top k paths, and in many cases only the single best path is needed. If the <limit-clause> is not specified, only a single trip is returned.

4.2.3 Operators

For TRANQUYL, I define approximately two dozen new operators that are divided into six differentent categories.

4.2.3.1 Computing Trips

We begin by introducing an operator that returns all of the trips, or paths, between two vertices in the transportation network. It is defined as

All_Trips(`vertex`, `vertex`) → `TRIP`

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

4.2.3.2 Structural Operators

A `trip` is composed of subcomponents that can be accessed using the following operators:

origin(`trip`|`leg`|`edge`) → `vertex`

destination(`trip`|`leg`|`edge`) → `vertex`

intermodal_transfers(`trip`) $\rightarrow$ `TRANSFER`

intramodal_transfers(`trip`) $\rightarrow$ `TRANSFER`

all_transfers(`trip`) $\rightarrow$ `TRANSFER`

legs(`trip`) $\rightarrow$ `LEG`

edges(`trip`|`leg`) $\rightarrow$ `EDGE`

vertices(`trip`|`leg`|`edge`) $\rightarrow$ `VERTEX`

4.2.3.3 Temporal Operators

The second set of operators are to access the temporal properties of the relations. The operators return the *expected* value of the uncertain variable. Call these temporal bounding operators. We begin with operators for the starting and ending times of trips and their path-based components:

starts(`trip` | `leg` | `edge`) $\rightarrow$ *instant*

ends(`trip` | `leg` | `edge`) $\rightarrow$ *instant*

We can also look at the arrival and departure times of vertices and transfers. The values are tied closely to the starting and ending times returned by the previous operators. The departure time at a transfer corresponds to the starting time of the subsequent leg.

arrival(`vertex` | `transfer`) $\rightarrow$ *instant*

departure(`vertex` | `transfer`) $\rightarrow$ *instant*

Finally, one may wish to find the duration of the trip or any of its subcomponents, be it a leg, transfer, specific vertex, or even the time spend on specific modes or pathways.

duration(`trip` | `leg` | `edge`) → *real*

duration_of_mode(`trip`, mode) → *real*

duration_of_pathway(`trip` | `leg`, pathway_type) → *real*

duration(`transfer` | `vertex`) → *real*

4.2.3.4 Spatial Operators

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

geometry(`trip`|`leg`|`edge`) → *line*

geometry(`transfer`|`vertex`) → *point*

length(`trip`|`leg`|`edge`) → *real*

length_of_mode(`trip`, mode) → *real*

length_of_pathway(`trip` | `leg`, pathway_type) → *real*

In Section 4.1.3, I discussed that certain spatial datatypes (e.g., *point* and *line*) can be imported from STQL [Erwig and Schneider, 2002b]. We can also make use of their defined spatial predicates as the trip geometries are represented in a compatible manner. Therefore, one can use predicates such as **touches, overlaps, inside**, and **crosses** in the <where-clause> to put additional, interesting constraints on the trip path.

4.2.3.5 Certainty Operators

In addition to the <with-certainty-clause>, which provides means to place an absolute bound on the likelihood of all temporal constraints being met, I also provide an additional certainty operator to access the likelihood of specific temporal events or statements.

likelihood(inequality statement) → `real`

The inequality statement must include a temporal operator (one found in Section 4.2.3.3) as only temporal constraints are subject to uncertainty. If the operator is used to place a constraint on a temporal inequality, the constraint supersedes the required certainty defined in the <with-certainty-clause>. For example, consider the following query,

```
SELECT *
FROM ALL_TRIPS(vertex1, vertex2) AS t
WITH ORDERED STOP_VERTICES v1
WITH MODES pedestrian, bus, rail
WITH CERTAINTY .7
WHERE  "atm" IN v1.resources
  AND DURATION(v1) = 10min
  AND LIKELIHOOD(STARTS(t) > 10:00am) = 1.0
  AND ENDS(t) <= 11:30
MINIMIZE DURATION(t)
LIMIT 1;
```

There are three temporal constraints in the <where-clause> and a specified certainty constraint in the <with-certainty-clause>. If not for the use of the **likelihood** operator, all the constraints would need to be met with probability .7. In this query, the **likelihood** operator specifies that the trip must start after 10:00am with probability of 1.0; after this constraint has been met the remaining temporal constraints must be met with probability .7.

The operator may also be used to access the likelihood of a statement rather than specifying a constraint. This may be useful for performing comparisons of more than one trip. Examples of this are shown in Section 4.2.5.

4.2.3.6 Additional Transportation Operators

We also define an operator to calculate the fiscal cost of a trip, leg, or transfer.

cost(`trip`|`leg`) → *real*

Unlike length, cost is not a monotonic function of the edges in the path. It may depend on the time of day, age of the person riding, modes used, etc. We do not attempt to address these issues at this time, but rather I simply introduce the syntactic component necessary. We present a short discussion of cost processing in Section 4.4.1.3. We also provide functions to return the roads, routes, and pathways used in a trip, leg, or edge. We show how these operators can be used to place additional restrictions on the trip as well as perform analysis on trips in Section 4.2.5.

roads(`trip`|`leg`|`edge`) → `ROAD`

routes(`trip`|`leg`|`edge`) → `ROUTE`

pathways(`trip`|`leg`|`edge`) → `PATHWAY`

4.2.4 Semantics

Previously, I gave soft or intuitive definitions of the semantics. Here I define them formally 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 <with-stop-vertices-clause>. For each such trip t, define a binding ρ for <with-stop-vertices-clause> to be a function that maps X to the set of vertices on the trip t. We say that such a binding ρ is *proper* if either the <with-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 <with-stop-vertices-clause> list, it is the case that the station $\rho(x)$ is identical to, or appears before, the station $\rho(y)$ in t. From the set $\mathcal{F}$, select those trips t such that there exists a proper binding ρ and the <where-clause> condition of the query is satisfied by the pair (t, ρ) with probability greater than or equal to the value specified by the <with-certainty-clause>. From the resulting trips, further select those for which the value specified by the <optimize-clause> is optimal where expected values are used for the temporal operators.

Now, define the probability of satisfaction of the <where-clause> conditions 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-clause> conditions of query. Now 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 4.2.3.3. The satisfaction of a non-temporal condition by the pair (t, ρ) has no uncertainty in it. Thus, for each non-temporal atomic condition, 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, 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, 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 split that leg at that point into two legs. By doing this, get a trip t' so that all points in X are mapped to transfer points in t' by ρ. I define the probability of satisfaction of ϕ by the pair (t, ρ) to be the probability of satisfaction of ϕ' by (t', ρ) that is defined as follows.

Let 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 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 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 $starts(t)$ by y_1 and $ends(t)$ 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 condition ϕ'. If any of the terms from ϕ' used in generating ϕ'' are bounded by the *likelihood()* operator (Section 4.2.3.5), add an inequality to ϕ'' that is the term bounded by the probability specified by the operator inequality. 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 $i = 1, ..., n-1$, 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 $starts(trip) \geq 8 \wedge ends(trip) \leq 10$ which requires the trip to start after 8 and finish before 10; all units of time are in hours. Now there are 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, 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.

4.2.5 Example Queries

In order to make our examples more interesting and more readable, let us first define some additional tables and make some variable declarations. First, let us define three tables that represent meaningful geographic spaces,

```
park(id:int, name:string, area:region, is_free:boolean)

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

river(id:int, name:string, geometry:line)
```

Next, let us define two variables that represent the locations of a person's home and employer. We can assign the values to variables in SQL as follows.

```
LET home = (SELECT * FROM vertex WHERE id=52363);

LET work = (SELECT * FROM vertex WHERE id=36971);
```

The two variables are chosen to make the queries more succinct and easier to read.

4.2.5.1 Trip Planning Queries

Begin with a simple example of a trip query, *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, rail
MINIMIZE COUNT(INTERMODAL_TRANSFERS(t)
LIMIT 1;
```

Next, add some temporal constraints and the uncertainty bounds,

2. Find a transit-based 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, rail
WITH CERTAINTY .8
WHERE ENDS(t) <= 7:00pm
MINIMIZE DURATION_OF_MODE(t, bus)
LIMIT 1;
```

One can combine temporal constraints with the <with-stop-vertices> clause,

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 MODES pedestrian, bus, rail
WITH CERTAINTY .75
WITH ORDERED STOP VERTICES v1, v2
WHERE "pharmacy" IN v1.resources
  AND "florist" IN v2.resources
  AND DURATION(v1) >= 10min
  AND DURATION(v2) >= 10min
MINIMIZE COST(t)
LIMIT 1;
```

Next, consider a complex query with an embedded selection in which the number of street edges that have more than 4 lanes of traffic has been restricted to be zero.

4. Find the shortest path from home to work that lets me ride a bicycle on only streets with no more than 4 lanes of traffic.

```
SELECT *
FROM All_Trips(home, work) AS t
WITH MODES bicycle
WHERE
  COUNT (SELECT *
         FROM EDGES(STREETS(t)) AS e
         WHERE e.num_lanes  > 4) = 0
LIMIT 1;
```

In this query I show how another table can be used to place additional constraints on a trip. We also use the borrowed spatial operators mentioned in Section 4.2.3.4.

5. Find a trip home from work using a bicycle, that minimizes my walking, does not go through Lincoln Park, and stops at a pizza place.

```
SELECT *
FROM All_Trips AS t, park AS p
WITH MODES pedestrian, bicycle
WITH STOP VERTICES v1
WHERE "pizza_place" IN v1.resources
  AND p.name = "Lincoln Park"
  AND NOT INTERSECTS(GEOMETRY(t), p.area)
MINIMIZE DURATION_OF_MODE(t, pedestrian)
LIMIT 1;
```

One can also use embedded selections to put constraints on paths and make more interesting optimization statements.

6. Find the automobile-based trip to home that has the fewest traffic control devices but does not include freeways.

```
SELECT *
FROM All_Trips(work, home) AS t
WITH MODES automobile
WHERE (NOT ("freeway" IN (SELECT type
                          FROM PATHWAYS(t)))
MINIMIZE COUNT(SELECT *
               FROM VERTICES(t)
               WHERE has_traffic_light = true
                 OR has_stop_sign = true)
LIMIT 1;
```

The final trip query demonstrates how to disallow specific streets.

> *7. Find the shortest automobile 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 avoid both "Grand" and "Ogden" streets.*

```
SELECT *
FROM All_Trips(work, home) AT t
WITH MODES automobile
WITH CERTAINTY .8
WHERE STARTS(t) <= 5:00pm
  AND END(t) <= 7:00pm
  AND NOT ("Grand" IN (SELECT name
                       FROM STREETS(t))
  AND NOT ("Ogden" IN (SELECT name
                       FROM STREETS(t))
MINIMIZE LENGTH(t)
LIMIT 1;
```

4.2.5.2 Queries Over Trips

In this section, assume that I have saved the results of a query in a variable, X. For example,

```
LET X = (SELECT *
         FROM All_Trips(work, home) AS t
         WITH MODES pedestrian, automobile, bicycle, bus, rail
         MINIMIZE DURATION(t)
         LIMIT 1);
```

The variable X will contain a single tuple of the relation `trip`. We are able to select from the variable in order to analyze the components and attributes of the trip. The operators used in this section could be used in generating a trip as well. For example, where and when do the transfers occur?

1. Given a trip X, where are the transfers and at what times do they occur?

```
SELECT geometry(t), arrival(t)
FROM TRANSFERS(X) AS t;
```

2. Given a trip X, what was the average delay and variance caused by intersections with traffic control devices?

```
SELECT AVG(v.mean), AVG(v.variance)
FROM VERTICES(X) AS v
WHERE v.has_stop_sign = true
  OR v.has_traffic_light = true;
```

More elaborate queries might make use of more than one variable and query.

3. Given a trip X, what percent of the roads (in terms of length) had bike lanes?

```
LET hl = (SELECT SUM(length)
          FROM EDGES(X)
          WHERE has_bike_lane = true);
LET nl = (SELECT SUM(length)
          FROM EDGES(X)
          WHERE has_bike_lane = false);
SELECT hl/nl;
```

Reusing the definition of a neighborhood from Section 4.2.5,

4. Given a trip X, what neighborhoods does it pass through?

```
SELECT name
FROM neighborhood
WHERE INTERSECTS(GEOMETRY(X), area);
```

4.2.5.3 Multi-Trip Queries

Through the <limit-clause> one can query for more than one trip at a time. Additionally, multiple trips could be queried from multiple distinct queries and stored in variables.

1. *Find 5 paths from home to work using transit that arrive by 10:00am. Then, display their arrival time, duration, length, and likelihood of on time arrival; all sorted by the likelihood that the trip completes on time.* (This way the user can see the tradeoff between certainty and the other variables.)

```
LET X = (SELECT *
         FROM All_Trips(home, work) AT t
         WITH CERTAINTY .8
         WHERE ENDS(t) <= 10:00am
         LIMIT 5);

SELECT ENDS(t), DURATION(t), LENGTH(t),
  LIKELIHOOD(ENDS(t) <= 10:00am)
FROM X AS t
ORDER BY LIKELIHOOD(ENDS(t) <= 10:00am) DESC;
```

4.3 Trips and Other Models

The proposed model should not be considered entirely in a vacuum. I 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. In Section 2.2.1, I mentioned Güting's data model and query language for network constrained moving objects. We already use a number of their data types in our relation definitions (e.g., *line, point, instant, period*), and would be interested in exploring some level of compatibility between the two languages.

Despite different underlying models, both data models are based on a network representation. Their queries of interest are still primarily dealing with how trajectories interact with various spatial and temporal phenomena. They model a moving vehicle in the network as a network constrained moving point (*mgpoint*) and the entire trajectory as a line through the network (*gline*). We could define functions to export our trip, leg, edge, vertex, and transfer relations to compatible data types. For example,

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

would allow for at least a minimal translation of our concepts. This would allow for accessing the temporal aspects of the concepts. Unfortunately, such simplistic translations lose some important information, the first being the aspect of multimodality. One could extend the *mgpoint* type with information about the current mode: *mmgpoint*—a multimodal moving graph point. This would not capture the probabilistic link speeds, explicit timetables, runs, stops, transit routes, and other concepts.

4.4 Query Processing

Given the wide range of queries expressible in TRANQUYL, query processing is an important consideration. Luckily, a wide range of path algorithms for finding paths in graphs have been developed in computer science, transportation science, and operations research. Similarly, there have been decades of research on query processing in relational databases, also including graph and recursive queries.

I propose a framework for query processing by which classes of queries can be identified and an appropriate algorithm for their processing can be selected. We describe the classes as some canonical form in TRANQUYL. There are a number of additional methods that one can apply to facility the graph processing. We first present these as they apply to all queries, and then go on to describe the canonical forms and algorithms.

4.4.1 General Processing

For all query structures, there are a number of general processing steps and approaches that one can take. Each of the following techniques can be applied, independent of the path query processing technique selected.

4.4.1.1 Edge Constrains

Many of the constraints that can be specified dramatically reduce the search space for the shortest path algorithms. Any edges that are of a disallowed mode or pathway type, from a specifically removed named road or route, or are within blocked geographic regions can be removed from the search space before any additional processing takes place. Similarly, the street and freeway edges that are not relevant to the time period of interest can be removed from the search space.

4.4.1.2 Resources

When a query that requires a stop vertex with a specific resource is issued, it may be the case that one must compute multiple shortest paths—one to each vertex with such a resource. Whenever such a query is processed, all resources that have an *accessible_period* that is outside the time of interest should be removed from the set of possible resources.

4.4.1.3 Cost Processing

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. Due to the wide variability in pricing schemes, I do not present an algorithm for calculating the cost. If deployed, a specific algorithm would have to be developed for each market.

4.4.1.4 Uncertainty

Another major query processing consideration is introduced by the uncertainty clause. In this section I present a simplification to the processing mentioned in Section 4.2.4. 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 4.2.4, I gave a definition of the probability of satisfaction of the <where clause> 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 a joint probability distribution on a larger set of variables is known can be eliminated.

4.4.2 Shortest Paths Algorithms

4.4.2.1 Simple Shortest Path

The following two constraints define a class of queries that can be processed using simple shortest path algorithms.

1. <with-stop-vertices-clause> is not used

2. <optimize-clause> 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(vertex231, vertexi921) AS t
WITH MODES pedestrian, bus, rail
MINIMIZE DURATION(t);
```

The constraints are such that the graph is limited to the edges that have numeric weight values and 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.

