Robust Natural Language Processing for Urban Trip Planning

Joel Booth[*]
Sprout Social
30 N Racine Ave #110, Chicago, IL, USA
joelbooth@gmail.com

Barbara Di Eugenio[†]
University of Illinois at Chicago
Department of Computer Science
851 S. Morgan (M/C 152), Chicago, IL, USA
bdieugen@uic.edu

Isabel F. Cruz
University of Illinois at Chicago
Department of Computer Science
851 S. Morgan (M/C 152), Chicago, IL, USA
ifc@cs.uic.edu

Ouri Wolfson
University of Illinois at Chicago
Department of Computer Science
851 S. Morgan (M/C 152), Chicago, IL, USA
wolfson@cs.uic.edu

June 17, 2014

Abstract

Navigating large, urban transportation networks is a complicated task. A user needs to negotiate the available modes of transportation, their schedules, and how they are interconnected. In this paper we present a Natural Language interface for trip planning in complex multimodal urban transportation networks. Our objective is to provide robust understanding of complex requests while giving the user flexibility in their language.

We designed TRANQUYL, a Transportation Query Language for trip planning; we developed a user-centric ontology, that defines the concepts covered by the interface and allows for a broad vocabulary. NL2TRANQUYL, the software system built on these foundations, translates English requests into formal TRANQUYL queries. Through a detailed intrinsic evaluation, we show that NL2TRANQUYL is highly accurate and robust with respect to paraphrasing of requests as well as handling fragmented or telegraphic requests.

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

---

[*] This work was conducted while the first author was a PhD student at the University of Illinois at Chicago, supported in part by NSF award DGE-549489.

[†] Corresponding author.

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 one of the easiest ways to find a new place or get directions is to simply ask someone. It is not surprising that more and more services that provide directions—whether online at home, on your car navigation system, or on your cell phone—are available to users that need to go somewhere. While these systems are useful they all lack one or more useful features. This task is frequently more complicated than simply finding a path from Point A to Point B, made familiar by interactive map-based services such as Google Maps and GPS units. These systems know very little (if anything) about you—the user—and therefore cannot make use of personal information. Additionally, the interfaces tend to be map or menu driven which may not be appropriate in many circumstances (e.g., while driving or walking). In this paper we will present a robust natural language system that addresses these issues and more, including specifying intermediate stops at generic or specific facilities (*a florist, my doctor*), and expressing degrees of certainty.

Intelligent Transportation Systems such as those supported by the Intellidrive initiative of the US Department of Transportation (United States Research and Innovative Technology Administration, 2011) address transportation problems ranging from real-time routing and navigation, to autonomous driving, to inferring driving patterns via data mining. A large group of researchers to which we belong is engaged in developing an Intelligent Traveler Assistant (ITA)[*] (Dillenburg *et al.*, 2002; Zhong *et al.*, 2008), which will run on handheld computers and will be networked to a traffic information center that can plan multi-modal routes for users (note that here *multi-modal* refers to multiple modes of transportation, such as driving, walking, using public transportation, as opposed to multiple modes of input to, and output from, a computer, such as voice or touch).

In this paper, we present NL2TRANQUYL, the module of our ITA that allows users to ask for directions using Natural Language (NL). Especially when faced with a large multimodal transportation system, knowing how to best get around a city is a challenge. A person must understand the various train and bus routes, schedules, fare structures, and even how those systems interact with the pedestrian and automobile networks. Depending on the circumstances, a person may wish to optimize their trip in different ways: taking the least time, traveling the shortest distance, or choosing the least expensive route. Additionally, the

---

[*] For more information on the research group, visit the Computational Transportation Science page at http://cts.cs.uic.edu/.

system should understand references specific to that user and their current context. For the remainder of this paper, we consider a “trip” to be an itinerary, or set of instructions, that satisfy a user’s needs when traveling through the transportation network.

The following requests illustrate some of these requirements (note that all numbered example requests throughout the paper were used in the formal evaluation). Note that to minimize confusion, we use the term *request* when referring to the English question or command, and the term *query* to refer to the formal query in the TRANQUYL language.

(1) *Arriving after 6:00pm, find the cheapest route to my apartment.*

(2) *Which transit route home has the fewest transfers and allows me to visit a grocery store?*

(3) *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?*

One question arises, why use NL in such a system, since the state of the art in speech and language processing does not allow for full, unconstrained language interactions yet. However, whereas 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. Further, various studies (Cohen, 1992; Grasso *et al.*, 1998; Kaufmann & Bernstein, 2008) show that NL is typically better at expressing certain questions/commands, such as those requiring negation, quantification, and temporal information; additionally, using NL typically reduces task completion time.

Apart from the general area of interfaces to transportation systems, there has been recent interest in using NLIs to semantic web applications or databases. We will discuss related literature more in detail later in the paper, but here we note that for example (Kaufmann & Bernstein, 2008) found that NLIs were effective in helping non-expert users access information in semantic web databases. Interestingly, they determined that users appreciated the power of fully grammatical sentences more than sentence fragments or keywords.

For all of these reasons, we decided to explore NL solutions to the trip planning problem. Our system, NL2TRANQUYL, is capable of processing complex requests with a robust vocabulary and very loose grammatical constraints. We feel that this emulates the most straightforward method of trip planning—simply asking someone for help. We emphasize robustness in allowing for sentence fragments and poor grammar, as may arise in natural conversation. Additionally, we utilize a user-centric ontology to infer the meaning of personal references and preferences so that they do not need to be explicitly stated for each request. Naturally, our system builds on a Transportation Query Language, TRANQUYL, that we had to design, since no existing transportation query language was broad enough to capture our needs.

In the remainder of the introduction we provide a background of what makes urban transportation networks a unique and interesting domain to address. We will also discuss some general requirements that arise for the kind of language a NLI to an ITA needs to support.

1.1 The Multimodal Urban Transportation Network

Multimodal urban transportation networks provide new challenges for both data modeling and Natural Language processing. The urban network is comprised of multiple different modes of transportation (e.g., bus, train, automobile, pedestrian) that interconnect and serve users and facilities (e.g., stores, hospitals). All of these entities have both spatial and temporal dimensions that may change over time (e.g., hours of operation of a store, the current position of a user, the status of a road at a specific time). We also know that there is a great deal of uncertainty in the system: busses do not always run on time and roads become congested. For these reasons, we have found that the chosen domain provides new research challenges.

At this time, we would like to point out some key differences to what is 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 not only on your destination, but also on 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.

This richer problem domain gives us an opportunity to process more elaborate requests that have features not present in earlier systems. Also, while many previous systems rely on a dialogue to collect enough information to form a query, we restrict ourselves to a single input sentence. All of the query constraints must be inferred from the input sentence and our background knowledge of the user. We show that even without a chance to fix the input we are able to achieve high accuracy.

1.2 Natural Language Considerations

Before discussing the software implementation, it is important to understand the scope of natural language we are 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, we 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 & Seneff, 2007; Gruenstein, 2008), or restaurants, including how to reach them (Bangalore & Johnston, 2009). The Let's Go corpus (Raux *et al.*, 2005; Black *et al.*, 2011) 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 we discussed above, neither ATIS nor any of the other corpora include trip planning requests of the sort we are interested in, comprising the complex constraints we discussed earlier. Hence, we set out to collect a new corpus of urban trip planning requests.

In order to guide our focus we solicited requests 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 (instructions are included in Appendix A). The responses ranged from nearly trivial (e.g., *get me home*) to complex planning situations expressed via multiple sentences that required getting multiple users to a single destination—while constraining the arrangement to an entire paragraph of additional details. Half were requests for trip plans with a few different constraints, and we decided to focus on those. The language used ranged from fully grammatical sentences, as in Example (4a), to short fragments, as in Example (4b).

(4a) *Leaving from 1000 W. Taylor St., what is the fastest way to my house for less than $7?*

(4b) *train home by 7:00pm*

This collection of requests showed us that NL2TRANQUYL should allow language that is not grammatically trivial. The main clause can be a command, a question or a fragment. The request can include coordinate and subordinate clauses, specifically preposed gerundive adjuncts and relative clauses. Example (5b) below includes all three types of clauses and is successfully processed by NL2TRANQUYL. We note that the relative clause in turn includes two conjoined clauses, as well as negation in the first conjunct. As far as Noun Phrases (NPs) are concerned, NPs can be conjoined, modified by relative clauses, include numbers, or comparative and superlative expressions. Example (5a), which is also successfully processed by NL2TRANQUYL, includes a complex NP whose head is *route*, which is modified by two relative clauses; one other NP includes three conjuncts: *a bank, grocery, and bar*, and one other NP includes a comparative: *fewer than 7 transfers*. Prepositional Phrases (PPs) are also used and processed successfully, although they create some problems when the main clause is a fragment, as we will discuss in Section 5.5.

(5a) *Is there [*$_{\texttt{NP}}$ *a transit route to my home [*$_{\texttt{rel-clause}}$ *that stops at [*$_{\texttt{NP}}$ *a bank, grocery, and bar]] [*$_{\texttt{rel-clause}}$ *that arrives before 5:00pm and has [*$_{\texttt{NP}}$ *fewer than 7 transfers]]]?*

(5b) *[*$_{\texttt{ger-adjunct}}$*Taking the least expensive path to my apartment,] does there exist one [*$_{\texttt{rel-clause}}$ *that doesn't cross a river and still arrives no later than 7:00pm with 99% certainty on that time]?*

In addition to the language that was of interest to the users, the biggest constraint was the expressive power of TRANQUYL. The language was designed for trip planning, therefore requests beyond that scope were not of interest for this work—all of the requests handled are either explicitly or implicitly asking for a trip plan. Some requests that ask for different types of information, for example *find the nearest pharmacy*,[*] are looking for a location, but a trip to that location implicitly answers the question of where the place is. In the evaluation, we will show that these implicit requests are successfully processed by NL2TRANQUYL.

Beyond the fact that all requests must be looking for trips, it is important to understand what vocabulary can be used and what types of trip requests 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 3.2. Some of the concepts in the ontology are purely for organization purposes and not directly usable in a request. 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 & 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 requests to be described in Section 5.

2 Related Work

NLIs have a long history in the NLP and databases communities. Our 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 interest in NLI as applied to trip planning is also prompted by the proliferation of car and portable navigation units. People are unable to type or use a mouse while driving (or certainly not allowed to), 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 (Rosé & Lavie, 2000; Li *et al.*, 2005), clarification dialogs (Kaufmann *et al.*, 2006), the use of WordNet to boost vocabulary (Moldovan *et al.*, 2007; Lopez *et al.*, 2006), and statistical methods to improve disambiguation of ambiguous terms (Meng & Chu, 1999). NLIs have been increasingly applied to ontologies

---

[*] This request is one of the 12 implicit requests evaluated separately from the main evaluation corpus, see Section 5.4.