4.4.2.2 One or More Ordered Stop Vertex

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

1. <with-stop-vertices-clause> has one or more vertices specified

2. <optimize-clause>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 s1, s2
WITH MODES auto
WHERE "wine_store" IN s1.resources
  AND "hotel" IN vs.resources
MINIMIZE DURATION(t);
```

One 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 resources), 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 stop vertices with the facility specified by the first station. For each shortest path found add a station and link to G' where the link 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 station and link to G'. After this go to step 5.

4. Consider the next station specification. Compute the shortest path from all of the last set of stop vertices to all of the stop vertices that have the facility specified by the next station. Add a corresponding station and link 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 $O(mv(e + v^2))$.

4.4.2.3 Transfer Constrained Multimodal Paths

Define another query formation,

1. <with-modes-clause> is transit and pedestrian modes
2. <with-stop-vertices-clause> is not used
3. <where-clause> has a constraint on the number of intermodal transfers
4. <optimize-clause> is the minimization of the sum of some numeric edge attribute (e.g., length, duration)

An example of such a query is,

```
SELECT *
FROM ALL_TRIPS(work, home) AS t
WITH MODES pedestrian, bus, rail
WHERE COUNT(INTERMODAL\_TRANSFERS(t) < 4
MINIMIZE DURATION(t);
```

This is a simple shortest path with the additional constraint of a maximum number of intermodal transfers. Here one can apply the algorithm proposed by Lozano and Storchi [Lozano and Storchi, 2001] for computing shortest paths in multimodal transportation networks. The label-correcting algorithm calculates a set of Pareto-optimal hyperpaths between an origin and destination and allows the specification of an upper limit on the number of intermodal transfers.

4.4.2.4 Temporally Constrained Multimodal With Transit

More complicated than the previous algorithm, one can allow for temporal constraints as well as constraints on the number of transfers in the <where-clause>:

1. <with-modes-clause> any and all modes are allowed
2. <with-stop-vertices-clause> may have a vertex
3. <where-clause> has one or more temporal constraints and possible constraints on the number of transfers.
4. <optimize-clause>is the minimization of the walking or waiting times

For this class of queries, one can use an algorithm for itinerary planning in multimodal networks [Zografos and Androutsopoulos, 2008]. Unlike the previous algorithm, this allows for multiple temporal constraints as well as the constraint on the number of intermodal transfers. It also allows automobile use in addition to transit modalities. It is designed to minimize the duration of walking or waiting (at transfers). It could be modified for other optimization criteria as well. This is done through lexicographical optimization of en-route time, total walking time, total waiting time, and number of transfers (in any order).

4.5 Summary

I have presented a detailed model that captures the spatio-temporal aspects of a modern, urban multimodal transportation network. I showed how the model translates to a relational representation and introduced a powerful query language built on that relation. I incorporated a number of elements from proven models in order to facilitate compatibility with other system while focusing on a unique problem and perspective in the database community.

CHAPTER 5

TRANSLATING NATURAL LANGUAGE TO TRANQUYL

In this chapter I present the NL2TRANQUYL system. In Section 5.1, I begin with a discussion of the issues pertaining to the natural language I wish to process, i.e., what type of queries will I allow the user to ask, or the subset of English understood. Next, I introduce the TRANQUYL ontology which I use in the translation between English and TRANQUYL in Section 5.2. Section 5.3 contains the details of the translation process, beginning with an overview of the translation steps. The chapter concludes with a short discussion in Section 5.4.

5.1 Language Considerations

Before discussing the software implementation, it is important to understand the scope of natural language I am working with as well as the reason for that selection. Clearly it would be ideal to allow users to ask any possible question even tangentially related to transportation, but such a broadly functional system is still beyond the state of the art for any natural language processing application. Instead, I chose to focus on a single task area that is in high demand: trip planning.

A few corpora pertaining to specific travel needs exist, such as: flight reservations (the ATIS corpus, one of the first available spoken dialogue corpora, and the COMMUNICATOR corpus, which can be considered as the successor to ATIS); finding information about specific points of interest in a city, such as landmarks [Gruenstein and Seneff, 2007; Gruenstein, 2008], or restaurants, including how to reach them [Bangalore and Johnston, 2009]. The Let's Go [Raux et al., 2005; Black et al., 2011] corpus is a collection of user interactions with a telephone-based information system for the Pittsburgh area bus system that answers questions about the current status of busses and bus routes. As I discussed earlier in Section 2.3, neither ATIS nor any of the other corpora include trip planning queries of the sort I am interested in, comprising the complex constraints I have discussed earlier. Hence, I set out to collect a new corpus of urban trip planning queries.

In order to guide my focus I solicited queries from two sources: the first being a group of graduate students who have interests both in computer science and transportation, the second being friends and colleagues not affiliated with the program. The responses ranged from nearly trivial (e.g., *get me home*) to complex planning situations that required getting multiple users to a single destination—while constraining the arrangement to an entire paragraph of additional details. Half were queries for trip plans with a few different constraints, and I decided to focus on those. The language used ranged from short fragments (e.g., *train home by 7:00pm*) to fully grammatical sentences (e.g., *What is the fastest transit route from my house to Lakeview at 5:00pm?*).

In addition to the language that was of interest to the users, the biggest constraint was the expressibility of TRANQUYL. The language was designed for trip planning, therefore queries beyond that scope were not of interest for this work—all of the queries handled are either explicitly or implicitly asking for a trip plan. There are some queries that ask for different types of information, for example, *find the nearest pharmacy* is looking for a location, but the software I developed is able to interpret it in such a way to answer the question anyway. In that case, a trip to that location implicitly answers the question of where the place is. In the evaluation, I show that these implicit queries can be processed.

Beyond the fact that all queries must be looking for trips, it is important to understand what vocabulary can be used and what types of trip queries can be expressed. The expressive power of TRANQUYL is greater than what NL2TRANQUYL can process. Many of the details on the understood language are presented in the following sections, but some can be introduced here. The vocabulary understood corresponds to the concepts in the ontology, which is explained in Section 5.2. A more complete listing of concepts is found in Chapter 8. Some of the concepts in the ontology are purely for organization purposes and not directly usable in a query. Some concepts, such as times (e.g., “4:30pm”) and addresses are treated as instances of the concepts and are found via regular expressions. NL2TRANQUYL has no set grammar, but there is an implicit grammatical assumption. The software relies on the Stanford Parser [Klein and Manning, 2003], which in turn uses a probabilistic context-free grammar of English. Ideally, the input to NL2TRANQUYL would be of the same grammatical form as understood by the Stanford Parser, but there is no enforcement mechanism in place. Some of the workable sentence formations can be seen in the corpus of test queries found in Chapter 9.

5.2 TRANQUYL Ontology

Since on one side there is a database schema that the output query must follow and on the other there is a naive user that wishes to issue a query against said database without understanding the underlying schema, I use an ontology to bridge the representational gap between the user and the database.

The ontology models the concepts found in TRANQUYL and the user, along with the interactions between them. For the transportation aspects, this includes concepts such as a trip, places, regions, time, metrics (e.g., cost, distance) and modes and how they are interrelated. The user aspects define personal preferences such as the most common mode, preferred metrics, and profile facts like their current location, home and work addresses. The more knowledge that the user profile contains, the less information the user must explicitly state for any given query. A small subset of the roughly 90 concept hierarchy is shown in Figure 4, while the entire list of classes are enumerated in Chapter 8.

I chose to develop an ontology after looking at existing ontologies available for use and not finding one that fit my specific needs. The reasons for this were discussed in Section 2.6. In addition to modeling the transportation network and the TRANQUYL query language, the ontology is used as a method to increase the vocabulary understood by the system. The pertinent concepts (i.e., concepts that users will express rather than those used for organizational purposes) have been manually annotated with additional lexical information. For example, the concept *train* is labeled “train” in the ontology. Locally, the urban trains are also known as “The L” and therefore a colloquial label denoting such is included. The concept is annotated with the relevant WordNet entry (with root word and sense key): *train%1:06:00*. The relatively small size of the ontology meant that manual annotation was feasible and likely faster than implementing an automatic method. How this information is used will be discussed in the next

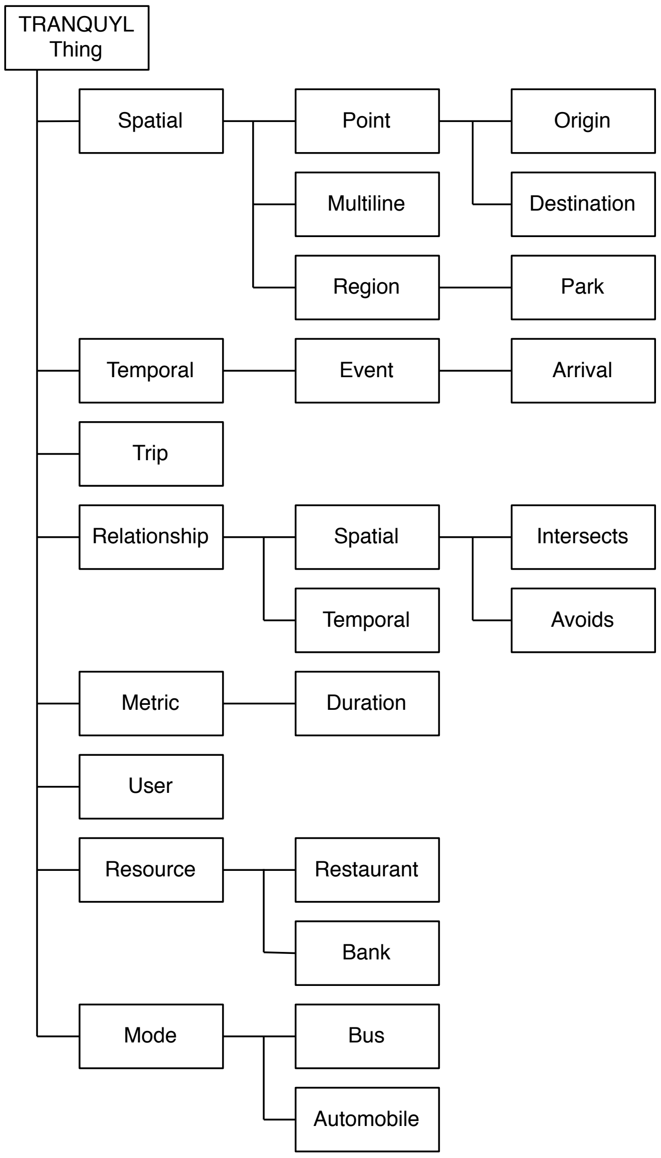


Figure 4. Subset of the TRANQUYL class hierarchy

subsection. Finally, the *place* concepts are annotated with their Google Places [Google Places API, 2011] class label. This is done to support a small web-based demonstration found in 6.3.

At this time no reasoner is used in conjunction with the ontology. It is only being used as a knowledge model to guide the translation and advanced lexicon to improve the scope of language understood. There may be ways to exploit the more advanced aspects of ontologies and the semantic web, but at this time it is beyond the scope of the research.

5.3 Translation Pipeline

The translation from natural language to TRANQUYL occurs in several distinct steps, as shown visually in Figure 5. The four steps are called 'filters' as they attempt to remove noise or the lack of information. The may mean taking information present in the previous step and adding additional, useful information to it or removing unnecessary information. The four stages correspond to parsing, concept identification, concept attachment, and query generation. The details of the stages, along with a running example, are covered in the following two subsections.

5.3.1 Parsing and Concept Identification

The first phase of the translation from NL to TRANQUYL involves parsing the input and identifying the key concepts. It begins by parsing the input with the Stanford Parser [Klein and

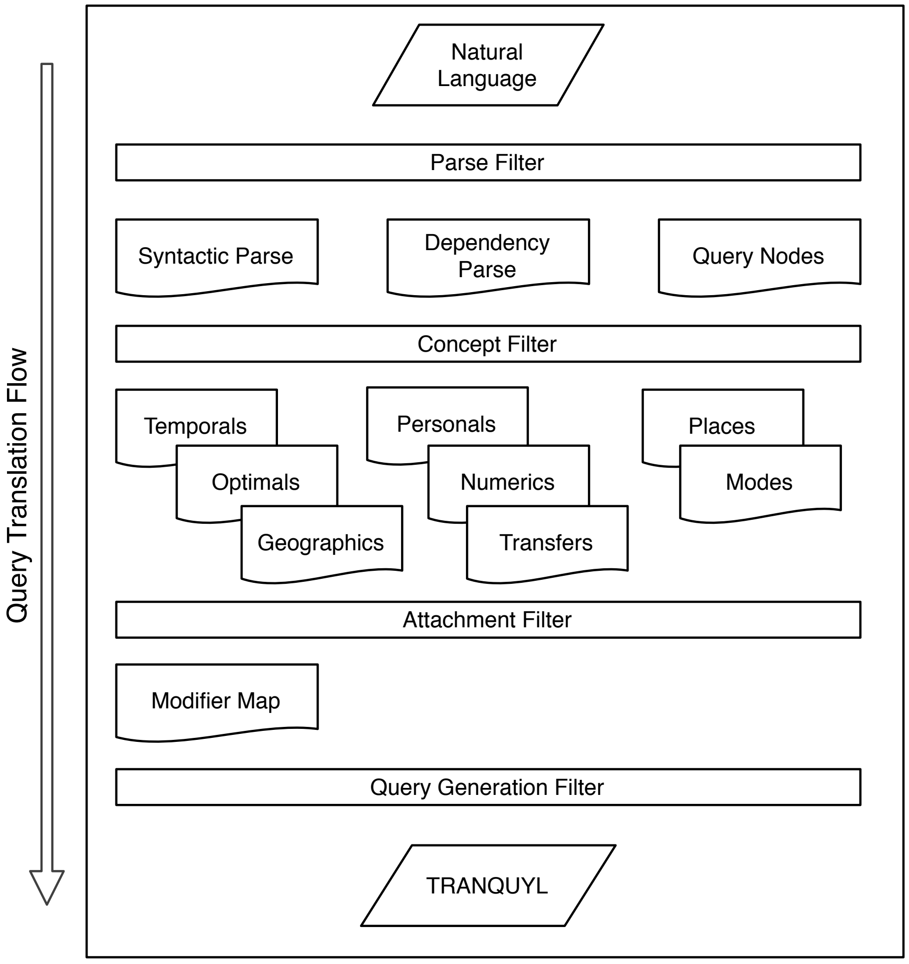


Figure 5. Overview of the NL to TRANQUYL translation pipeline

Manning, 2003] in order to obtain both a constituency[1] and dependency parse[2]. For illustrative purposes, example parses for the query *Can I walk to 400 W. Washington Ave. by 4:00pm?*[3] are presented in Figure 6 and Figure 7.

From the set of nodes found in the constituency parse tree, the relevant ones are selected and form a set N. The relevant nodes are adjectives, adverbs, verbs, nouns, prepositions, and simple phrases—concepts that may exist in the ontology. In order to determine which concept in the ontology a specific node corresponds to, two approaches are taken: 1. comparing nodes to concepts in the ontology, and 2. identifying specific concepts via regular expressions. The algorithm, in pseudocode, is found in Algorithm 1.

First a pairwise matching between the node and all of the concepts in the ontology is performed using a set of metrics including a direct string match, match within 1 edit, stemmed match, and matches via hyponym and hypernym in the WordNet lexical tree. Each match type

[1]Constituency (a.k.a., phrase structure or structural) parses break a sentence into its constituent components. The result is a tree of more and more granular concepts–moving from phrases (e.g., noun phrases, prepositional phrases, verb phrases) to smaller constituent components (e.g., adjectival nouns, compound nouns) until single base types are reached (e.g., noun, verb, adjective). The representation only captures syntactic relationships related to how the sentence is organized.

[2]A dependency parse is another form of syntactic parse. The parse output takes the form of a graph where words are nodes and the relationships, or dependencies, between them are modeled as labeled edges (e.g., adjectival modifier, noun modifier, cardinal number). This parse form may be more appropriate for extracting relationships between words.

[3]While this sentence may constitute poor phrasing with respect to telicity, this construction was found to be used by native English speakers in multiple instances. The sentence is implicitly asking for a path to 400 W. Washing Ave., such that they can walk there and arrive by 4:00pm. The understanding is that given the current time, is there enough time before 4:00pm to complete the trip? The inelegant language underscores the ability of NL2TRANQUYL to understand imperfect input.

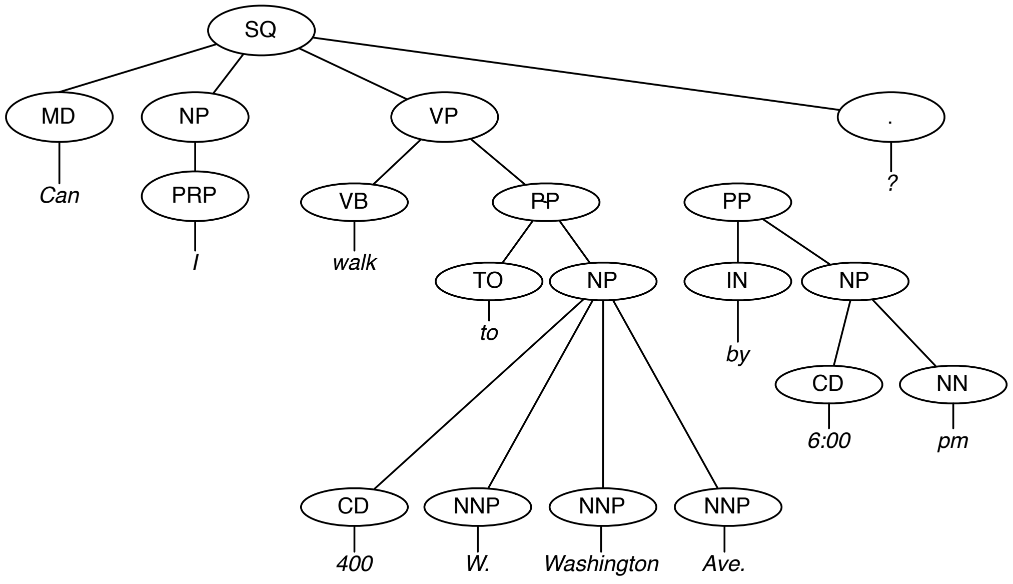


Figure 6. Example constituency parse tree

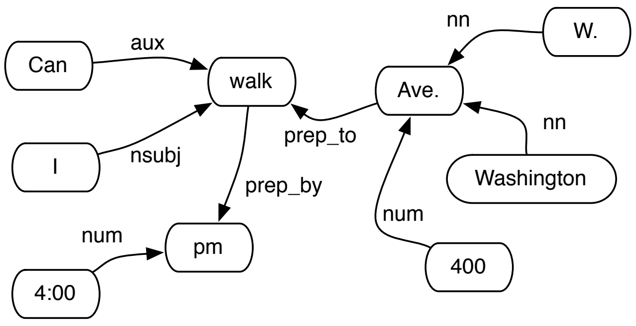


Figure 7. Example dependency parse graph

Algorithm 1 Parsing and Concept Identification

```
**input**: Ontology $ont$, String $query$, Metrics $M$, RegularExpressions $R$
$C \leftarrow$ getAllConcepts($ont$)
$p \leftarrow$ constituencyParse($query$)
$d \leftarrow$ dependencyParse($query$)
$N \leftarrow$ nodesFromParse($p$)
$N \leftarrow$ prune($N$)
Mapping $M$ = new Mapping()
$M$.addMappings(getOntologyConcepts($N, C, M$))
$M$.addMappings(getRegExConcepts($N, C, R$))
```

is associated with a score, and scores for each concept are commutative (i.e., the scores for each match type are added to the concept's total score). The concept with the highest score, subject to a minimum threshold, is selected as the concept representing the node. The match types and their corresponding weights are found in Table VI and Table VII. A more detailed pseudocode description of the algorithm is found in Algorithm 2.

Secondly, in addition to the ontology concepts, several other concepts that are not explicitly within the ontology are extracted, namely: specific instances of *times, addresses*, and other *numeric values* such as costs (e.g., \$3.50) and certainty specification (e.g., 50%). These concepts are identified using regular expressions, and while the specific instance of a time (e.g., 4:00pm) is not in the ontology, the results can be associated with the appropriate concept in the ontology (e.g., \$3.50 is mapped to the concept *cost*). This is shown in Algorithm 3.

The ontology concept match criteria were selected based on the information contained in the model. Within the ontology there is only the concept/instance label, colloquial annotation, and WordNet annotation. It was imperative that the most information possible be drawn from

Algorithm 2 getOntologyConcepts