(Wang *et al.*, 2007; Bernstein *et al.*, 2005; Dzikovska *et al.*, 2008), especially in the domain of question answering (Garcia *et al.*, 2006; Lopez *et al.*, 2006; Moldovan *et al.*, 2007; Ferrández *et al.*, 2011). In some cases, only shallow NLP techniques are used and partially formed requests (i.e., sentence fragments) are allowed (Kaufmann *et al.*, 2007). This approach relies on keywords alone rather than grammatical relations. When more sophisticated approaches are proposed like in the QALL-ME framework (Ferrández *et al.*, 2011), the system identifies NL question patterns and associates them with DB query patterns; it then identifies a minimal subset of all the defined mappings. For a new question, it first transforms it into a pattern, then uses a textual entailment engine to map the discovered pattern into one of those contained in the minimal subset, and hence, onto a DB query. Needless to say, if the user question does not fall into one of the predefined patterns, the system will fail to return an answer, which happens in 9.73% of the user questions in the evaluation presented in (Ferrández *et al.*, 2011) (another 10.62% of failures are attributed to the entailment engine failing to finding the correct entailed patterns in the minimal set).

A different take on NLI's comes from the recent impetus on *Personal Assistants (PAs)* deployed on mobile platforms, fueled by the success of Apple Siri and Google Voice (Neustein & Markowitz, 2013). (Meisel, 2013) notes that PAs can attempt at assisting the user in every realm of life, as Siri attempts to do with mixed success, or be composed of specialized subcomponents, such as an ITA like NL2TRANQUYL. Some specialized PAs concerning urban navigation had been devised for older mobile hardware, such as MATCH for the Fujitsu PDA (Johnston *et al.*, 2002) and City Browser (Gruenstein & Seneff, 2007) for a web interface accessible on mobile devices. These two projects provide multimodal (in terms of input) navigation of urban areas, however they largely dealt with finding resources rather than planning trips or itineraries.

(Meisel, 2013) notes that recognizing speech processing is only one crucial component of a PA, with the NLP component being equally crucial. Hence, in many ways, NL2TRANQUYL is situated between the aforementioned works, be they NLI's to static applications, or PAs. We focus on the NLP component, and use WordNet as a straightforward method of expanding the vocabulary; additionally, we allow for other word forms (i.e., colloquial annotations) to be captured in the TRANQUYL ontology. The use of the ontology resource in our manner is unique in the literature we 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 as an additional lexicon.

Similarly, our system functions with fully grammatical sentences as well as some ungrammatical sentences and fragments. This is because we make use of the constituency parse only as concerns nodes, and hence concepts, of interest. The relations between concepts are inferred as based on the dependency parse, which is generally more stable across changes in syntax. Another difference from previous work is that 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.

The work by Dzivoska et al (Dzikovska *et al.*, 2008) is particularly relevant to our efforts. It presents a systematic approach to linking lexica and ontologies to port dialogue systems to new domains. Specifically, they define formal mapping rules that link the lexicon, to what they call the LF ontology, used in parsing and semantic interpretation, to the KR ontology that represents the domain. Additionally, they describe computational methods that take those mapping rules and propagate their coverage as appropriate. They show that the developer effort in porting a parser from one domain (e.g., the logistics of moving cargo for a planning system) to another (e.g., basic electronics and electricity for a tutoring domain) is much reduced, but that the parser still performs very well. Our work differs from theirs in that, to the extent possible, we focused on reusing available NLP tools such as the Stanford parser (as far as we know, the parser by (Dzikovska *et al.*, 2008) was never made available). If we were to port our work to a different domain, we would make use of their ideas to support the mapping (and their software, if it were to become available).

Whereas the mapping rules in (Dzikovska *et al.*, 2008) are written by hand, much recent work focuses on learning semantic representations. Some of this recent work connects semantics and the world state, through either perceptual or action execution contexts, e.g. (Zettlemoyer & Collins, 2009; Liang *et al.*, 2009; Chen *et al.*, 2010; Branavan *et al.*, 2010; Branavan *et al.*, 2012a; Branavan *et al.*, 2012b; Liang *et al.*, 2013). The work closest to ours (Thompson & Mooney, 2003; Kate & Mooney, 2007; Muresan, 2008) applies learning of semantics to language interfaces to databases, or to question-answering applications. The evaluation on the GEOQUERY application from (Thompson & Mooney, 2003; Kate & Mooney, 2007) is similar in size to ours, since they evaluate their system on 250 requests (our formal evaluation, described later in the paper, consists of 162 requests; an additional 43 requests were informally evaluated). The biggest difference between our work and previous work on learning semantic representations is that we had to define the semantic representation itself, whereas they start from a pre-existing one. For example in (Thompson & Mooney, 2003), requests are paired with their Prolog representations; in (Muresan, 2008), examples are annotated with domain-independent semantic interpretations (the application is medical question-answering). Even if later work by these research groups has moved away from supervised approaches given how costly it is to develop the training data, their foundations remain in precisely specified representations. We did not have a predefined language: we had to invent the query language itself, TRANQUYL. Hence, it is foreseeable that in future work, once TRANQUYL has been further validated in the larger project, we could investigate learning semantics from either an annotated corpus of language requests and their TRANQUYL representations, or in a semi-supervised or unsupervised fashion from e.g. language requests, their answers and partial traces of how the TRANQUYL queries were built.

Other related work concerns transportation applications proper, such as those studying the best type of interface between drivers and their intelligent vehicles, e.g., the best combination of input-output modalities (Bernsen & Dybkjær, 2002; Buehler *et al.*, 2003; Coletti *et al.*, 2003; Geutner *et al.*, 2002; John *et al.*, 2004; Salvucci, 2005). Of these, the closest to our work describe the VICO project (Coletti *et al.*, 2003; Geutner *et al.*, 2002), aimed at developing a car assistance system for accessing tourist information databases and obtaining driving assistance. However they focused on understanding speech rather than on understanding complex requests, which are in turn translated into a query language. The Let's Go project (Raux *et al.*, 2005; Black *et al.*, 2011) is a call-in 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.