```
input: Nodes $N$, Concepts $C$, Metrics $M$
Mapping $map$ = new Mapping()
for all $n \in N$ do
   for all $c \in C$ do
      for all $m \in M$ do
         $V_c \leftarrow V_c + \text{matchScore}(m, c, n)$
      end for
   end for
   $w \leftarrow \text{argmax}_c V_c$
   if $V_c > threshold$ then
      $map$.addMapping($n \rightarrow w$)
   end if
end for
return $map$
```

Algorithm 3 getRegExConcepts

```
input: Nodes $N$, Concepts $C$, RegularExpressions $R$
Mapping $map$ = new Mapping()
for all $n \in N$ do
   for all $r \in R$ do
      if regExMatches($n, r$) then
         $x \leftarrow \text{getConceptForExpression}(r)$
         $map$.addMapping($n \rightarrow x$)
      end if
   end for
end for
```

these labels. For the concepts found in the input, there is the surface form (i.e., how the concept is written) and it is possible to lookup the relevant WordNet entry. Due to there being multiple senses for some words (e.g., a bank that holds money vs. a bank of a river) the found sense may not be correct. At minimum, knowing the part of speech of the concept allows a slightly better entry lookup.

Previous works, for example Budanitsky and Hirst [Budanitsky and Hirst, 2006], have used matching criteria such as edit distance and distance in the WordNet hierarchy to match concepts. The match criteria weights and minimum threshold were determined empirically. A small corpus of 12 sentences was used. Within those 12 sentences, there were 43 concept nodes. The weights and thresholds were tuned in order to maximize the precision and recall for those concepts. This was done through trial and error and was not exhaustive. My approach was originally published in [Booth et al., 2009]. The minimum threshold value was determined to be .24.

Continuing with the example query of *Can I walk to 400 W. Washington by 4:00pm?*. The system will identify *pedestrian,* and *before* as concepts and *400 W. Washington* and *4:00pm* as instances of concepts in the ontology. Note that the concept labels do not exactly match their representation in the query sentence.

5.3.2 Knowledge Map

After the concepts in the query have been identified, it is important to determine how they are related. The dependency parse and a number of heuristics, guided by the ontology, are used to generate a knowledge map. This map is effectively a subgraph of the ontology. The

TABLE VI

WEIGHTS FOR STRING MATCHES

Concept Target String	**Distance Measure**	**Weight**
Plain Text$_{ConceptLabel}$	Equal	1
Plain Text$_{ColloquialLabel}$	Equal	1
Plain Text$_{InstanceLabel}$	Equal	1
Plain Text$_{ConceptLabel}$	Stemmed Equal	.25
Plain Text$_{ColloquialLabel}$	Stemmed Equal	.25
Plain Text$_{InstanceLabel}$	Stemmed Equal	.25
Plain Text$_{ConceptLabel}$	Edit Distance: 1	.15
Plain Text$_{ColloquialLabel}$	Edit Distance: 1	.15
Plain Text$_{InstanceLabel}$	Edit Distance: 1	.15

TABLE VII

WEIGHTS FOR WORDNET MATCHES

Node Source	**Concept Target**	**Weight**
Synonym$_{Lemma+POS}$	WNI	.5
Hypernym$_{Lemma+POS}$	WNI	.1
Hyponym$_{Lemma+POS}$	WNI	.1
Synonym$_{Lemma}$	WNI	.25
Hypernym$_{Lemma}$	WNI	.05
Hyponym$_{Lemma}$	WNI	.05

previous step gave the subset of concepts in the ontology that are of concern, and in this step it is determined how they are interrelated. Before describing the procedure, I present a example knowledge map in order to illustrate the objective. Once again I use the query *Can I walk to 400 W. Washington Ave. by 4:00pm?*.

Each query contains two key concepts, *user* and *trip*, that are central to the problem of trip planning. Branching off from these two concepts are other supporting concepts within the ontology. The only concepts explicitly mentioned in the input query are the four instances (rounded rectangles) in the graph. The explicit relationships in the ontology tell us how the instances may be related. Certain relationships may not be determinable with perfect reliability. In the remainder of this section I briefly discuss how the knowledge map is built and in the following section I discuss how it is used to generate the final TRANQUYL query.

In generating the knowledge map, the first sources of information are the query parses. Of course, well formatted input increases the likelihood that the parses are accurate. Though, even with well formatted input the constituency parse can vary dramatically depending on the phrasing of the query. In my experience, the dependency parse is much more stable across sentence formulations. When possible, the dependency parse is used exclusively, but if it is incomplete it is possible to fall back to the constituency parse or plaintext input and heuristics.

Ideally, all of the identified concepts fit nicely into the ontology in unambiguous ways—in that case the map is built by simply following the predefined relationships. In general, there are three primary tasks in building the map: identifying personal references, aligning modifiers, and determining which data are missing.

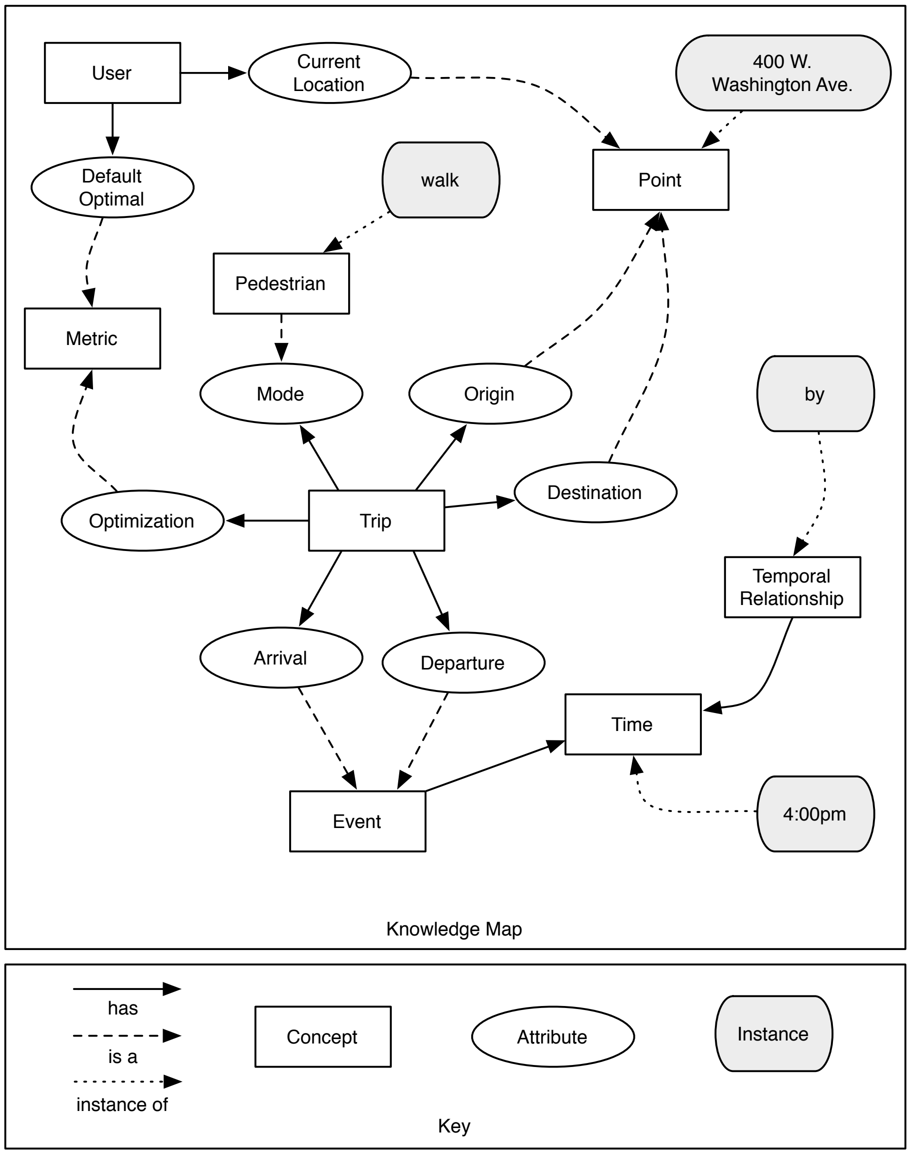


Figure 8. Example knowledge map

For the possessives, the most direct manner of identification relies on the dependency parse. The parse will return a dependency of type *possessive* that points to the node in question. The concept which corresponds to the node will have been identified by the means discussed in the previous section. For queries where an explicit statement of possession has not been made, the ontology can be consulted and simple heuristics based on linguistic theories can be used. Based on knowledge of referring expressions [Webber, 1978; Hirst, 1981], if a user says, *the bank* it is likely referring to a specific one; whereas *a bank* or *any bank* could be satisfied by any bank. If a specific, even if unnamed, resource is specified the user's profile is checked to see whether they have a preferred instance of that resource. If so, replace the generic concept with the specific instance. The same happens for the explicit possessive case, though a generic will be substituted if no instance is found (e.g., a user did not complete their profile).

Next, the correct modifier alignment is determined. Which concepts are modified by which concepts must be understood. Generally adverbs, adjectives, and prepositions are modifiers. Some concepts (e.g., *cost*) have the potential to modify more than one concept (e.g., a cheap restaurant vs. a cheap trip). If available, use the modifier relations returned by the dependency parser; if not, expand a window around the nodes representing the compatible concepts in the sentence and select the node that is the closest.

Finally, look for what is not known—constraints and values that are missing from the query. Because the query is always for a trip, simply check the ontology and determine which of the required concepts associated with a trip have not been found yet (e.g., origin, destination, mode choice). Similarly, if concepts that are not associated with any other have been identified, an anchor for the concept can be found within the ontology. The missing components are highlighted in the knowledge map, which is then passed onto the query generation step. The high level pseudo-code for the knowledge map generation is found in Algorithm 4.

5.3.3 Query Generation

The final step is generating the TRANQUYL query using the knowledge map. It begins with the generic TRANQUYL query structure (the standard SQL format of SELECT, FROM, WHERE that has been extended to include 4 new clauses) and fill in the appropriate values. To continue with the illustrative example, the TRANQUYL query for the running example, *Can I walk to 400 W. Washington Ave. by 4:00pm?* is as follows,

```
SELECT * FROM
  ALL_TRIPS(user.current_location, 400 W. Washington Ave.) AS t
  WITH MODES pedestrian
  WHERE ENDS(t) <= 4:00pm
  MINIMIZE DURATION(t)
  LIMIT 1;
```

In this case, the <select clause> contains a call to the ALL_TRIPS operator which returns a non-materialized set of tuples of all of the trips between the user's current location and 400 W. Washington Ave. The <with modes clause> specifies that the trip may only use the pedestrian mode. Furthermore, the trip must be completed by 4:00pm and the software made the decision to use the user's default optimization metric of duration as none was found in the input query. The rules for generating each of these clauses, as well as the ones not used in this example, are as follows:

Algorithm 4 Knowledge Map Generation