3 Models and Resources

We start the presentation of our work by briefly discussing the TRANQUYL DataBase Language, and the attendant Query Language; and the transportation ontology we developed. Developing a new DataBase Language was a crucial component of our effort, but is not a focus of this paper; further details on TRANQUYL can be found in (Booth *et al.*, 2009a; Booth, 2011).

3.1 The TRANQUYL DataBase Language

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

For each $m \in M$ we define a set $edge\_attributes_m \subseteq L$ that describe attributes of the edges in the graph of that *mode*. Different modes have different attributes. For example, *run-id* (a numeric identifier of each run of a bus) is an attribute of the bus mode, but is irrelevant for the pedestrian mode. Each edge has values for the attributes specified by the *mode* of the edge. Similarly, we define a set of attributes $vertex\_attributes \subseteq L$ that describe the attributes of vertices. Vertices are not associated with a mode, therefore all vertices have the same attributes $vertex\_attributes = \{name, geometry, facilities\}$. The *name* represents some real-world name of the vertex. For example, it may be the name of the intersection of streets, the name of the train station, etc. The *name* value may be *null* if there is no appropriate name available. The *geometry* of a vertex represents its real-world geometry (e.g., the x,y coordinate on a map[*]). Finally, *facilities* represents the set, $f \subseteq F$, of classes of facilities (e.g., grocery store, restaurant, bank) present at the vertex. We assume that the facility is available at the given point in time if it is present in the set.

We define the graph as $G = (V, E, \Psi)$ where V is the set of labeled vertices and E is a set of labeled edges. Each edge is defined as a 4-tuple (v_1, v_2, m, ϕ_m) where v_1, v_2 are the endpoints of the edge, m is the *mode* label of the edge, and ϕ_m is a function that maps the attributes defined for the mode to values. Two important attributes of edges are *mean speed*, and *speed variance*. Using these, TRANQUYL enables the specification of uncertainty in the language.

For each vertex v and vertex attribute x, Ψ specifies the value of attribute x for vertex v. Note that this defines a single unified graph. Edges of multiple modes may be incident on the same vertex, and in fact this is how the transfer between modes is modeled.

Figure 1 contains a simplified graph representation 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.

[Figure 1 about here.]

---

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

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

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

3.1.1 Querying for Trips: The TRANQUYL Query Language

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

First, to query trips we introduce an operator

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

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

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

`WITH MODES` A list of the modes to be allowed in the trip (see the set M of five modes mentioned in the previous section).

`WITH [ORDERED] STOP_VERTICES` A set of vertices (that may be ordered) to be included in the trip; they will be used to specify intermediate stops.

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

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

With these new clauses we define a generic query structure, which allows for the full description of trips in an urban transportation system in a relational-like syntax:

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

The following is one example of a NL request and its TRANQUYL equivalent, as computed by NL2TRANQUYL:

(6) *Get a route to my office that stops at a florist and my doctor using transit*

```
SELECT * FROM
  ALL_TRIPS(user.current_location, user.office) AS t
    WITH MODES bus, rail, pedestrian
    WITH STOP_VERTICES s0, s1
    WHERE "Florist" IN s0.facilities
      AND s1 = user.doctor
    MINIMIZE DURATION(t)
```

The request is looking for a trip (a.k.a. a route) between two locations as represented by the current location, and the user's office, the latter as specified in the user profile. The query specifies that the bus, rail and pedestrian modes are allowed (that is how *transit* is defined in the ontology); and that the path of the trip will be minimized with respect to its duration (this is a default for the current speaker). Furthermore, it requires that a florist and the user's doctor be included at some point along the way—specified by the `Florist IN s0.facilities` and `s1 = user.doctor` statements. The variable `s0`, being a node, has an attribute that lists the facilities it possesses (i.e., the types of resources a user may find interesting: banks, grocery stores, pharmacies, as discussed in the previous section); on the other hand, the variable `s1` is equated with `user.doctor` (also specified in the user profile) since the speaker is requesting a stop at his/her doctor's, not at any doctor.

We will provide further details on how a TRANQUYL query is built in Section 4.3.

3.2 The TRANQUYL Ontology

We developed an ontology as a way to model and store relevant information for our system. On one side we have a database schema that our output query must follow, and on the other we have a naive user who wishes to issue a request against said database without understanding the underlying schema. We use an ontology to bridge the representational gap between the user and database.

The ontology models the user and the concepts found in TRANQUYL, 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. A small subset of the 80 concept hierarchy is shown in Figure 2. 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 the user profile contains, the less information the user must explicitly state for any given request.

[Figure 2 about here.]

We chose to develop our own ontology since existing ontologies did not fit our specific needs. High level ontologies such as SUMO (Suggested Upper Merged Ontology, 2011), OpenCyc (OpenCYC, 2011), or REWERSE (Brunner *et al.*, 2006) contain some information about geography and/or transportation systems, but the focus of these ontologies is general knowledge and hence, they are much too broad for our purposes. A few transportation specific ontologies exist, but most of them focus too heavily on the mechanics on the transportation network (Fonseca *et al.*, 2000; Lorenz *et al.*, 2005; Benslimane *et al.*, 2000; Teller *et al.*, 2007). Our 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 interacts with the road network and shipping ports is far beyond our scope, and would only serve to introduce noise into the model. Very few ontologies focus on personalized route planning systems. One was proposed by (Niaraki & Kim, 2009). However, their 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 interactions with them. Closer to our work, (Wang *et al.*, 2005) proposes an ontology focusing on the urban public transport system, and their query algorithm takes into account user constraints such as transfer times, walking distance, and price. Clearly, this ontology is missing crucial components of the urban network. (Mnasser *et al.*, 2010; Marçal de Oliveira *et al.*, 2013) developed an ontology to support user travel planning. Like ours, their ontology contains information about route selection preferences and potential resources in the network.