```
input: Ontology $ont$, ConceptMap $map$, DependencyParse $dep$, String $query$
for all Concept $c \in map$ do
  if $dep$.has($c$,“poss”) ∥ $dep$.has($c$,“det”) ∥ $c$.hasParentConcept(“personal”) then
    $c.isPersonal = true$
  else
    $c.isPersonal = false$
  end if
end for
ModifierMap $mods$ = new ModifierMap()
String[] $modTypes \leftarrow \{‘amod', ‘advmod', ‘nn', ‘infmod', ‘num'\}$
for all Concept $c \in map$ do
  for all String $m \in modTypes$ do
    if $dep$.has($c, m$) & $ont$.existsPath($c$,conceptFor($dep$.targetOf($c, m$)) then
      $mods$.makeConnection($c$, conceptFor($dep$.targetOf($c, m$)))
      $c.hasPath = true$
    end if
  end for
end for
for all Concept $c \in map$ do
  if $c.hasPath \neq true$ then
    for int $i$ = 1...4 do
      $windowConcepts$ ←conceptsXWordsFrom($i$,$c$,$query$)
      for all $Conceptw \in windowConcepts$ do
        if $ont$.existsPath($c$,$w$) then
          $mods$.makeConnection($c$,$w$)
        end if
      end for
    end for
  end if
end for
Concept[] $unknowns \leftarrow \{\}$
for all Concept $c \in ont$.getPropertyConceptsForConcept(‘trip’) do
  if $c.hasPath \neq true$ then
    $unkowns$.add($c$)
  end if
end for
```

<select clause> Fill in the origin and destination in the All_Trips operator using the values found in the knowledge map. If only one place has been found, treat it as the destination and use the user's current location as the origin. If the destination has been determined to be another resource, substitute the appropriate variable.

<with modes clause> List the modes found in the query, expanding *transit* to be the appropriate modes it includes. If no modes are listed, use the preference in the user profile.

<with stop vertices clause> If an intermediate node is required (e.g., in the case of visiting a resource), add the variable and pass the variable name onto the <where clause> generator.

<with certainty clause> If there is one or more temporal constraint, include the clause and specify the required level of certainty. The value can be found in the query or the user's profile.

<where clause> Include all of the spatial, temporal, transfer, cost, resource, and other constraints that have been specified and are included in the knowledge map. If a resource is a personal reference, use the appropriate value from the user's profile.

<optimize clause> If a trip optimization metric is found in the knowledge map generate the appropriate statement, else use the default from the user profile.

<limit clause> The clause is set to be `LIMIT 1` by default and cannot be changed.

In all cases, the system looks for default or preferential values in the user profile if some information is missing from the knowledge map. If the information is not specified anywhere, the system can be set to use a system-wide default setting or drop the statement entirely.

5.4 Discussion

In Section 6.2 I will demonstrate that the method described above is effective at generating trip planning queries in TRANQUYL; however, before doing so I would like to highlight some of the key aspects of the system that are difficult to capture in a formal evaluation and may not be immediately obvious from the algorithm.

The system is robust to sentence structure, ungrammaticality, and fragmentation. As an example, the following four input queries will produce the same, correct output.

- Find the fastest way to my home, that uses transit, and stops at the bank by 7:00pm.
- Arriving at home by 7:00pm, also visit a bank using transit.
- Getting there by 7:00pm to home stopping at bank with transit.
- Fastest transit to home with bank by 7:00pm

Being robust in understanding the same query in a wide range of formulations means that users do not need to always form queries in the exact same manner following a prescribed format. This is important if users are distracted or multitasking, unsure about using a new system, using spoken language, or unwilling to spend significant time training. I demonstrate how robust the system is by taking the corpus of grammatical queries and systematically breaking each into two different ways and then comparing the results to those for the grammatical queries.

In Section 5.1, I briefly touch on the idea that users are interested in more than just straight trip planning queries. Many of the resource location queries (e.g., find the bank) and binary response (e.g., is there a bus?) can be formulated as implicit trips. Consider the following queries,

- Find the nearest grocery store.
- Is there a train to 3200 N. Halsted St.?
- Can I walk to 400 W. Washington Ave. by 4:00pm?

Each can be represented as a query looking for a trip that meets those requirements, and NL2TRANQUYL will successfully interpret them as such.

I have shown examples of the temporal and resource constraints on trips in a number of places, but there are a few more to mention in detail. The first are constraints (inclusive and exclusive) on geographic regions. The user can issue queries such as *Find a way to walk home that includes a park* where a park is some type spatial extent in the database.

While it is unlikely to be used much in practice, I include the ability to include constraints on the probability of a trip fulfilling the temporal constraints. As mentioned previously, one of the key features of TRANQUYL is that it models the underlying uncertainty in the transportation network. It is possible to explicitly state the certainty requirements in natural language using the system (e.g., *Find a way home by bus that arrives at 6:00pm with 99% likelihood*). In the future I will explore the use of non-numeric likelihood values (e.g., very likely, absolutely) as they are more intuitive to users and could be customized. The user is also able to place constraints on the number of transfers, monetary cost of the trip, and use additional optimization metrics (e.g., reliability).

5.5 Summary and Possible Extensions

The discussion shows key contribution for the natural language interface. In the evaluation (Section 6.2) I will show that the system is highly accurate and provides robust handling of trip planning queries expressed in natural language. Not only does the system perform well on well-formed queries, but it also succeeds on poorly and partially formed input sentences. This provides a solid foundation for a real-world system. This is feasible as I begin by identifying the smallest identifiable semantic units in the input and building up the knowledge from there with the ontology providing guidance of what to look for—I do not start with the assumption that the input is good or in any singularly coherent format.

There are a number of directions I am pursuing for further work on this aspect of the dissertation. The first is continuing with collaborators in the development of a fully functioning ITA where the NLI can provide an interface option. Once the system is situated in a user accessible environment, I can further evaluate the efficacy of the NLI as it exists now. Additionally, I would like to explore allowing a user to interact with the system (e.g., clarifying ambiguities, providing information) as it has been shown to improve accuracy [Misu and Kawahara, 2006]. How this interaction would take place will depend on the modality of the returned results. It may also be easier for users to provide the queries in multiple steps (i.e., sentences), which is currently not supported.

In summary, I have presented a robust natural language interface for trip planning in multimodal urban transportation networks—a previously unstudied domain that presents a unique combination of spatial, temporal, and path concepts that can be tied into a user-centric model that allows for a variety of self-referential language.

CHAPTER 6

EVALUATION

In this chapter I present a form of evaluation for each of TRANQUYL and NL2TRANQUYL.

6.1 TRANQUYL

Evaluating query languages is a difficult task as there are no agreed upon standards and such evaluations are necessarily 'soft' in many cases. It is unlike comparing different implementations of a database system where there are metrics and benchmarks by which performance can be judged. In evaluating the language itself, questions such as *how easy is it to use?* and *how expressive is it?* are most appropriate. For the first question, I can only offer anecdotal evidence that language is easy to use. A student was given minimal training in the language and was able to consistently write correct queries. This is discussed more in the evaluation of NL2TRANQUYL (Section 6.2). For the second question, I can show how TRANQUYL is more expressive than the closest model available.

6.1.1 Expressive Capabilities

TRANQUYL has been shown to be a very expressive language, capable of capturing a wide range of queries. In Section 2.2 I discussed how our language differs from existing moving objects and spatio-temporal languages—some of which offered methods of computing a simple shortest path between two vertices. TRANQUYL, at its core, is a graph query language and therefore it is important that I compare it to existing languages in greater detail.

Of these works, Dar and Agrawal's proposed an extension to SQL named SQL/TC [Dar and Agrawal, 1993] is the most similar to TRANQUYL. They present a relational algebra and query language for generalized transitive closure. Their language captures more, but not all of the expressive capabilities of TRANQUYL. They define a graph query, Q, as

$$Q \equiv_{\pi_A,\delta} AGG_{Y,Z,\sigma} CON_{X,\xi} PATHS_{\lambda}(R[S,T,L])$$

where R is a table with a source and destination (S and T) with optional attribute L. For TRANQUYL, there are equivalent tables for vertices, edges and their associated attributes. The $PATHS$ operator is equivalent to **All_Trips** and the optional λ predicates can be expressed in the <where-clause>. The label concatenation operator (CON) obtains the values of the labels in R and the ξ and σ predicates allow for arc and path selection. In TRANQUYL, the functionality is covered through the use of operators, the <where-clause>, and the <with-modes-clause>. Finally, the label aggregation operator AGG and its path selection predicate δ is handled by the <optmize-clause>; the projection, π, is handled implicitly.

There are no explicit means to handle spatial or temporal variables, nor ways to quantify or capture uncertainty, in SQL/TC. Neither can it express queries for paths that include specific intermediate vertices. I also feel that, because TRANQUYL was developed specifically for transportation applications, the language syntax is more intuitive and succinct in expressing equivalent queries when possible.

6.2 NL2TRANQUYL

In this section I present the evaluation of NL2TRANQUYL. First, I introduce the procedure used, secondly I present an evaluation of system performance on well formatted and grammatical input. Then, I show how the system performs on paraphrases of the queries, some of which are ungrammatical or awkward, as well as telegraphic or fragmented queries. In all cases the system performs with high accuracy—that is, the input queries are correctly translated into TRANQUYL. Finally, I briefly discuss the success rate of processing the implicit queries mentioned in Section 5.1. I then present a discussion of an informal evaluation to identify shortcomings in the language scope and to direct future work.

6.2.1 Evaluation Procedure

As noted in [Jurafsky and Martin, 2009], the best way to evaluate the performance of a NL system is to embed it in an application and to measure the end-to-end performance of the application. In such extrinsic evaluations, various measures of dialogue quality, of performance of the users on tasks of interest, and of user satisfaction are collected, and possibly combined via frameworks such as PARADISE [Walker et al., 1997] However, such extrinsic evaluations are often not possible, as in the work here, since the ITA in which NL2TRANQUYL is to be embedded is not available yet; neither is an appropriate database. Hence, I conducted an intrinsic evaluation, i.e., an evaluation independent of the application, in which I evaluated the quality of the translations of NL queries into TRANQUYL.

Even for an intrinsic evaluation, I faced an obstacle, namely, the lack of any standard corpus, available for trip planning of the sort I am addressing in this work. I discussed the lack of such corpus earlier in Section 5.1, which forced me to collect a small corpus of queries. However, since those queries had been used for development, they clearly could not be used for testing.

It became then clear that I needed to generate a second corpus of queries, with their desired translations, in order to be able to compare the system's outputs to the desired outputs. The desired outputs would need to be generated with sufficient accuracy to provide a gold standard for system evaluation; to do so, human intervention was required. Hence, the evaluation follows the paradigm of assigning to humans the same task that the system has to perform, and evaluating the differences (if any) between the outputs produced by the software and the human(s). Whereas the ultimate incarnation of this type of evaluation would be the Turing test, it has been used on a much smaller scale on a variety of language processing tasks, from information retrieval [Will, 1993], to Natural Language Generation [Lester and Porter, 1997] to Machine Translation [Vilar et al., 2006]

In order to perform the evaluation, I pitted the NL2TRANQUYL software against two humans: an expert and a novice. The test was to see if the software would produce semantically equivalent TRANQUYL queries as the humans. The novice user was a Ph.D. student in the department who had no previous knowledge of the details of the TRANQUYL language. He was given a description of the language that included the BNF, a list of the operators and parameters, and ten example queries. His task was then to translate the test queries from English into TRANQUYL. I selected a person unfamiliar with the software and language in order to remove any potential bias from prior knowledge.

I translated the queries into TRANQUYL in order to provide a true gold standard in addition to the unbiased results of the novice. Furthermore, I can determine how well a novice compares to the software. Even if the software produces errors, it may not be significantly worse than the errors produced by a highly competent database query writer—let alone a truly naive non-technical user.

6.2.2 Well-Formed Queries

In order to perform the intrinsic evaluation, I generated a corpus of 162 total queries. I began with 50 queries that were generated by 3 students. Each of these well-formed queries were paraphrased into two different forms: general paraphrase and fragmented paraphrase (see Section 6.2.3). An additional 12 queries were generated to demonstrate the system's effectiveness on implicit queries (see Section 6.2.4). All of the test queries can be found in Chapter 9.

For the first 50 queries, I solicited the input from 3 students related to the transportation research project. I provided a list of queries that demonstrated the functionality of the system as well as a background on roughly how the system worked. They were instructed to generate new queries that included the different types of constraints and to order them as they wished – adding and removing clauses freely without worrying about a specific structure. The queries were checked for spelling and content to the point that the queries should be answerable—that is, they were within the scope and vocabulary of the system. The three users were not allowed to write random queries outside of the scope of the system; this evaluation was designed to

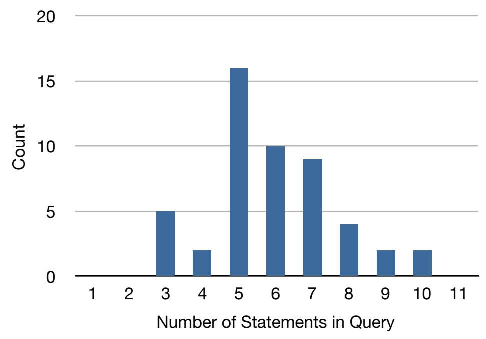


Figure 9. Query length distribution

test the system on what it should be able to process. They could fail by way of grammar, complexity, or the use of synonyms.

In the well-formed test set there were a total of 50 queries that ranged in complexity from 1 to 10 necessary statements. A statement translates to a line in the output query that contains generated values—line breaks for easier reading are not counted, nor is `SELECT * FROM` considered a separate statement if it appears on a separate line. Consider a statement to be a unit of information independent of its representation in the natural language input. That unit of information may be expressed as anything from a single word to an entire clause, which is why I judge the complexity based on the necessary output in a formal language.

The distribution of query complexity in terms of number of statements can be found in Figure 9. Informally, it was found that queries of length 5-7 statements were able to capture most reasonable trip queries. Some longer queries were included to demonstrate the power and flexibility of the system, but in order to make such an elaborate query the NL sentences are often long and convoluted.

To illustrate what a complex query looks like, the query *Is there a transit route to my home that stops at a bank, grocery, and bar that arrives before 5:00pm and has fewer than 7 transfers?* has 10 statements, *What is the most reliable transit route to a restaurant that costs less than $2.50?* has 5, and *Find the best way to my apartment* has 3.

The set of 50 evaluation queries contains a total of 296 necessary statements. In terms of raw accuracy, the system achieved 96% in that it produced only 13 errors in total; the novice human generated only 3 errors, for an accuracy of 99% for the 296 statements. All of the system errors occurred within 10 of the queries, which means 80% of the queries were processed correctly. The novice processed 94% of the queries correctly. Of those 10 queries where the machine made errors, 1 had 3 errors, 1 had 2 errors, and the remaining 8 each had 1 error. Only 3 of the queries with errors had 6 or fewer statements—meaning that errors generally occurred on long, complicated queries

In addition to accuracy, I can calculate a measure of precision and recall for different aspects of the query by using the atomic unit of a statement. This measure makes sense as a statement can be incorrect in two ways: the case that the statement is wrong and the case that the statement is missing. Table VIII and Table IX show where these errors occur. Table VIII provides the details for which clause the error occurred in, and Table IX details the types of errors found within the <where clause>. I do not include a detailed analysis of the novice human, since his 3 errors were all in the <optimize clause>—making the novice worse than the expert, who had 0 errors, but better than the machine.

TABLE VIII

PRECISION AND RECALL FOR DIFFERENT CLAUSES (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN STATEMENTS WITHIN EACH OF THE CLAUSES)

	Precision	Recall	F-Score
Select	.95	.95	.95
With Modes	1.00	1.00	1.00
With Stop Vertices	.88	.79	.83
With Certainty	1.00	1.00	1.00
Where	.94	.90	.92
Optimize	1.00	1.00	1.00
Total	*.97*	*.95*	*.96*

The types of errors chronicled in Table IX deserve further explanation. Each line represents a class of errors. Temporal errors are errors that relate to temporal constraints in the <where clause>. The primary error was that of sign inversion (e.g., specifying that a trip end before, rather than after, 8:00pm). Temporal constraints were often the most missed type of constraint—i.e., they were not included in the output in any form whatsoever. Facility constraints are those specifying that the trip include a certain resource along the way (e.g., stopping at the bank). The most common error was swapping an intermediate node with the resource

TABLE IX

PRECISION AND RECALL FOR DIFFERENT CONSTRAINTS (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN FOR STATEMENTS OF A GIVEN CONSTRAINT TYPE)

	Precision	Recall	F-Score
Temporal	.93	.87	.90
Resource	.97	.94	.96
Geographic	1.00	1.00	1.00
Cost	1.00	1.00	1.00
Transfers	1.00	1.00	1.00
Total	*.96*	*.93*	*.95*