However, none of these ontologies were designed to facilitate the use of natural language.

To summarize, some of these preexisting models were too focused on the minutiae of the transportation network, whereas we are interested in only the highest level concepts in the ontology and how a user interacts with them. Those very few that focus on user trip planning, are not concerned with supporting the usage of natural language. Hence, we chose to develop our own ontology from the ground up, which ties in some spatial and temporal attributes and relationships missing from the transportation specific ontologies. Furthermore, we also use the ontology 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 annotated with additional lexical information. For example, the concept *train* is labeled “train” in the ontology. Locally, our urban trains are also known as “The L” (because most of these trains run ELevated); therefore we include a colloquial label denoting such. We also annotate the concept with the relevant WordNet entry (with root word and sense key): *train%1:06:00*. How this information is used will be discussed in the next section. The small size of the ontology meant that manual annotation was feasible and likely faster than implementing an automatic method.

At this time we do not use a reasoner in conjunction with the ontology. We are using it as a knowledge model to guide the translation and as a lexicon (via the lexical annotations) to improve the scope of language understood. Currently, exploiting the more advanced aspects of ontologies and the semantic web is beyond the scope of our research.

4 Translating Natural Language to TRANQUYL

In this section we present the NL2TRANQUYL system. The translation from natural language to TRANQUYL occurs in several distinct steps, as shown visually in Figure 3. The four steps are called ‘filters’ as they attempt to remove noise and remediate the lack of information. This 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 three subsections.

[Figure 3 about here.]

4.1 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 & Manning, 2003) in order to obtain both constituency and dependency parses. For illustrative purposes, example parses for the example request (7) are presented in Figures 4 and 5. This request is one of the simplest that NL2TRANQUYL successfully processes, but we use it in this paper for ease of presentation. We will discuss more complex requests in Section 5.

(7) *Can I walk to 300 W. Humboldt Blvd. by 4:00pm?*[*]

[Figure 4 about here.]

[Figure 5 about here.]

The algorithm in Figure 6 shows the high-level pseudocode for processing an input request. The set N is formed by selecting the relevant nodes among all those found in the constituency parse tree (cp). The relevant nodes are defined as adjectives, adverbs, verbs, nouns, prepositions, and simple phrases (adjective phrases like *least expensive*, compound nouns like *grocery store*, addresses) —concepts that may exist in the ontology. In order to determine which concept in the ontology a specific relevant node $n \in N$ corresponds to, two approaches are taken: a) comparing nodes to concepts in the ontology (**getOntologyConcepts**), and b) identifying specific concepts via regular expressions (**getRegExConcepts**). Note that the dependency parse dep is not used at this point; it will be used later, when concept attachment is addressed (see Section 4.2).

[Figure 6 about here.]

The first algorithm to extract concepts, **getOntologyConcepts**. performs a pairwise matching between each node n and all of the concepts c in the ontology (please see the algorithm in Figure 6).

[Figure 7 about here.]

---

[*] While this sentence may show 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 300 W. Humboldt Blvd., such that the user 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.

getOntologyConcepts uses a set of metrics that evaluate two types of matching. The first type of match (Table 1) concerns the text string contained by the node n and the concept and colloquial labels associated with the concept c; in turn, this type of match is evaluated via three kinds of metric: a direct string match, stemmed match, and match-within-1 edit. The second type of match (Table 2) is between the WordNet label associated with the concept c and the look-up of the lemma associated with the node n in the WordNet lexical tree (similarly to (Budanitsky & Hirst, 2006)). Each match type / metric is associated with a score, and 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 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. The minimum threshold value was determined to be .24.

Match types and their corresponding weights are found in Tables 1 and 2. Table 2 refers to how the WordNet entry located via look-up on node n relates to the WordNet annotation on the concept c in the ontology. For example, suppose our request asks *Are there any trains home after 12am?*. For the candidate concept `train` in our ontology, the associated WordNet entry is *train%1:06:00*. For the WordNet look-up, we will index WordNet twice: simply via the lemma *train*, or via the lemma with its POS tag, NN in this case. We then assess the relationship between each of the synsets returned by WordNet and the WordNet annotation for the concept c. Since there are multiple senses for some words (e.g., for *train*, WordNet returns 17 different senses) using the part of speech of the lemma (as returned by the Stanford parser) allows for a better match: e.g. for *train*, only 6 senses are returned for NN, one of which is the sense annotated on the concept `train` in the ontology.

[Table 1 about here.]

[Table 2 about here.]

The second algorithm to extract concepts, **getRegExConcepts** extracts several other concepts that are instances of concepts contained within the ontology, specifically: instances of *times, addresses*, and other numeric values such as *costs* (e.g., \$3.50) and *certainty* specifications (e.g., 50%). These concepts are identified using regular expressions, and while specific instances are 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*) (we don't include getRegExConcepts' pseudo-code since it is very simple).

The complexity of the Concept Identification component is dominated by the **getOntologyConcepts** algorithm. It is $O(n * c)$ where n is the number of nodes extracted from the parse tree and c, the number of concepts in the ontology. Since both n and c are below 100, this is not problematic. Because n depends on the size of the parse tree which in turn depends on the length and complexity of the input requests, we do not foresee n increasing substantially, since our inputs are already complex enough (see Section 5). The situation is different for c, the number of concepts. As the ITA within which our work is situated gets further developed, it will assist users with different types of requests (e.g., tourist information); the ontology will then obviously need to be expanded. Better indexing and search techniques for the ontology will then become essential (Sharman *et al.*, 2007; Staab & Studer, 2009). However, these are beyond the scope of the current work.

Continuing with the example request of *Can I walk to 300 W. Humboldt Blvd. by 4:00pm?*, the system will identify two types of concepts in the ontology: *pedestrian* as a mode of transportation, and *by* as a temporal relationship; and two instances of concepts in the ontology, *300 W. Humboldt Blvd.* and *4:00pm*. Note that the concept labels do not exactly match their representation in the input sentence, which attests to the generality of the mapping. On the other hand, the ontology concept match criteria were selected based on the information contained in the model. The ontology only provides the following information: concept label, colloquial annotation, and WordNet annotation. It was imperative that the most information possible be drawn from these labels.

4.2 Concept Attachment: Knowledge Map Generation

After we have identified the concepts in the request, it is important to determine how they are related. We use the dependency parse (*dep* in Figure 6) and three sets of strategies, guided by the ontology, to generate a knowledge map. This map is effectively a subgraph of the ontology. The previous step gave us the subset of concepts in the ontology that we are concerned with, and in this step we determine how they are interrelated. Before describing the procedure, we present a example knowledge map in order to illustrate our objective. Once again we use the request *Can I walk to 300 W. Humboldt Blvd. by 4:00pm?*.

[Figure 8 about here.]

Each request 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 request are the four instances (rounded rectangles) in the graph. The explicit relationships in the ontology tell us how the instances may be related. In the remainder of this section we briefly discuss how we build the knowledge map and in the following section we discuss how it is used to generate the final TRANQUYL query.

In generating the knowledge map, our first sources of information are the parses for the input request. 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 request. In our experience, the dependency parse is much more stable across sentence formulations. Unfortunately, we may not receive well formed input, so we do not rely on parses alone. If the parse trees do not provide sufficient information, we resort to vicinity in the input request.

The high level pseudo-code for the knowledge map generation is found in Figure 9. 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 9 about here.]

Identifying personal references. Recall that the requests NL2TRANQUYL is able to process at the moment only concern the user who is issuing the request. Whereas this vastly simplifies processing referring expressions, we still need to identify whether the user is using personal references or not – such as *my bank*, *the bank* or *a bank*. If a possessive is explicitly present in the input request, it will be directly identified in the dependency parse, which 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 requests where an explicit statement of possession has not been made, the ontology will be consulted. Based on knowledge of referring expressions (Webber, 1978; Hirst, 1981; Prince, 1981), definite noun phrases such as *the bank* are likely to refer to resources specific to the user;[*] indefinite noun phrases such as *a bank* or *any bank* will instead be satisfied by any bank. This is implemented by the first FOR loop

---

[*] NL2TRANQUYL does not conduct a dialogue with its users yet, and we do not maintain any model of discourse entities; hence, the function of definite NPs as referring to entities previously mentioned in the dialogue is not accounted for.

in Figure 9. If a resource is specified, either via a possessive or a definite description, we check whether it is of type *Personal*: in the ontology, the concept ”personal” of which many concepts inherit, is used to denote that some term or object can be a personal reference, such as a *bank* or a *florist* (as opposed to a *train*). In this case, the user's profile is checked to see whether they have a preferred instance of that resource. If so, the generic concept is replaced with the specific instance. The generic concept will be kept if it is not a definite or a possessive, or if no specific instance is found. This may happen even with an explicit possessive if e.g., a user did not complete their profile – namely the user says *my bank* but no bank is specified in the user profile.

Modifier alignment. Next, we determine the correct modifier alignment (the two FOR loops in the middle of Figure 9). We must understand which concepts are modified by other concepts. 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, we use the modifier relations returned by the dependency parser. If not, we map back from concept c to the node n that was matched to it, we expand a window w around node n in the original sentence, gather the corresponding concepts c_w, and choose the concept $c_{w[i]}$ that is closest to c, if it exists.

Missing information. Finally, we look for what we don't know – constraints and values that are missing from the request, but are required to build a proper query in this domain (bottom of Figure 9). We know that we are looking for a trip, so we check the ontology and determine which of the required concepts associated with a trip have not been found yet (essentially, origin and destination). Similarly, if we have identified concepts that are not associated with any other, we can determine which concept in the ontology could anchor them. After the missing components are identified (if possible), the knowledge map is passed onto the query generation step.

4.3 Query Generation

The final step is generating the TRANQUYL query using the knowledge map. We begin with the generic TRANQUYL query structure, discussed in Section 3.1.1 and fill in the appropriate values. To continue with our illustrative example, the TRANQUYL query for our running example, *Can I walk to 300 W. Humboldt Blvd. by 4:00pm?* is as follows,

```
SELECT * FROM
  ALL_TRIPS(user.current_location, 300 W. Humboldt Blvd.) AS t
    WITH MODES pedestrian
    WITH CERTAINTY .78
    WHERE ENDS(t) <= 4:00pm
    MINIMIZE DURATION(t)
```

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 300 W. Humboldt Blvd. 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 request.

HERE!!!

The minimum requirement for a well-formed TRANQUYL query is that it contains a SELECT clause with a call to the ALL_TRIPS operator, and a WITH MODES clause (possibly filled with a default value). The criteria for generating each of the required and optional clauses are as follows:

SELECT Fill in the origin and destination in the ALL_TRIPS operator using the values found in the knowledge map (see Figure 8, the *Origin* and *Destination* attributes of the *Trip* concept). If the origin is unspecified, use the user's current location as the origin.

WITH MODES List the modes found in the request, `pedestrian` in the example above. For the earlier example (6) whose corresponding TRANQUYL query was presented in Section 3.1.1, *transit* was expanded to the appropriate modes it includes. If no modes are listed, use the preference in the user profile.