with the destination node—this also affects the precision and recall for the <select clause> and <with stop vertices clause> in addition to the <where clause>.

I find these results to be very promising; the overall accuracy is high and the errors are more prevalent in long, complicated queries which seem less likely to be issued in real world applications. It is important to remember that these queries are on task and meet a minimal standard of correctness in their design. I discussed some of the ways in which the system is robust to imperfect input in Section 5.4 and will present additional thoughts on informal evaluative measures in the following subsection.

6.2.3 Paraphrases and Fragments

In order to test the robustness of the system, I decided to systematically rewrite the first set of 50 test queries used in the previous evaluation. To illustrate, I provide the following two examples. Query A is the initial query, B is the paraphrased rewrite, and C is the fragmented rewrite.

Example 1

A. *What is the cheapest way to visit my bank, a bakery, and a gym on they way to my office.*

B. *Start by visiting my bank, a baker, and a gym and then go to my office for the cheapest amount of money.*

C. *cheapest way to my office after bank, bakery, gym*

Example 2:

A. *Can I drive to a bank not crossing a river while arriving no later than 5:45pm?*

B. *Arriving at a bank no later than 5:45, can I do it by driving and not crossing a river?*

C. *drive to bank not crossing river by 5:45pm*

In each case, query B is rewritten in order to capture variability in how queries may be asked. Short queries had additional clauses added even if they were not necessary. Long queries had their clauses rearranged or compacted. Where possible, I changed the verb tenses or used synonyms. The fragmented queries leave behind mostly keywords—all of the important information for the query remains, but it is impossible to perform a full parse of the input because it does not follow any grammatical convention.

For this experiment I only compared the system output to the correct output; I did not involve another human translator. In each case the output for queries A, B, and C should be identical and I already have the correct output for each A query to compare against. Similarly, the complexity of the queries remains the same in terms of number of statements required—reinforcing the fact that query complexity is not tied to a specific representation of the query.

To begin, recall that in terms of raw numbers only 10 of the 50 A queries had any type of error. For the B queries, once again 10 of the queries resulted in an error. The C queries fared slightly better with only 9 queries resulting in any error. These results show that the system is robust in handling queries that are expressed in varying ways—the user is not required to follow a prescribed grammar.

While the overall number of erred queries are roughly equal, the errors did not occur in the same queries. Of the incorrectly translated A queries, the B queries produced errors in 2 and C queries produced errors in 1. That means that there were 8 B queries and 8 C queries where there was an error, but the original A was handled correctly. The queries in which errors were made generally had a greater number of errors. This is reflected in Table X and Table XI, which contain the same precision and recall statistics, as calculated for the A queries, for the B queries. The most significant difference, when compared to the A results is in the resource constraints, and therefore the <with stop vertices> clause. Table XII and Table XIII contain the same statistics for the C queries, which are comparable to the B queries.

It is somewhat surprising that the rewritten queries were processed correctly even though the initial query was not, as the rewritten forms were generally less grammatical. I was unable to find any one consistent reason for why the system performed better in these cases, though it appears that prepositional and adjectival attachment errors in the parse are a likely culprit in many cases.

TABLE X

PRECISION AND RECALL FOR DIFFERENT CLAUSES IN *B* QUERIES (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN STATEMENTS WITHIN EACH OF THE CLAUSES)

	Precision	Recall	F-Score
Select	.98	.98	.98
With Modes	1.00	1.00	1.00
With Stop Vertices	.63	.63	.63
With Certainty	1.00	1.00	1.00
Where	.95	.95	.95
Optimize	1.00	1.00	1.00
Total	*.88*	*.96*	*.91*

6.2.4 Implicit Queries

As discussed previously, a query such as *Find the nearest bank* can be interpreted as an implicit query for a trip that goes to the nearest bank—even though the user was really asking for only the location of the nearest bank. Similarly, the query *Is there a bus to my house after 10pm?* can be interpreted as looking for a trip meeting a temporal constraint rather than a binary yes/no response. In the earliest stages of development, I considered allowing queries such

TABLE XI

PRECISION AND RECALL FOR DIFFERENT CONSTRAINTS IN *B* QUERIES (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN FOR STATEMENTS OF A GIVEN CONSTRAINT TYPE)

	Precision	Recall	F-Score
Temporal	.88	.88	.88
Resource	.74	.72	.73
Geographic	1.00	1.00	1.00
Cost	1.00	1.00	1.00
Transfers	1.00	1.00	1.00
Total	*.82*	*.80*	*.81*

TABLE XII

PRECISION AND RECALL FOR DIFFERENT CLAUSES IN *C* QUERIES (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN STATEMENTS WITHIN EACH OF THE CLAUSES)

	Precision	Recall	F-Score
Select	1.00	1.00	1.00
With Modes	1.00	1.00	1.00
With Stop Vertices	.63	.63	.63
With Certainty	1.00	1.00	1.00
Where	.97	.97	.97
Optimize	1.00	1.00	1.00
Total	*.97*	*.97*	*.97*

TABLE XIII

PRECISION AND RECALL FOR DIFFERENT CONSTRAINTS IN C QUERIES (EFFECTIVELY, THE RATE AT WHICH ERRORS OCCURRED IN FOR STATEMENTS OF A GIVEN CONSTRAINT TYPE)

	Precision	Recall	F-Score
Temporal	.81	.81	.81
Resource	.71	.73	.72
Geographic	1.00	1.00	1.00
Cost	1.00	1.00	1.00
Transfers	1.00	1.00	1.00
Total	*.78*	*.79*	*.79*

as these to be answered explicitly and presented a simple heuristic for classifying queries based on what type of target they were for (e.g., a trip, resource, time) [Booth et al., 2009]. However, I found that focusing on trips alone actually provides greater functionality and accuracy. If a user asks whether or not they can take a bus home after a certain time, telling them which bus they can take is more useful than a simple “yes” or “no” response.[1]

Some of the previous test queries were of this form as well, so given the robust results presented in the previous sections, it is not surprising that queries written like this can be processed correctly. In order to further validate this, I tested an additional set of 12 queries and all were processed correctly. In addition to the two queries mentioned just above, *Can*

[1]This assumption is supported by notions of indirect speech acts and pragmatic implicature [Searle, 1965; Grice, 1975].

I drive to a pharmacy before 7:30? and *Where is the nearest gas station?* are also implicit queries that were tested. The full list of queries is found in Chapter 9.

6.2.5 Informal Evaluation

After performing a formal evaluation of NL2TRANQUYL I decided to pursue a further, but significantly less formal, evaluation in order to have a better understanding of the system coverage for potential real-world system deployment. The approach and focus was similar to that of the initial informative data collection. I solicited queries from laypersons with no experience with this research. This was done by posting a "note" on Facebook (http://www.facebook.com) and asking random contacts to read the note and respond with queries. The note contained a list of dozen queries from the evaluation corpus as well as a brief description of the project and its goals. For generating new queries, the contacts were told,

> At this time, the system is really designed for trip planning (i.e., find a route from an origin to a destination subject to various constraints). Below are some examples of questions that my system does (or should) understand. It should give you an idea of the scope of the language that the system works with. Feel free to play with the grammar and vocabulary a bit; not everthing is captured in the list below.

The use of Facebook for data collection is non-traditional, but provided useful qualities. Unlike in the previous query collection steps, I wanted users who were not familiar with the work—which eliminates colleagues and students in my research area. Facebook provided a free method to contact many different users from many different backgrounds (race, age, location, education).

I was interested in determining to what extent users would be able to use the system for their needs with only minimal training, as well as in what directions I should take the project in the future. From this solicitation I received 42 new queries. Of these, roughly one third were immediately compatible with the system, with the caveat that certain landmarks needed to be added to the ontology. At this time I do not have a comprehensive database of all possible locations of interest for each user, and the users were allowed to ask open ended questions.

Roughly another third of the queries would be understandable within the framework with small extensions. These queries included new metrics (e.g., cleanliness and safety of a route, presence of traffic control signals) and references to *other users'* profile (e.g., *Find a route to Robert's house.*). Some were queries looking for an attribute of the trip (e.g., *How long does it take to walk to my house?*)—the implicit trip query is understood but there is no way to explicitly ask for the desired information. In addition to being able to avoid spatial regions (e.g., parks), one user wanted to avoid specific roads in the network.

While it might be easy to view these unsupported queries as shortcomings, I take a more optimistic approach. The system design is flexible enough to add the desired features without a significant reworking; the ontology driven model of constraints allows for straightforward extensions. This is important because no system will immediately address all user wishes, especially when they are allowed to express those wishes in natural language.

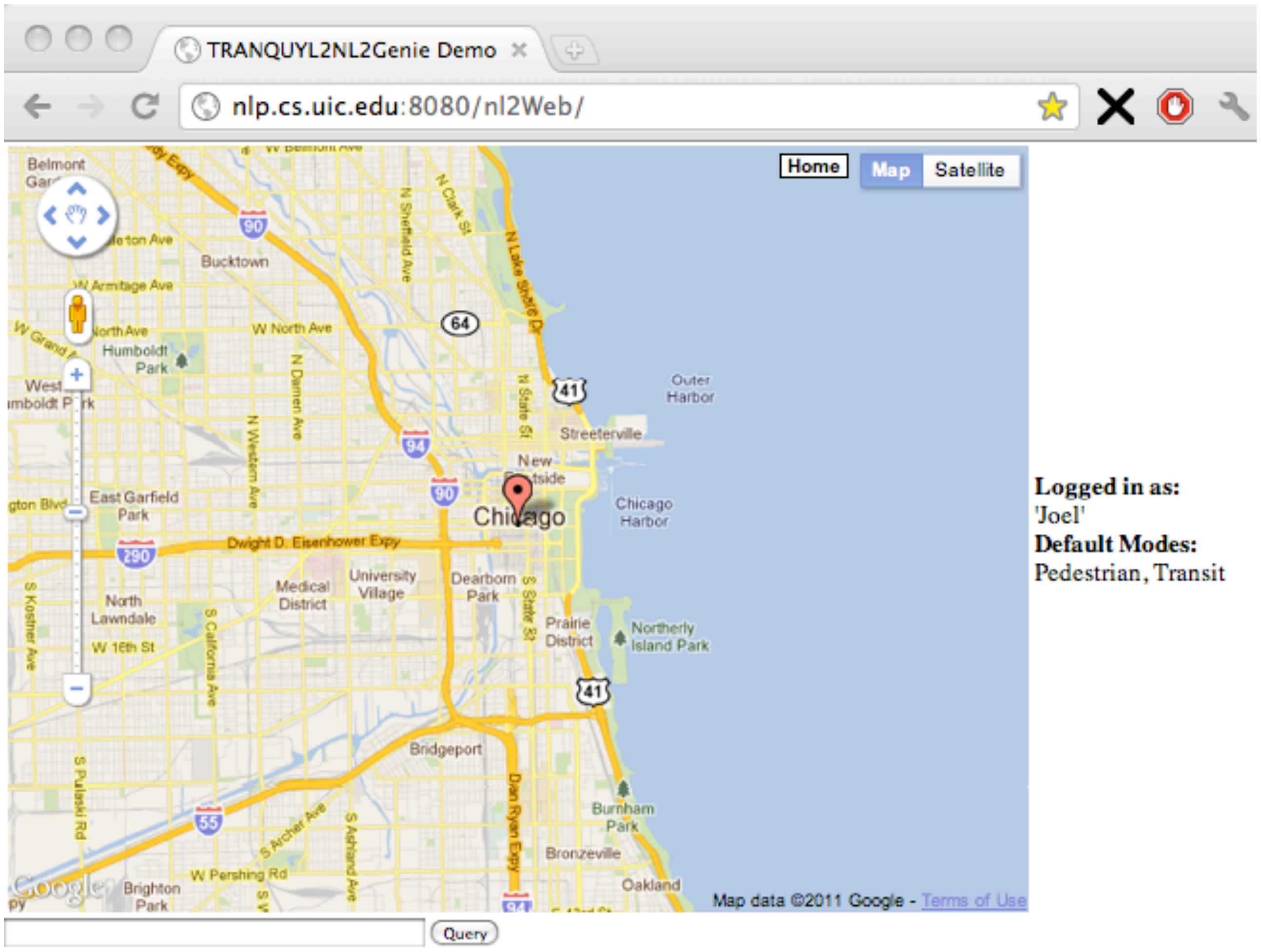
The queries that were clearly beyond the scope of the work involved tasks such as coordinating trips for multiple people so that they would arrive at some mutually agreeable location (e.g., *Where can I go for dinner that is easy for Alice and Bob to meet me?)*, multi-part queries that seek information beyond the data model (e.g., *Find a bus to the nearest hotel... how was it rated by previous guests*, *Which bus goes to the airport? ... Is it air-conditioned?*)

The final use for the results obtained in this informal evaluation is that I understand the scope of training and guidance needed for future users of the system. While the phrasing and formality of the language is flexible, there are still transportation-centric queries that may be related to trip planning but are beyond the understanding of the system as it stands.

6.3 NL2TRANQUYL2WEB: A Web-Based Demo

In order demonstrate some of the capabilities on NL2TRANQUYL, I developed a small web-based demonstration. A screenshot of the home page of the demo is found in Figure 10. The user is represented by the movable marker on the embedded Google Map. The marker is dragged to the desired position, the query is entered in natural language, and when submitted the query is parsed using NL2TRANQUYL (running as a web service). The NL2TRANQUYL service uses the coordinates of the marker to define its current location. The display also shows the current user and their default modes of transportation. Note that in this demo, the only user that exists is Joel. It was not practical to populate the necessary data fields for multiple users for the demo.

The demo is capable of showing only a subset of the queries that NL2TRANQUYL understands. This is because the routing services used only accept an origin, destination, and a start or end time. One cannot specify the other constraints allowed in TRANQUYL. If constraints not compatible with the services are included, they are simply ignored.


TRANQUYL2NL2Genie Demo
nlp.cs.uic.edu:8080/nl2Web/
Home
Map
Satellite
Query
Map data ©2011 Google - Terms of Use

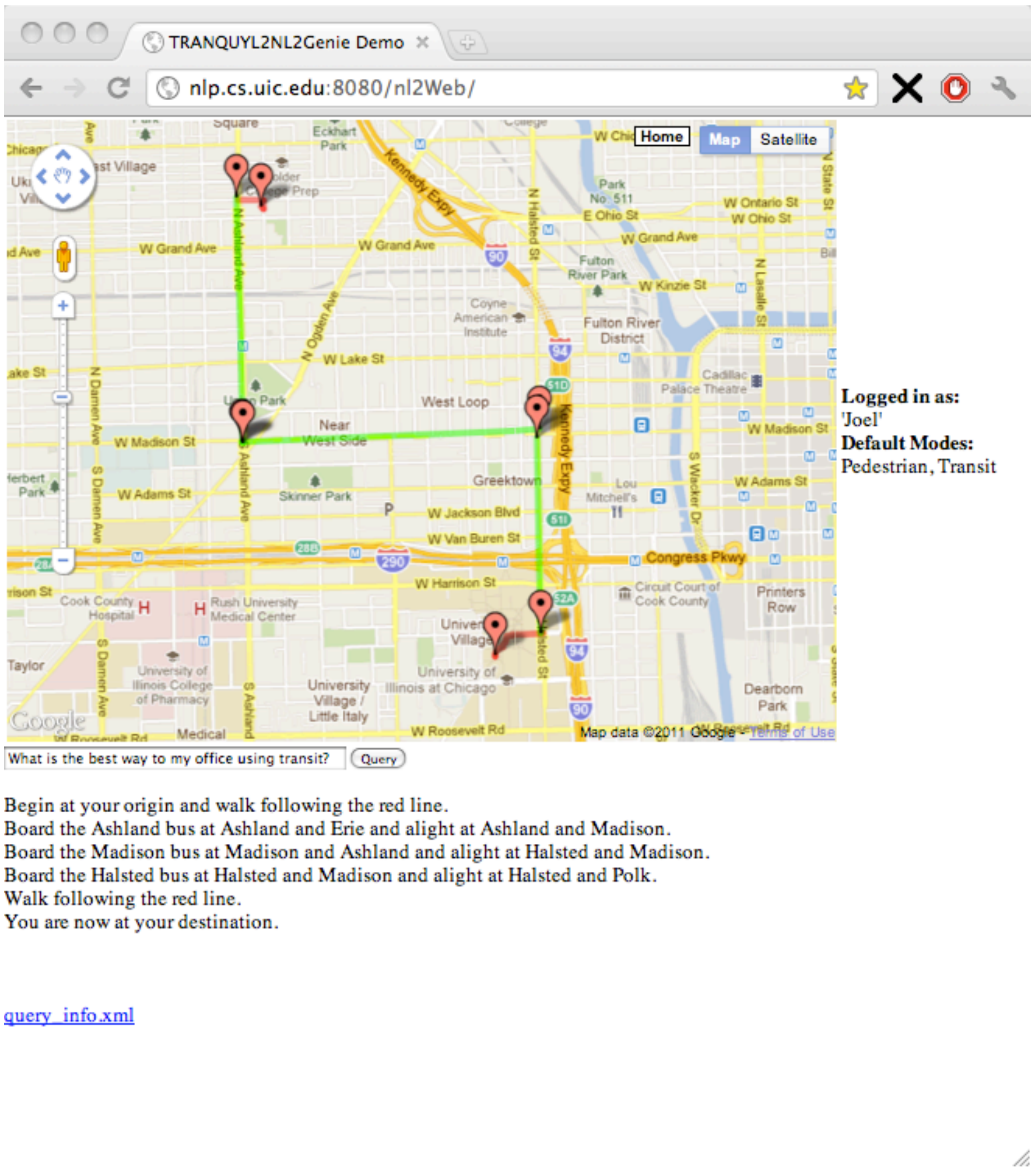
Logged in as:
'Joel'
Default Modes:
Pedestrian, Transit

Figure 10. The web interface

The system allows for two different routing methods. The first is TransitGenie [Biagioni et al., 2009], a multimodal routing system that utilizes realtime transit information. This is the method used for the bus, train, and pedestrian modes in the demo. Figure 11 shows a query and the resulting directions for a query that explicitly asks to use transit. Figure 12 shows that given a query that does not specify the modes, the default modes will be used. In each of these instances, the address of the destination was supplied explicitly. In the first, it was contained in the user ontology where information about "my office" is modeled. In the second it was specified as a string in the query. In each case, a link to the query information is displayed. The XML result includes the TRANQUYL parse, the TransitGenie query string, and TransitGenie query response. A screen shot of an XML result can be found in Figure 15.

The graphical representation (map overlay) and text directions are simple, but effective at conveying the necessary information. The natural language generation is very simple: each leg of the trip is generated independently through a template. The transit path overlays do not follow the actual path geometry in all cases. This is due to a difference in representations between TransitGenie and Google Maps.

For driving directions, the routing is done using the API provided by Google Maps. Figure 13 and Figure 14 show two driving queries and their respective results. The queries also demonstrate the difference in how a personal reference (i.e., *my bank*) is treated differently than a generic reference (i.e., *a bank*). The former routes the trip to a predetermined location and the latter looks for the nearest bank and selects it as a destination. The finding of a generic resource is done through the Google Places API. The specified type of generic resource is mapped

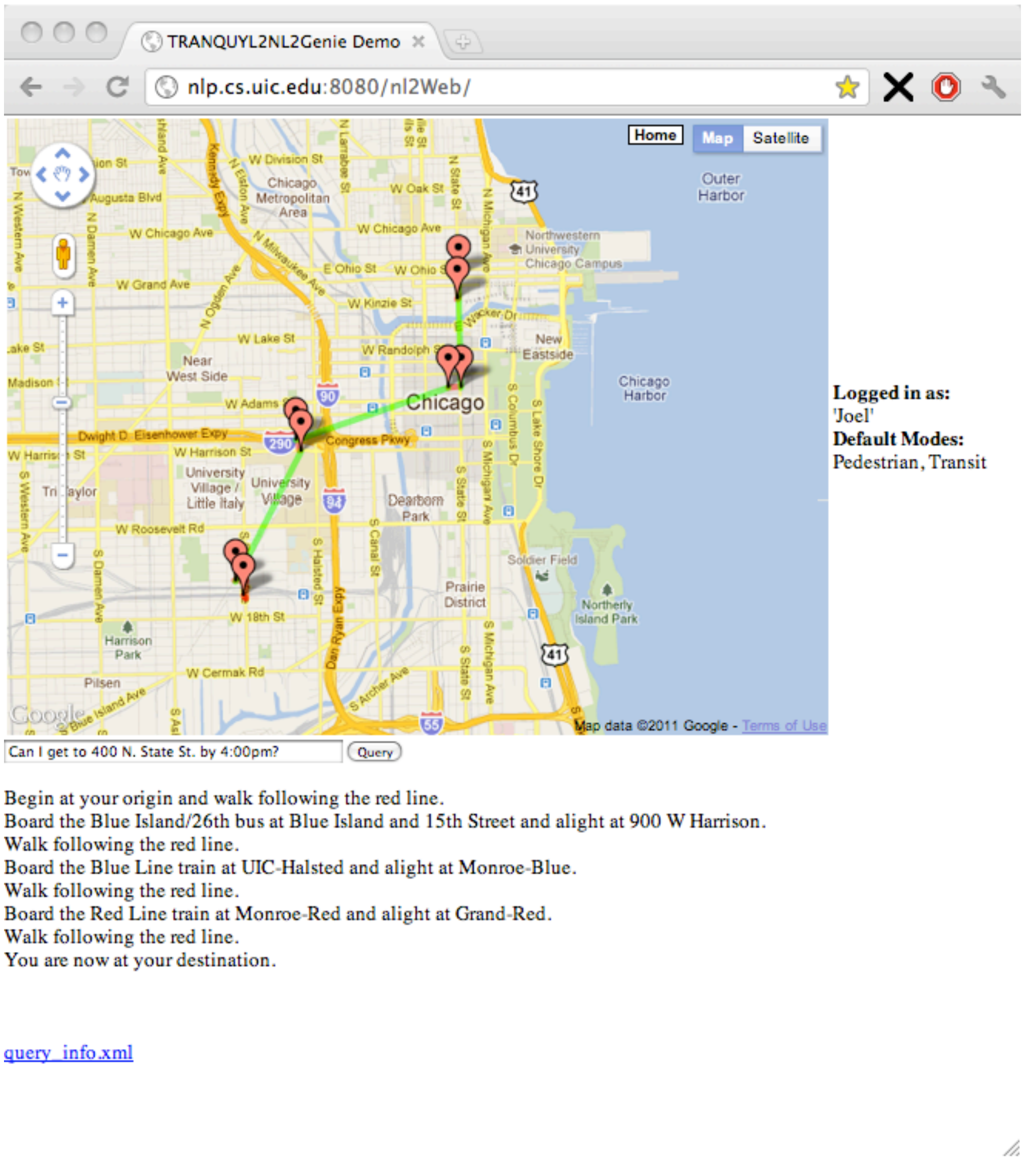


Logged in as:
'Joel'
Default Modes:
Pedestrian, Transit

What is the best way to my office using transit?

Begin at your origin and walk following the red line.
Board the Ashland bus at Ashland and Erie and alight at Ashland and Madison.
Board the Madison bus at Madison and Ashland and alight at Halsted and Madison.
Board the Halsted bus at Halsted and Madison and alight at Halsted and Polk.
Walk following the red line.
You are now at your destination.

Figure 11. Simple transit directions


TRANQUYL2NL2Genie Demo
nlp.cs.uic.edu:8080/nl2Web/
Home
Map
Satellite
Chicago
Query
Map data ©2011 Google - Terms of Use

Logged in as:
'Joel'
Default Modes:
Pedestrian, Transit

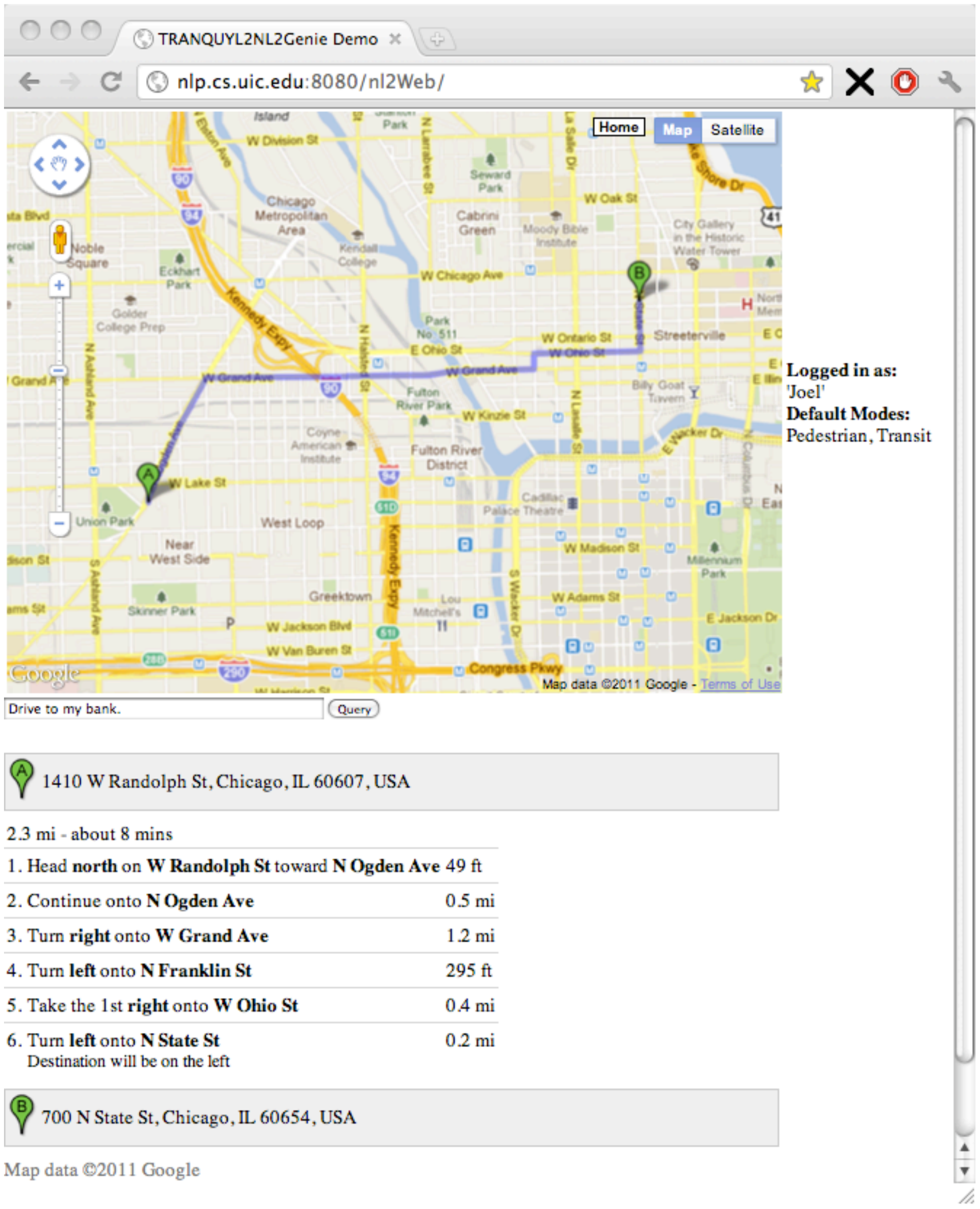
Can I get to 400 N. State St. by 4:00pm?

Begin at your origin and walk following the red line.
Board the Blue Island/26th bus at Blue Island and 15th Street and alight at 900 W Harrison.
Walk following the red line.
Board the Blue Line train at UIC-Halsted and alight at Monroe-Blue.
Walk following the red line.
Board the Red Line train at Monroe-Red and alight at Grand-Red.
Walk following the red line.
You are now at your destination.

query_info.xml

Figure 12. Default transit directions

to the appropriate definition in the Google Places API. In these cases the Google Maps query string and response, rather than TransitGenie, are included in the XML result. Both graphical and textual directions are provided by the Google API.


TRANQUYL2NL2Genie Demo
nlp.cs.uic.edu:8080/nl2Web/
Home
Map
Satellite
Map data ©2011 Google - Terms of Use
Query

Logged in as:
'Joel'
Default Modes:
Pedestrian, Transit

Drive to my bank.

1410 W Randolph St, Chicago, IL 60607, USA

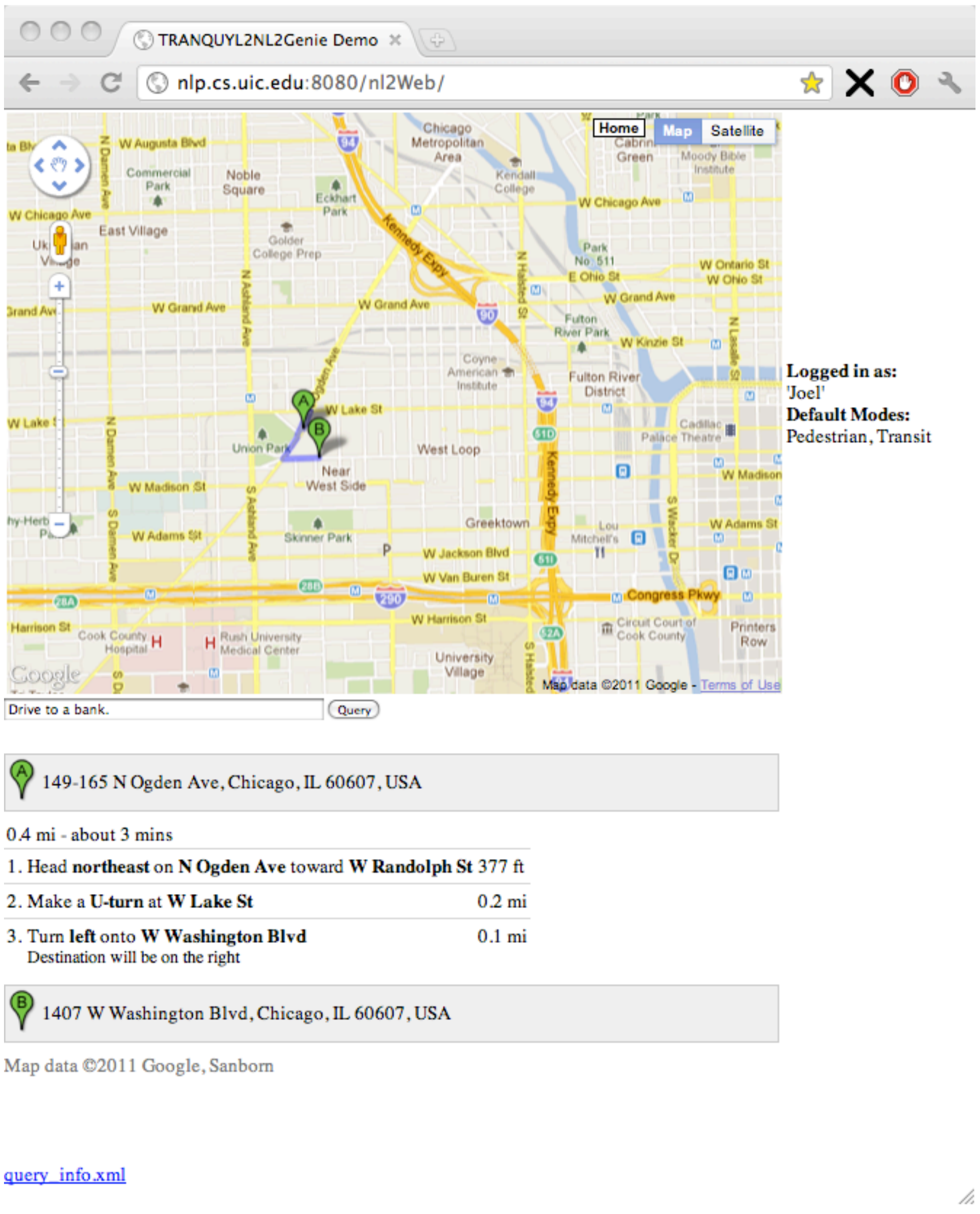
2.3 mi - about 8 mins

1. Head **north** on **W Randolph St** toward **N Ogden Ave**	49 ft
2. Continue onto **N Ogden Ave**	0.5 mi
3. Turn **right** onto **W Grand Ave**	1.2 mi
4. Turn **left** onto **N Franklin St**	295 ft
5. Take the 1st **right** onto **W Ohio St**	0.4 mi
6. Turn **left** onto **N State St** Destination will be on the left	0.2 mi

700 N State St, Chicago, IL 60654, USA

Map data ©2011 Google

Figure 13. Driving to a personal reference



Logged in as:
'Joel'
Default Modes:
Pedestrian, Transit

Drive to a bank.

149-165 N Ogden Ave, Chicago, IL 60607, USA

0.4 mi - about 3 mins

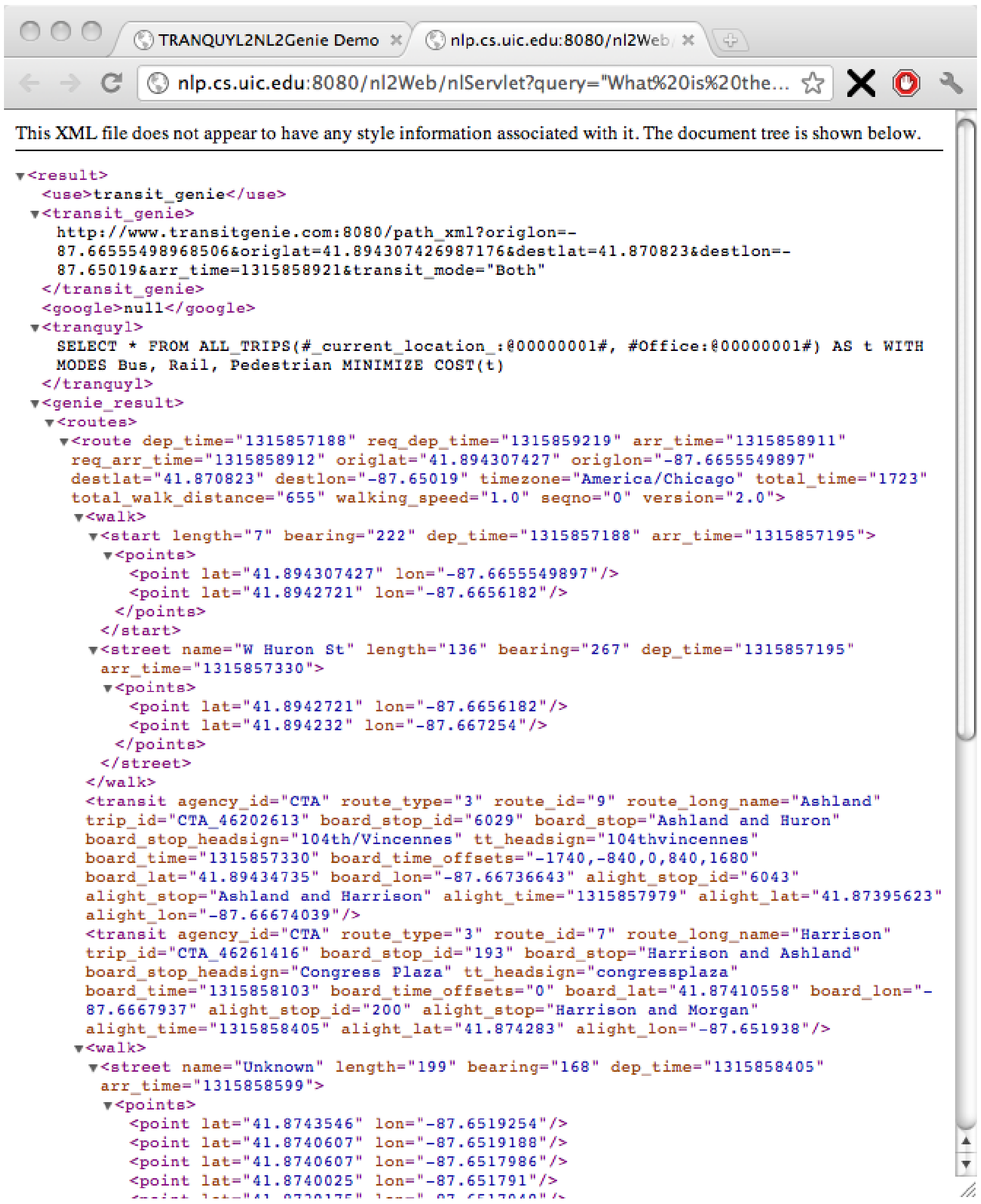
1. Head **northeast** on **N Ogden Ave** toward **W Randolph St** 377 ft
2. Make a **U-turn** at **W Lake St** 0.2 mi
3. Turn **left** onto **W Washington Blvd** 0.1 mi
   Destination will be on the right

1407 W Washington Blvd, Chicago, IL 60607, USA

Map data ©2011 Google, Sanborn

query_info.xml

Figure 14. Driving to a generic resource


TRANQUYL2NL2Genie Demo
nlp.cs.uic.edu:8080/nl2Web
nlp.cs.uic.edu:8080/nl2Web/nlServlet?query="What%20is%20the...

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<result>
   <use>transit_genie</use>
 ▼<transit_genie>
     http://www.transitgenie.com:8080/path_xml?origlon=-
     87.66555498968506&origlat=41.894307426987176&destlat=41.870823&destlon=-
     87.65019&arr_time=1315858921&transit_mode="Both"
   </transit_genie>
   <google>null</google>
 ▼<tranquyl>
     SELECT * FROM ALL_TRIPS(#_current_location_:@00000001#, #Office:@00000001#) AS t WITH
     MODES Bus, Rail, Pedestrian MINIMIZE COST(t)
   </tranquyl>
 ▼<genie_result>
   ▼<routes>
     ▼<route dep_time="1315857188" req_dep_time="1315859219" arr_time="1315858911"
       req_arr_time="1315858912" origlat="41.894307427" origlon="-87.6655549897"
       destlat="41.870823" destlon="-87.65019" timezone="America/Chicago" total_time="1723"
       total_walk_distance="655" walking_speed="1.0" seqno="0" version="2.0">
       ▼<walk>
         ▼<start length="7" bearing="222" dep_time="1315857188" arr_time="1315857195">
           ▼<points>
               <point lat="41.894307427" lon="-87.6655549897"/>
               <point lat="41.8942721" lon="-87.6656182"/>
             </points>
           </start>
         ▼<street name="W Huron St" length="136" bearing="267" dep_time="1315857195"
           arr_time="1315857330">
           ▼<points>
               <point lat="41.8942721" lon="-87.6656182"/>
               <point lat="41.894232" lon="-87.667254"/>
             </points>
           </street>
         </walk>
         <transit agency_id="CTA" route_type="3" route_id="9" route_long_name="Ashland"
         trip_id="CTA_46202613" board_stop_id="6029" board_stop="Ashland and Huron"
         board_stop_headsign="104th/Vincennes" tt_headsign="104thvincennes"
         board_time="1315857330" board_time_offsets="-1740,-840,0,840,1680"
         board_lat="41.89434735" board_lon="-87.66736643" alight_stop_id="6043"
         alight_stop="Ashland and Harrison" alight_time="1315857979" alight_lat="41.87395623"
         alight_lon="-87.66674039"/>
         <transit agency_id="CTA" route_type="3" route_id="7" route_long_name="Harrison"
         trip_id="CTA_46261416" board_stop_id="193" board_stop="Harrison and Ashland"
         board_stop_headsign="Congress Plaza" tt_headsign="congressplaza"
         board_time="1315858103" board_time_offsets="0" board_lat="41.87410558" board_lon="-
         87.6667937" alight_stop_id="200" alight_stop="Harrison and Morgan"
         alight_time="1315858405" alight_lat="41.874283" alight_lon="-87.651938"/>
       ▼<walk>
         ▼<street name="Unknown" length="199" bearing="168" dep_time="1315858405"
           arr_time="1315858599">
           ▼<points>
               <point lat="41.8743546" lon="-87.6519254"/>
               <point lat="41.8740607" lon="-87.6519188"/>
               <point lat="41.8740607" lon="-87.6517986"/>
               <point lat="41.8740025" lon="-87.651791"/>
```

Figure 15. The partial XML result for the query in Figure 11

CHAPTER 7

CONCLUSIONS

In this dissertation I have presented three key contributions to the community,

1. a model for multimodal urban transportation networks,
2. a query language for trip planning,
3. a natural language interface

These contributions, in concert, represent not only an approach to providing transportation information to users in a usable, cohesive fashion, but they also show how integrating components from multiple interdisciplinary areas can yield important practical and theoretical gains for each field.

The transportation network model is unique in that it was designed with both computer science and urban transportation uses in mind. It is graph based, and therefore open to numerous existing applications; but it was also designed in a manner to make it appropriate for translation to a relational database. Unlike previous models, it contains mechanisms to handle spatial, temporal, and uncertain data. Other models only capture some subset of these.

On top of this model is built the Transportation Query Language. This is the first query language that is designed with trip planning as a focus. It allows for succinct and elegant expression of a wide range of trip-based queries. It was shown that its expressive power was greater than existing graph query languages, and no other query language is as intuitive for trip planning due to the purpose-built nature of TRANQUYL. The clever use of well known spatio-temporal data types and operators means that it could be integrated with other database systems in the future. And importantly, unlike proprietary languages, it is open and available for anyone to use and build on.

The language is well defined with both syntax and semantics. The wide range of expressible queries were demonstrated through numerous examples. While there was no database management system implemented as a component of this dissertation, efforts were made to show how the necessary information could be stored and a framework for query processing was developed.

The natural language interface is an important step in bringing information to real users. While TRANQUYL is intuitive for its purpose, it is still a formal query language that requires some technical proficiency. It is not as accessible or intuitive as natural language itself. For the average user, asking questions in the manner in which they have been doing so their entire life is straightforward. Previous research has shown that natural language is an effective input modality in many circumstances, especially in the hands-busy or eyes-busy situations that are common when traveling.

It is for this reason that I proposed and developed an NLI that is both robust and flexible. The system does not rely on a specific, prescribed grammar. Through the use of linguistically motivated heuristics and a domain ontology, it parses the natural language query and finds key concepts. Using the domain ontology, it generates a knowledge map from these concepts—filling in missing pieces as needed. The map is then translated into the appropriate TRANQUYL query.

The use of a domain ontology as a bridge between natural language and a database language is somewhat unique. Generally, ontologies serve as the target for natural language interfaces rather than an additional mediating resource. The ontology was used to expand the vocabulary as well as capture personal references—a feature not present in many NLIs. Being able to distinguish between personal and generic resources is important for usability.

NL2TRANQUYL was shown to be effective through an intrinsic study in which the software was pitted against a query writer. Both the human and software were given the same natural language input and made to generate the formal queries. These queries were then compared against a gold standard to calculate accuracy. While the human was better than the software, he still made mistakes. The software's accuracy is considered to be very high for a natural language processing application. The queries that weren't translated correctly were generally those that were the most complicated. My intuition is that the more complicated and error prone queries are more complicated and convoluted in structure than what a user would normally ask.

While this work is comprehensive in scope and the primary goals were achieved, there are a number of directions for which future work could occur. The first, and most involved, would be the implementation of the database management system. That, along with the procurement of the necessary data, would allow for a full scale deployment of this work on the proposed Intelligent Traveler's Assistant device. The second and most related task would be further extending the graph and relational models. There are certainly aspects of the transportation network that are not fully captured at this time. A full model of pricing, both road and parking, would be a useful extension. Currently only urban passenger transportation is considered, but in the future extensions for freight and airline travel could be included.

The NLI is robust and effective, but like all natural language systems it is not comprehensive in its coverage. It does not cover the entire range of queries expressible in TRANQUYL. Queries for multiple trips, comparisons of different trips, and certain constraints are not yet understood by NL2TRANQUYL. Each of these features is significantly more complicated than what is currently done. Pending investigation of user needs, it may be worthwhile to add such functionality in the future.

While these directions for future work are certainly interesting, they do not weaken the overall contribution of this dissertation. A cohesive data model, query language, and natural language interface—all designed to work together in the domain of trip planning in urban environments—were presented. Together they constitute a substantial contribution to the field of computer science and related problems in urban planning.