WITH STOP_VERTICES If an intermediate node is required (e.g., in the case of visiting a facility), add the variable and pass the variable name onto the WHERE clause generator. For Example (6), two variables are generated `s0, s1`.

WITH CERTAINTY 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 request or the user's profile, as in the example above. By design, only temporal constraints are assumed to be subject to uncertainty. Namely, spatial constraints are guaranteed to be satisfied (i.e., probability of 1.0), hence, this clause is included only if an explicit temporal constraint is included.

WHERE Include all of the spatial, temporal, transfer, cost, facility, and other constraints that have been specified and are included in the knowledge map. If a facility is a personal reference, use the appropriate value from the user's profile. For the example above, a temporal constraint is added. For Example (6), two intermediate stops are specified, with one (the doctor) being read off the user's profile.

OPTIMIZE If a trip optimization metric is found in the knowledge map generate the appropriate statement, else use the default from the user profile – `DURATION` in our example.

We have shown examples of temporal and resource constraints on trips, but there are a few more to mention in detail. The first are constraints (inclusive and exclusive) on geographic regions. The user can issue requests such as

(8) *I want to walk to home through a park*

where a park is of type *spatial extent* in the database. In Section 5, we will show the TRANQUYL query for a request that includes one such constraint. 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).

While it is unlikely to be used much in practice, we include the ability to include explicit 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 our system, as in

(9) *Find me a way to the theater with transit that has the fewest transfers and arrives by 8:30pm with 95% likelihood*

This constraint will generate the **WITH CERTAINTY** clause we discussed above. In the future we will explore the use of non-numeric likelihood values (e.g., *very likely, absolutely*) as they are likely more intuitive to users and could be customized.

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 Evaluation

In this section we present the evaluation of NL2TRANQUYL. First, we introduce the procedure used, secondly we present an evaluation of system performance on three sets of requests: well-formatted and grammatical requests as collected from external informants (set A); grammatical paraphrases of the requests (set B); telegraphic or fragmented requests (set C) – we generated sets B and C. In all cases the system performs with high accuracy – that is, for the majority of the input requests (80%), NL2TRANQUYL generates a completely correct translation into TRANQUYL; the accuracy on individual statements in the TRANQUYL queries is 94% (namely, even when the translation of a request is not completely correct, most of the individual statements are). The evaluation will also touch on processing the implicit requests mentioned in Section 1.2. We then present a discussion of an informal evaluation to identify shortcomings in the language scope and to direct future work.

Due to space limitations, we are unable to include the test corpus in this article: it can be found at http://nlp.cs.uic.edu/nl2tranquyl.

5.1 Evaluation Procedure

As noted for example in (Jurafsky & 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 our work here, since the ITA in which NL2TRANQUYL is to be embedded is not available yet; neither is an appropriate database. Hence, we conducted an intrinsic evaluation, i.e., an evaluation independent of the application, in which we evaluated the quality of the translations of NL requests into TRANQUYL.

Even for an intrinsic evaluation, we faced an obstacle, namely, the lack of any standard corpus, available for trip planning of the sort we are addressing in our work. We discussed the lack of such corpus earlier in Sec. 1.2, which forced us to collect our own corpus of requests. However, since those requests had been used for development, they clearly could not be used for testing. It became then clear that we needed to generate a second corpus of requests, 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, our 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 & Porter, 1997) to Machine Translation (Vilar *et al.*, 2006)

In order to perform the intrinsic evaluation, we developed a corpus of 162 test requests: 50 of those (which we call A requests) were generated independently from us, 100 (B and C requests) were systematic variations on the A requests that we generated; finally, we generated 12 additional requests (D requests) to specifically test NL2TRANQUYL on implicit requests. As concerns the set of 50 A requests, we solicited the input from three students related to the transportation research project. We provided a list of requests that demonstrated the functionality of the system as well as a background on roughly how the system worked, similarly to how we had solicited the initial set of development questions (see Appendix A – the sentence *For at least half of them, please make them multi-part questions* was omitted in this case, and so where multi-sentence examples). They were instructed to generate new requests 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 three users were not allowed to write random requests outside of the scope of the system; this evaluation was designed to test the system on what it should be able to process. They could fail by way of grammar, complexity, or the use of synonyms. Hence, for this evaluation, we spellchecked the requests. Note that since the requests were provided by email the vast majority of spelling mistakes were automatically flagged – other than one or two instances of easily confused real words such as “their” instead of “there” or “its” instead of “it’s”. We also substituted occasional generic names of facilities, like *acupuncturist*, that were not one of the 24 classes of *Resources* existing in the ontology (Figure 2 only includes two out of 24, *restaurant* and *bank*): our interest was not in assessing the coverage of the ontology, but the performance of the translation pipeline.

On this set of 50 A requests, we pitted the NL2TRANQUYL software against two humans: an expert and a novice. The goal of the test was to assess whether the software would produce semantically equivalent TRANQUYL queries to those produced by 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 requests from English into TRANQUYL. We selected a person unfamiliar with the software and language in order to remove any potential bias from prior knowledge.

We also had an expert (the first author and designer of the language) translate the requests. Given his knowledge of both TRANQUYL and NL2TRANQUYL, he should be able to produce perfect queries. This provides us with a true gold standard. We realize that the designer of the language could have introduced bias, conscious or unconscious, into the translations, since he knows how NL2TRANQUYL would translate those requests. We believe this concern is assuaged because the queries produced by the novice constitute a second unbiased corpus of translations. As we will show later, the "novice" turned out to be a highly competent TRANQUYL query writer (not surprisingly, for a PhD student in the database area). By comparing how well the software performs with respect to the novice, in addition to how it performs with respect to the expert, we obtain a measure of performance for the system that is not tainted by potential bias. Even if the software produces errors, we will show that its performance is comparable to either of the two humans, the expert or the "novice".