APPENDICES

CHAPTER 8

TRANQUYL ONTOLOGY CLASS HIERARCHY

The following classes comprise the TRANQUYL ontology. The parent class `thing` has been omitted to ease readability. Note that some classes have multiple parent classes. Some classes, such as `Place`, are open to rapid extension for adding vocabulary to NL2TRANQUYL.

```
Origin
Destination
Trip
User
Event
  - Arrival
  - Departure
  - Transfer
  - Traveling
  - Visit
Geographic
  - Forest
  - Park
  - River
Mode
  - Auto
  - Bike
  - Bus
  - Pedestrian
  - Rail
  - Transit
Optimals
  - Cheap
  - Expensive
  - Fewest Transfers
  - Most Reliable
  - Spatial Short
  - Temporal Short
```

```
Spatial Concept
  - Physical
    * Path
    * Point
    * Region
  - Relational
    * Avoids
    * Intersects
Temporal Concept
  - Instant
    * Time
  - Period
 Personal
 - Dwelling
 - Work
```

```
Metric
 - Cost Metric
   * Cheap
   * Expensive
 - Reliability Metric
   * Arbitrary Reliability
   * Most Reliable
 - Temporal Metric
   * Duration
     ~ Temporal Long
     ~ Temporal Short
   * Proximal
     ~ Soon
       - Spatial Metric
   * Distance
     ~ Spatial Short
     ~ Spatial Long
   * Proximity
     ~ Near
     ~ Far
   * Size
     ~ Big
     ~ Small
 - Transfer Metric
   * Number of Transfers
```

```
Place
- Address
- Airport
- ATM
- Aquarium
- Baker
- Bank
- Bar
- Book Store
- Cafe
- Dentist
- Doctor
- Florist
- Grocery Store
- Gym
- Hospital
- Library
- Office
- Pharmacy
- Post Office
- Restaurant
- School
- Stadium
- Theater
- Work
```

CHAPTER 9

TEST QUERIES

The following are all of the test queries used in the evaluation. They are labeled as *a, b,* and *c* has referred to in the evaluation of NL2TRANQUYL (Section 6.2).

1a. Find the best way to my apartment.
1b. Show the best path to where I live.
1c. best my apartment
2a. Tell me how to get home with public transportation.
2b. how to get home with public transportation.
2c. home transit
3a. Give me the shortest path to a grocery store by 7:00pm with 50% certainty.
3b. find shortest trip to grocery by 7:00pm, 50% certainty.
3c. shortest grocery 7:00pm 50%
4a. What is the cheapest way to visit my bank, a bakery, and a gym on they way to my office.
4b. Start by visiting my bank, a baker, and a gym and then go to my office for the cheapest amount of money.
4c. cheapest way to my office after bank, bakery, gym
5a. What is the most reliable train to home?
5b. Find a path with a train that gets me home with the best reliability.
5c. reliable train home
6a. Find me a way to the theater with transit that has the fewest transfers and arrives by 8:30pm with 95% likelihood.
6b.To the theater by 8:300pm with 95% certainty using transit and fewest transfers.
6c. transit to theater with fewest transfers by 8:30pm, 95% certain
7a. Find a way to home that visits a bar with transit.
7b. Going to home, can I visit a bar with transit.
7c. transit to home with bar
8a. I need to take a bus or walk to my apartment while stopping at a grocery.
8b. Visiting a grocery store, can I walk or bus to my apartment.
8c. to apartment with grocery by bus or walk
9a. I want to walk to home through a park.
9b. Is there a way that I can walk to my home that passes through a park?
9c. walk to home through park
10a. Get a route to my office that stops at a florist and my doctor using transit.
10b. Stopping by a florist and my doctor, is there a way that I can take transit to my office?
10c. to my office, with florist and my doctor with transit
11a. Give me the cheapest path to a dentist that stops by my doctor on the way.
11b. Including a stop at my doctor in the middle, find a way to a dentist that is cheapest.

11c. to dentist with doctor, cheapest
12a. What is the shortest way to walk to my house?
12b. In what way can I walk to my house that is the shortest distance.
12c. walk my house shortest
13a. Find the fastest trip that arrives at my doctor and leaves from work that stops at a florist with less than 3 transfers.
13b. From work to my doctor, find the fastest trip with less than 3 transfers.
13c. from work to my doctor fastest with less than 3 transfers.
14a. What is the most reliable transit route to a restaurant that costs less than $2.50?
14b. What is the most reliable way to take public transportation to a restaurant where the trip costs less than $2.50?
14c. reliable to restaurant for less than $2.50 fewer 3 transfers.
15a. Find the shortest pedestrian route to the hospital.
15b. Using the pedestrian mode, what is the route that has the shortest distance that will take me to a hospital?
15c. shortest pedestrian to hospital
16a. Is there a way to home that includes a pharmacy?
16b. Going to my home, is it possible to include a pharmacy along the way?
16c. to home with pharmacy
17a. What is the shortest path to drive to work that also goes to a bank and doesn't cross a river?
17b. Driving to work, is there a shortest path that doesn't cross a river but also includes a bank to stop at?
17c. shortest drive to work with bank don't cross river
18a. Can I bus it to the bank for less than $6?
18b. Taking a bus on the way, can I go to a bank for less than $6?
18c. bus to bank less than $6
19a. Find the least expensive route to my apartment that doesn't cross a river and arrives by 7:00pm with 99% certainty
19b. Taking the least expensive path to my apartment, does there exist one that doesn't cross a river and still arrives no later than 7:00pm with 99% certainty on that time?
19c. least expensive to my apartment not cross river by 7:00pm with 99% certainty
20a. I need to find the shortest way to home that avoids parks.
20b. Avoid a park and find the shortest path to home.
20c. to home avoid park shortest
21a. Find me a way to a bar using a bus or train that arrives no later than 3:00pm.
21b. Arriving no later than 3:00pm, find a way to a bar that uses a bus or train.
21c. to bar with bus or train by 3:00pm
22a. What is the fastest way to home stopping at a bank after 9:00am?
22b. Stopping at a bank on the way to home, what is the fastest way to get there after 9:00am?
22c. fastest to home including bank by 9:00am
23a. Get a route to a restaurant before 5:30pm.

23b. Arriving at a restaurant before 5:30pm, find the way to do it.
24c. restaurant before 5:30pm.
24a. Get me to my home by 8:32pm using trains and busses for less than $6.
24b. For less than $6, is there a way that uses trains and busses that can get me to my home before 8:32pm?
24c. home by 8:32 with train and bus less than $6
25a. What transit route has the fewest transfers and gets me to home by 4:00pm while stopping at a bakery all for less than $7.95?
25b. Trying for the fewest transfers, what transit route will get me to home by 4:00pm that costs less than $7.95 and also stops at a bakery.
25c. transit to home with bakery by 4:00pm less than $7.95 with fewest transfers.
26a. I need a reliable way to home from work by 7:30pm.
26b. Taking the most reliable way possible, can I go from work to home and get there by 7:30pm?
26c. reliable from work to home by 7:30pm
27a. Can I walk to my house before 4:45am?
27b. Getting there before 4:45am, can I walk to my house?
27c. walk my house before 4:45am
28a. What transit route has the fewest transfers and gets me home by 4:00pm?
28b. Getting home by 4:00pm, which route can I take that uses transit and has the fewest transfers?
28c. transit with fewest transfers to home by 4:00pm.
29a. I want to drive to a theater and arrive no later than 3:12pm.
29b. Getting to a theater by 3:12pm, can I do it by driving?
29c. drive theater by 3:12pm
30a. Find a way to a bank that stops at a florist and pharmacist.
30b. Is it possible to stop and a florist and a pharmacist on the way to a bank – my destination?
30c. to bank with florist and pharmacist
31a. What is the least expensive way to 0 N. State St. and 0 W. Madison St.?
31b. I want to go to the intersection of 0 N. State St. and 0 W. Madison St., and I would like to do it the least expensive way possible.
31c. least expensive to 0 N. State St. and 0 W. Madison St.
32a. Can I walk to 300 W. Humbolt Blvd. by 4:00pm?
32b. Arriving by 4:00pm, is it possible to walk to 300 W. Hulmbolt Blvd.?
32c. walk 300 W. Humbolt Blvd. 4:00pm
33a. Find the cheapest way to 200 N. State St. and leaves no earlier than 7:00pm.
33b. Leaving no earlier than 7:00pm, what is the cheapest way I can walk to 200 N. State St.?
33c. cheapest to 200 N. State St. leave after 7:00pm
34a. I need a way to 1300 N. State St. that avoids parks and stops at a florist.
34b. Avoiding all parks and stopping at a florist, what is the best way to 1300 N. State St.?
34c. 1300 N. State St. with florist avoids parks
35a. Is there a path to 800 S. Halsted St. that goes through a park and leaves after 9:00am?

35b. Leaving after 9:00am, I want to go through a park and get to 800 S. Halsted St.
35c. 800 S. Halsted St. through park after 9:00am
36a. Leaving from 1000 W. Taylor St., what is the fastest way to my house for less than $7?
36b. For less than $7, can I find the fastest way to my house that leaves from 1000 W. Taylor St.?
36c. fastest from 1000 W. Taylor St. to my house less than $7
37a. Can I drive to a bank not crossing a river while arriving no later than 5:45pm?
37b. Arriving at a bank no later than 5:45, can I do it by driving and not crossing a river?
37c. drive to bank not crossing river by 5:45pm
38a. Is there a transit route to my home that stops at a bank, grocery, and bar that arrives before 5:00pm and has fewer than 7 transfers?
38b. Going to my house, use transit and also stop at a bank, grocery and bar, and do it for less than 7:00 transfers and arrive before 5:00pm.
38c. transit to my house with bank, grocery, bar before 5:00 fewer 7 transfers
39a. Arriving after 6:00pm, find the cheapest route to my apartment.
39b. Find the cheapest route to my apartment that arrives after 6:00pm.
39c. cheapest to my apartment after 6:00pm
40a. Leaving from my bank, get the shortest walking path to my doctor that gets there by 4:00pm.
40b. Getting to my doctor by 4:00pm, leave from my bank and take the shortest walking way.
40c. from my bank to my doctor by 4:00pm shortest walk
41a. What is the fastest transit route from my house to my office that stops at a bank, grocery store, and pharmacist and arrives before 9:00pm for less than $6.50?
41b. Going to my office from my home, I would like the fastest transit route than costs less than $6.50 and arrives before 9:00pm; also include stops at a bank, grocery store, and pharmacist on my way.
41c. fastest transit from my home to my office by 9:00pm less than $6.50 with bank, grocery, and pharmacist
42a. Get me to any bank, my doctor, and then arrive at my house by 3:00pm for less than $2.00.
42b. Going to my house by 3:00pm, find a way that includes any bank, my doctor, and do it for less than $2.00.
42c. to my house with bank and my doctor by 3:00pm less than $2.00
43a. Can I get to a bar from work by 5:30pm for less than $5.00, while stopping at my bank on the way?
43b. Arriving at a bar by 5:30pm, I want to leave from work and stop at a bank on the way – for less than $5.00.
43c. from work to a bar by 5:30 with bank and less than $5.00
44a. Is there a way to my doctor that stops at a pharmacy on the way?
44b. Stopping at a pharmacy, go to my doctor.
44c. to my doctor with pharmacy
45a. Which transit route home has the fewest transfers and allows me to visit a grocery store?

45b. With the fewest transfers that allows me to visit a grocery store, I want to take public transit to home.
45c. to home with grocery store by transit with fewest transfers
46a. Can I find a train that goes to a bar for less than $3.00 and leaves after 8:00pm?
46b. For less than $3.00, can I leave after 8:00pm and take a train to a bar?
46c. train to bar by 8:00pm less than $3.00
47a. What is the shortest way to walk to my apartment that goes through the park, but still gets there by 7:00pm?
47b. Walking to my apartment, what is the shortest way to do it that goes through a park and gets there by 7:00pm.
47c. shortest walk to my apartment through park by 7:00pm
48a. Can I locate a bus that goes to a grocery store and a bank for under $6.00 and finishes by 3:33pm?
48b. Finishing by 3:33pm and costing less than $6.00, can I go to a grocery and a bank by bus?
48c. to grocery with bank by bus 3:33pm less than $6.00
49a. What is the fastest transit trip to a bank that has fewer than 3 transfers, costs less than $3.00, and arrives before 2:00pm.
49b. Going to a bank for less than $3.00, can I arrive before 2:00pm by using fewer than 3 transfers on public transportation – going the fastest possible?
49c. fastest transit to bank less than 3 transfers by 2:00pm and less than $3.00
50a. What is the cheapest way I can take a train to work by 6:00am this morning?
50b. Getting to work by train by 6:00am this morning, what is the cheapest way to do it?
50c. cheapest train to work by 6:00am

The following 12 queries are those used for testing implicit queries (Section 6.2.4):

1. Is there a bus to my house after 10:00pm?
2. Find the nearest bank.
3. Can I drive to a pharmacy before 7:30pm?
4. Where is the nearest gas station?
5. Are there any parks on my way home?
6. How long does it take to drive home?
7. Is there an atm on my way to work?
8. Does a bus go to my doctor?
9. Could I walk to a grocery store before it closes at 9:00pm?
10. When is the soonest I could take a train to the theater?
11. Which bus goes the closest to my grocery store?
12. Are there any bars in the area?

CITED LITERATURE

Androutsopoulos, I., Ritchie, G., and Thanisch, P.: Natural language interfaces to databases–an introduction. Journal of Language Engineering, 1(1):29–81, 1995.

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

Angles, R. and Gutierrez, C.: Survey of graph database models. Technical report, Computer Science Department, Unviersidad de Chile, 2005.

Antoniou, G. and van Harmelen, F.: A Semantic Web Primer. The MIT Press, April 2004.

Ayed, H., Khadraoui, D., Habbas, Z., Bouvry, P., and Merche, J. F.: Transfer graph approach for multimodal transport problems. In Modelling, Computation and Optimization in Information Systems and Management Sciences, eds. H. A. Le Thi, P. Bouvry, and T. Pham Dinh, volume 14 of Communications in Computer and Information Science, pages 538–547. Springer Berlin Heidelberg, 2008.

eds. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider The Description Logic Handbook. Cambridge University Press, 2nd edition, 2007.

Bangalore, S. and Johnston, M.: Robust understanding in multimodal interfaces. Computational Linguistics, 35(3):345–397, 2009.

Bay Area Rapit Transit Planner: http://www.bart.gov/, 2011.

Benslimane, D., Leclercq, E., Savonnet, M., Terrasse, M.-N., and Y©tongnon, K.: On the definition of generic multi-layered ontologies for urban applications. Computers, Environment and Urban Systems, 24(3):191 – 214, 2000.

Berners-Lee, T., Hendler, J., and Lassila, O.: The semantic web. Scientific American, pages 29–37, May 2001.

Bernsen, N. O. and Dybkjær, L.: A multimodal virtual co-driver's problems with the driver. In CD-ROM Proceedings of the ISCA Tutorial and Research Workshop on Spoken Dialogue in Mobile Environments, International Speech Communication Association, Kloster Irsee, Germany, June 2002.

Bernstein, A., Kaufmann, E., and Kaiser, C.: Querying the semantic web with ginseng: A guided input natural language search engine. In 15th Workshop on Information Technology and Systems (WITS), pages 45–50, Las Vegas, Nevada, USA, December 2005.

Bernstein, A., Kaufmann, E., Kaiser, C., and Kiefer, C.: Ginseng: A guided input natural language search engine for querying ontologies. In Jena User Conference, Bristol, UK, May 2006.

Biagioni, J., Agresta, A., Gerlich, T., and Eriksson, J.: Transitgenie: A real-time, context-aware transit navigator (demo abstract). In ACM Conference on Embedded Networked Sensor Systems (SenSys), pages 329–330, Berkeley, CA, USA, November 2009.

Bielli, M., Boulmakoul, A., and Mouncif, H.: Object modeling and path computation for multimodal travel systems. European Journal of Operational Research (EJOR), 175(3):1705–1730, December 2006.

Black, A. W., Burger, S., Conkie, A., Hastie, H., Keizer, S., Lemon, O., Merigaud, N., Parent, G., Schubiner, G., Thomson, B., Williams, J. D., Yu, K., Young, S., and Eskenazi, M.: Spoken dialog challenge 2010: Comparison of live and control test results. In ACL Special Interest Group in Discourse and Dialogue (SIGDIAL), pages 2–7, Portland, OR, USA, June 2011.

Booth, J., Di Eugenio, B., Cruz, I. F., and Wolfson, O.: Query sentences as semantic (sub) networks. In The 3rd IEEE International Conference on Semantic Computing (ICSC), pages 89–94, Berkeley, CA, USA, September 2009.

Budanitsky, A. and Hirst, G.: Evaluating WordNet-based measures of lexical semantic relatedness. Computational Linguistics, 32(1):13–47, 2006.

Buehler, D., Vignier, S., Heisterkamp, P., and Minke, W.: Safety and operating issues for mobile human-machine interfaces. In International Conference on Intelligent User Interfaces (IUI), pages 227–229, Miami, FL, USA 2003.

Coen, M., Weisman, L., Thomas, K., and Groh, M.: A context sensitive natural language modality for the intelligent room. In First International Workshop on Managing Interactions in Smart Environments (MANSE), Dublin, Ireland, December 1999.

Cohen, P. R.: The role of natural language in a multimodal interface. In ACM symposium on User interface software and technology (UIST), pages 143–149, New York, NY, USA, 1992. ACM.

Coletti, P., Cristoforetti, L., Matassoni, M., Omologo, M., Svaizer, P., Geutner, P., and Steffens, F.: A speech driven in-car assistance system. In IEEE Conference on Intelligent Vehicles, Columbus, OH, June 2003.

Cruz, I. F., Mendelzon, A. O., and Wood, P. T.: A Graphical Query Language Supporting Recursion. In ACM International Conference on Management of Data (SIGMOD), pages 323–330, San Francisco, CA, USA, May 1987.

Cruz, I. F., Mendelzon, A. O., and Wood, P. T.: G+: Recursive queries without recursion. In Conference on Expert Database Systems (EDS), pages 645–666, Vienna, VA, USA, April 1988.

Cruz, I. F. and Norvell, T. S.: Aggregative closure: An extension of transitive closure. In International Conference on Data Engineering (ICDE), pages 384–391, Los Angeles, California, USA, February 1989.

Dar, S. and Agrawal, R.: Extending sql with generalized transitive closure. IEEE Transactions on Knowledge and Data Engineering (TKDE), 5(5):799–812, 1993.

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

Dillenburg, J., Wolfson, O., and Nelson, P.: The intelligent travel assistant. In IEEE 5th Anual Conference on Intelligent Transportation Systems (ITS), pages 691– 696, Singapore, September 2002.

Ding, Z. and Güting, R.: Modeling temporally variable transportation networks. In Database Systems for Advanced Applications, eds. Y. Lee, J. Li, K.-Y. Whang, and D. Lee, volume 2973 of Lecture Notes in Computer Science, pages 651–724. Springer Berlin / Heidelberg, 2004.

Erwig, M. and Schneider, M.: STQL: A spatio-temporal query language. In Mining Spatio-Temporal Information Systems, eds. R. Ladner, K. Shaw, and M. Abdelguerfi, chapter 6, pages 105–126. Kluwer Academic Publishers, 2002.

Erwig, M. and Schneider, M.: Spatio-temporal predicates. IEEE Transactions on Knowledge and Data Engineering (TKDE), 14(4):881–901, 2002.

Fellbaum: WordNet: An Electronic Lexical Database. The MIT Press, 1998.

Fleischman, M. and Hovy, E.: Taking advantage of the situation: non-linguistic context for natural language interfaces to interactive virtual environments. In 11th international conference on Intelligent user interfaces (IUI), pages 47–54, New York, NY, USA, 2006. ACM.

Fonseca, F., Egenhofer, M., Jr., C. D., and Borges, K.: Ontologies and knowledge sharing in urban gis. Computers, Environment and Urban Systems, 24(3):251 – 272, 2000.

Garcia, V. L., Motta, E., and Uren, V.: AquaLog: An ontology-driven question answering system to interface the semantic web. In North American Chapter of the Association for Computational Linguistics Conference on Human Language Technology (NAACL-HLT), pages 269–272, New York, NY, USA, 2006.

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

George, B. and Shekhar, S.: Time aggregated graphs for modeling spatio-temporal network. Journal on Data Semantics XI, pages 191–212, 2008.

Geutner, P., Steffens, F., and Manstetten, D.: Design of the VICO spoken dialogue system: Evaluation of user expectations by Wizard-of-Oz experiments: A speech driven in-car assistance system. In International Conference on Language Resources and Evaluation (LREC), Canary Islands, Spain, June 2002.

Google Maps: http://maps.google.com/, 2011.

Google Places API: http://code.google.com/apis/maps/documentation/places/, 2011.

Grasso, M. A., Ebert, D. S., and Finin, T. W.: The integrality of speech in multimodal interfaces. ACM Transactions on Human-Computer Interaction (TOCHI), 5(4):303–325, 1998.

Grice, H.: Logic and Conversation. In Syntax and Semantics 3. Speech Acts, eds. P. Cole and J. Morgan. Academic Press, 1975.

Gruenstein, A. and Seneff, S.: Releasing a multimodal dialogue system into the wild: User support mechanisms. In 8th ACL Workshop on Discourse and Dialogue (SIGDIAL), pages 111–119, Antwerp, Belgium, 2007.

Gruenstein, A.: City browser: A web-based multimodal interface to urban information. In ACL Workshop on Mobile Language Processing Demo (ACL-HLT), Columbus, OH, USA, June 2008.

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

Güting, R. H.: Graphdb: Modeling and querying graphs in databases. In 20th International Conference on Very Large Data Bases (VLDB), pages 297–308, Santiago de Chile, Chile, September 1994.

Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., and Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Transactions on Database Systems (TODS), 25(1):1–42, 2000.

Hemphill, C. T., Godfrey, J. J., and Doddington, G. R.: The atis spoken language systems pilot corpus. In DARPA Speech and Natural Language Workshop, pages 96–101. Morgan Kaufmann, June 1990.

eds. D. A. Hensher and K. J. Button Handbook of transport modelling, chapter 3. Emerald Group Publishing, 2nd edition, 2007.

Hirst, G.: Discourse-oriented anaphora resolution in natural language understanding: a review. Computational Linguistics, 7:85–98, April 1981.

Houda, M., Khemaja, M., Oliveira, K., and Abed, M.: A public transportation ontology to support user travel planning. In International Conference on Research Challanges in Information Science (RCIS), pages 127–136, Nice, France, May 2010.

Houtsma, M. and Apers, P.: Algebraic optimization of recursive queries. Data and Knowledge Engineering (DKE), 7(299–325), 1992.

John, B., Salvucci, D., Centgraf, P., and Prevas, K.: Integrating models and tools in the context of driving and in-vehicle devices. In International Conference on Cognitive Modeling, pages 130–135, Pittsburgh, PA, USA, July 2004.

Johnston, M., Bangalore, S., Vasireddy, G., Stent, A., Ehlen, P., Walker, M., Whittaker, S., and Maloor, P.: Match: an architecture for multimodal dialogue systems. In Annual Meeting on Association for Computational Linguistics (ACL), pages 376–383, Stroudsburg, PA, USA, 2002.

Jurafsky, D. and Martin, J. H.: Speech and Language Processing. Pearson Education – Prentice Hall, second edition, 2009.

Kaufmann, E., Bernstein, A., and Fischer, L.: NLP-Reduce: A “naïve” but domain-independent natural language interface for querying ontologies. In 4th European Semantic Web Conference (ESWC), Innsbruck, Austria, June 2007.

Kaufmann, E., Bernstein, A., and Zumstein, R.: Querix: A natural language interface to query ontologies based on clarification dialogs. In 5th International Semantic Web Conference (ISWC), pages pp. 980–981, Athens, GA, USA, 2006.

Kaufmann, E. and Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In 6th international The semantic web and 2nd Asian conference on Asian semantic web conference (ISWC/ASWC), pages 281–294, 2008.

Klein, D. and Manning, C. D.: Accurate unlexicalized parsing. In 41st Annual Meeting of the Association for Computational Linguistics (ACL), pages 423–430, Sapporo, Japan, July 2003.

Köhler, E., Langkau, K., and Skutella, M.: Time-expanded graphs for flow-dependent transit times. In 10th Annual European Symposium on Algorithms (ESA), volume 2461, pages 599–611, Rome, Italy, September 2002. Springer.

Kray, C. and Porzel, R.: Spatial cognition and natural language interfaces in mobile personal assistants. In European Conference on Artificial Intelligence Workshop on Artifical Intelligence in Mobile Systems (ECAI), Berlin, Germany, August 2000.

Lester, J. C. and Porter, B. W.: Developing and empirically evaluating robust explanation generators: the KNIGHT experiments. Computational Linguistics, 23(1):65–102, 1997. Special Issue on Empirical Studies in Discourse.

Li, Y., Yang, H., and Jagadish, H. V.: NaLIX: an interactive natural language interface for querying XML. In ACM International Conference on Management of Data (SIGMOD), pages 900–902, New York, NY, USA, 2005. ACM Press.

Lopez, V., Motta, E., and Uren, V.: PowerAqua: Fishing the semantic web. In 3rd European Semantic Web Conference (ECSW), pages 393–410, Budva, Montenegro, June 2006.

Lorenz, B., Ohlbach, H. J., and Yang, L.: Ontology of transportation networks. Technical report, REWERSE Project IST-2004-506779, 2005.

Lozano, A. and Storchi, G.: Shortest viable path algorithm in multimodal networks. Transportation Research Part A: Policy and Practice, 35(3):225–241, March 2001.

Lozano, A. and Storchi, G.: Shortest viable hyperpath in multimodal networks. Transportation Research Part B: Methodological, 36(10):853–874, 2002.

Mapquest: http://www.mapquest.com/, 2011.

Meng, F. and Chu, W.: Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation. Technical report, University of California at Los Angeles, Los Angeles, CA, USA, 1999.

Misu, T. and Kawahara, T.: Dialogue strategy to clarify user's queries for document retrieval system with speech interface. Speech Communication, 48(9):1137 – 1150, 2006.

Niaraki, A. S. and Kim, K.: Ontology based personalized route planning system using a multi-criteria decision making approach. Expert Systems with Applications, 36(2, Part 1):2250 – 2259, 2009.

Open Geospatial Consortium: http://www.opengeospatial.org/, 2011.

OpenCYC: http://www.opencyc.org/, 2011.

Ordonez, C.: Optimizing recursive queries in sql. In ACM International Conference on Management of Data (SIGMOD), pages 834–839, Baltimore, Maryland, USA, June 2005.

Pallottino, S. and Scutella, M. G.: Shortest path algorithms in transportation models: classical and innovative aspects. Technical report, University of Pisa, 1997.

Pelekis, N., Theodoulidis, B., Kopanakis, I., and Theodoridis, Y.: Literature review of spatio-temporal database models. Knowledge Engineering Review, 19(3):235–274, 2004.

POSTGIS: http://postgis.refractions.net/, 2011.

PostgreSQL: http://www.postgresql.org/, 2011.

Prud'hommeaux, E. and Seaborne, A.: SPARQL query language for RDF - W3C working draft, 2011.

Raux, A., Langner, B., Bohus, D., Black, A. W., and Eskenazi, M.: Let's go public! taking a spoken dialog system to the real world. In Interspeech, pages 885–888, Lisbon, Portugal, June 2005.

Regional Transit Authority Trip Planner: http://tripsweb.rtachicago.com/, 2011.

Roy, D.: Grounding words in perception and action: computational insights. Trends in Cognitive Science, 9(8):389–396, August 2005.

Salvucci, D.: A multitasking general executive for compound continuous tasks. Cognitive Science, 29:457–492, 2005.

San Martin, M. and Gutierrez, C.: Representing, querying and transforming social networks with rdf/sparql. In The Semantic Web: Research and Applications, eds. L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvonen, R. Mizoguchi, E. Oren, M. Sabou, and E. Simperl, volume 5554 of Lecture Notes in Computer Science, pages 293–307. Springer Berlin / Heidelberg, 2009.

Searle, J. R.: What is a Speech Act? In Philosophy in America, ed. M. Black, pages 615–628. Ithaca, New York, Cornell University Press, 1965.

Sheng, L. and Özsoyoglu, G.: A graph query language and its query processing. In International Conference on Data Engineering (ICDE), pages 572–581, Sydney, Australia, March 1999.

Sistla, A. P., Wolfson, O., Chamberlain, S., and Dao, S.: Modeling and querying moving objects. In International Conference on Data Engineering (ICDE), pages 422–432, Birmingham, U.K, April 1997.

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

Stonebraker, M., Rowe, L., and Hirohama, M.: The implementation of postgres. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2(1):125–142, March 1990.

Suggested Upper Merged Ontology: http://www.ontologyportal.org/, 2011.

The Open Source Geospatial Foundation: http://www.osgeo.org/, 2011.

United State Research and Innovative Technology Administration: ITS strategic research plan, 2010-2014: Executive summary. 2011.

Vazirgiannis, M. and Wolfson, O.: A spatiotemporal model and language for moving objects on road networks. In 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD), pages 20–35, Redondo Beach, California, USA, July 2001. Springer-Verlag.

Vilar, D., Xu, J., D'Haro, L. F., and Ney, H.: Error analysis of statistical machine translation output. In 5th International Conference on Language Resources and Evaluation (LREC), pages 697—702, Genoa, Italy, May 2006.

Walker, M. A., Litman, D. J., Kamm, C. A., and Abella, A.: PARADISE: A Framework for Evaluating Spoken Dialogue Agents. In Annual Meeting of the Association for Computational Linguistics (ACL), pages 271–280, Madrid, Spain, July 1997.

Wang, C., Xiong, M., Zhou, Q., and Yu, Y.: Panto: A portable natural language interface to ontologies. In European Semantic Web Conference (ESWC), pages 473–487, Innsbruck, Austria, June 2007.

Wang, J., Ding, Z., and Jiang, C.: An ontology-based public transport query system. In International Conference on Semantics, Knowledge and Grid (SKG), page 62, Beijing, China, November 2005.

Washington Metropolitan Area Transit Authority Trip Planner: http://www.wmata.com/tripplanner_d/
tripplanner.cfm, 2011.

Webber, B.: A Formal Approach to Discourse Anaphora. Doctoral dissertation, Harvard University, Cambridge, MA, USA, 1978.

Will, C. A.: Comparing human and machine performance for natural language information extraction: results for english microelectronics from the muc-5 evaluation. In 5th Conference on Message understanding (MUC), pages 53–67, Baltimore, MD, USA, August 1993. Association for Computational Linguistics.

World Wide Web Consortium: OWL web ontology language reference, 2011.

World Wide Web Consortium: RDF vocabulary description language 1.0: RDF schema, 2011.

Yannakakis, M.: Graph-theoretic methods in database theory. In ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pages 230–242, Nashville, TN, USA, April 1990. ACM Press.

Ziliaskopoulos, A. and Wardell, W.: An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. In European Journal of Operational Research, volume 125, pages 485–502, 2000.

Zografos, K. G. and Androutsopoulos, K. N.: Algorithms for itinerary planning in multimodal transportation networks. IEEE Transactions on Intelligent Transportation Systems (ITS), 9(1):175–184, March 2008.

VITA

EDUCATION

- **PhD, Computer Science**, University of Illinois at Chicago, Chicago, Illinois, 2011
- **BA, Computer Science**, Kalamazoo College, Kalamazoo, Michigan, 2005
  - Minor in Computational Mathematics

ACADEMIC DISTINCTIONS

- National Science Foundation IGERT Fellowship, 2007-2010
- University Fellowship, University of Illinois at Chicago, 2005-2006, 2010-2011
- Magna Cum Laude, Kalamazoo College, 2005
- Graduation with High Honors, Kalamazoo College, 2005
- Member of the Phi Beta Kappa Academic Honor Society , 2005
- Distinction on Comprehensive Exams, Kalamazoo College, 2004

RESEARCH EXPERIENCE

- **Doctoral Dissertation**, 2011
  - *Modeling and Querying Multimodal Urban Transportation Networks*
- **Undergraduate Thesis**, 2005
  - *Design Patterns: The Advanced Placement Computer Science Aquarium Lab Series as a Case Study*

TEACHING EXPERIENCE

- **Guest Lecturer**, University of Illinois at Chicago
  - Natural Language Processing
  - Semantic Web
- **Teaching Assistant**, University of Illinois at Chicago
  - Intellectual Property Law
  - Engineering Law
- **Teaching Assistant**, Kalamazoo College
  - Introduction to Programming
  - Introduction to Computer Science

PUBLICATIONS

- **Peer Reviewed**
  - Joel Booth, Barbara Di Eugenio, Isabel F. Cruz, Ouri Wolfson. Query Sentences as Semantic (Sub) Networks. The 3rd IEEE International Conference on Semantic Computing (ICSC). Berkeley, CA, USA. September, 2009.
  - Joel Booth, Prasad Sistla, Ouri Wolfson, Isabel F. Cruz. A Data Model for Trip Planning in Multimodal Transportation Systems. The 12th International Conference on Extending Database Technology (EDBT). Saint Petersburg, Russia. March, 2009.
  - Joel Booth, Barbara Di Eugenio, Isabel Cruz, Ouri Wolfson. Understanding Ambiguous Language in Context-Aware Mobile Querying. The Mobile Language Processing Workshop at The 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT). Columbus, OH. 2008.
- **Peer Reviewed Publication in Review/Preparation**
  - Joel Booth, Ouri Wolfson, Isabel F. Cruz, Prasad Sistla. Modeling Time-Dependent Multimodal Transportation Networks for Urban Environmentsd. VLDB Journal. [In Preparation]
  - Joel Booth, Barbara Di Eugenio, Isabel F. Cruz, Ouri Wolfson. A Robust Natural Language Interface for Urban Trip Planning. Journal of Natural Language Engineering. [Under Review]
- **Non-Reviewed Presentations**
  - A Data Model and Query Language for Urban Transportation Networks. The 5th Annual Midwest Database Research Symposium. Chicago, IL. October, 2008.
  - A Data Model for Trip Planning in Multimodal Transportation Networks. The 2nd Annual UIC Student Research Forum. Chicago, IL. April, 2009.

PROFESSIONAL SERVICE

- **Program Committee Member**
  - International Conference on Language Resources and Evaluation (LREC), 2012
  - IEEE Conference on Semantic Computing (ICSC), 2010
- **Adjunct Paper Reviewer**
  - ACM Conference on Information and Knowledge Management, 2011
  - ACM International Conference on Advances in Geographic Information Systems, 2007, 2008, 2011
  - ACM Special Interest Group on Management of Data, 2008
  - Databases, Information Systems and Peer-to-Peer Computing, 2007

- European Semantic Web Conference, 2007
- IEEE Consumer Communications and Networking Conference, 2010, 2011
- IEEE Intelligent Systems, 2007
- IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, 2008
- IEEE International Workshop on Vehicular Networking, 2008
- IEEE Transactions on Mobile Computing, 2008
- International Conference on Advanced Geographic Information Systems, Applications, and Services, 2011
- International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2011
- International Conference on Data Engineering, 2009, 2010, 2011
- International Conference on Mobile Data Management, 2009, 2010
- International Conference on Resource Intensive Applications and Services, 2011
- International Conference on Very Large Data Bases, 2010
- International ICST Conference on Mobile and Ubiquitous Systems, 2007
- International Symposium on Computer and Information Sciences, 2009
- International Symposium on Spatial and Temporal Databases, 2009, 2011
- International Symposium on Web and Wireless GIS, 2007, 2008
- International Workshop on Sensor Network Technologies for Information Explosion Era, 2008
- Workshop on Data and Services Management in Mobile Environments, 2008
- IEEE Globecom Multimedia Communications, Software and Services Symposium, 2007

UNIVERSITY SERVICE

- Council Member for UIC Graduate Student Council, 2009-2010
- General Council Member for UIC Computer Science Graduate Student Association, 2006-2011