We briefly note that we attempted a second type of evaluation, this one exploiting Amazon's Mechanical Turk. We took our input requests and blindly ran them through the software system. The first author then translated back from the TRANQUYL query to an equivalent English request. These natural language "outputs" were then paired with their corresponding input requests. "Turkers" were asked to determine whether the two English requests were equivalent. This attempt at evaluation failed for two reasons: first, not many turkers took up the task; second, those few who did produced very inconsistent results. Clearly, simply asking "do those two requests mean the same" means different things to different people: as others have noted (Stoyanchev *et al.*, 2012), "turkers" need to be provided with very specific, focused tasks for their work to be effective. As a consequence, we abandoned that attempt and we focused on the evaluation we are now turning to describe.

5.2 Performance on Human Generated Requests (Set A)

[Figure 10 about here.]

As mentioned, the first test set (A set) contains a total of 50 requests. They are translated into TRANQUYL queries that range in complexity from 1 to 10 necessary statements. A statement is 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 we judge complexity based on the necessary output in a formal language.

The distribution of query complexity can be found in Figure 10. Informally, we found that queries of length 5-7 statements were able to capture most reasonable trip requests. To illustrate the correspondence between input request and number of query statements, the examples in (10) below list the number of statements in the TRANQUYL query corresponding to the NL request (Example (10c) repeats the earlier Example (5a)).

(10a) 3 statements: *Find the best way to my apartment*

(10b) 5 statements: *What is the most reliable transit route to a restaurant that costs less than $2.50?*

(10c) 10 statements: *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?*

The set of 50 evaluation requests resulted in queries that collectively contain 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 requests, which means 80% of the requests were processed correctly. The novice processed 94% of the requests correctly. Of those 10 requests where NL2TRANQUYL made errors, 1 had 3 errors, 1 had 2 errors, and the remaining 8 each had 1 error. Only 3 of the requests with errors had 6 or fewer statements – meaning that errors generally occurred on longer, more complex requests.

In addition to accuracy, we can calculate a measure of precision and recall for different aspects of the query by using our atomic unit of a statement. Precision and recall are appropriate metrics since a statement can be wrong, or can be missing altogether. Tables 3 and 4 show where these errors occur. Table 3 provides the details for which type of statement the error occurred in, and Table 4 details the types of errors found within the WHERE clause. We 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, but better than NL2TRANQUYL. It is interesting to note that if we were to consider the novice as the provider of the "gold-standard", NL2TRANQUYL would apparently perform worse than in comparison to the expert: the system does not make any mistakes on the OPTIMIZE clause (see Table 3), but that's where all the novice mistakes lie. Hence, NL2TRANQUYL would make 16 mistakes with respect to the novice query translations, for an accuracy of 95%.

[Table 3 about here.]

[Table 4 about here.]

The types of errors chronicled in Table 4 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 the most missed type of constraint, i.e., they were omitted from the output. Facility constraints are those specifying that the trip include a certain facility or resource along the way (e.g., stopping at the bank). One common error was swapping an intermediate node with the destination node – this also affects the precision and recall for the SELECT and WITH STOP_VERTICES clauses in addition to the WHERE clause.

We find these results to be very promising. The overall accuracy is high, with respect to both gold standards: the true one, produced by the expert, but possibly biased by his knowledge of the system; and the one produced by the "novice", which contains only three errors over 296 statements (interestingly, these three errors all concerns the OPTIMIZE clause, on which the system doesn't make any mistakes). NL2TRANQUYL errors are more prevalent in longer, more complex requests which seem less likely to be issued in real world applications. We will provide an analysis of performance on different types of syntactic constructions in Section 5.5 below.

5.3 Paraphrases and Fragments (Sets B and C)

In order to test the robustness of our system, we decided to systematically rewrite the first set of 50 test requests used in the previous evaluation. To illustrate, we provide the following two examples of the alternative formulations A/B/C. Query *A* is the initial request, *B* is the paraphrased rewrite, and *C* is the fragmented rewrite.

(11a) A. *What is the cheapest way to visit my bank, a bakery, and a gym on the way to my office.*

(11b) B. *Start by visiting my bank, a baker, and a gym and then go to my office for the cheapest amount of money.*

(11c) C. *cheapest way to my office after bank, bakery, gym*

(12a) A. *Can I drive to a bank not crossing a river while arriving no later than 5:45pm?*

(12b) B. *Arriving at a bank no later than 5:45, can I do it by driving and not crossing a river?*

(12c) C. *drive to bank not crossing river by 5:45pm*

In each case, request *B* is rewritten in order to capture variability in how requests may be asked. Short requests had additional clauses added even if they were not necessary. Long requests had their clauses rearranged or compacted. Where possible, we changed the verb tenses or used synonyms. The fragmented requests 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 standard grammatical conventions. The Stanford parser does return outputs even in these cases, where the root of the constituency tree is often marked as *FRAG*.

For this experiment we only compared the system output to the correct output; we did not involve another human translator. In each case the output for requests *A*, *B*, and *C* should be identical and we already have the correct output for each *A* query to compare against. Similarly, the complexity of the requests 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 request.

To begin, recall that in terms of raw numbers only 10 of the 50 *A* requests had any type of error, for a cumulative total of 13 mistakes. For the *B* requests, 14 of the requests produced errors, one resulting in a complete failure (no output generated, the only instance across the 162 total requests), with the remaining 13 cumulatively resulting in 21 total mistakes. The fragmented *C* requests fared worse: 24 gave rise to errors, for a total of 38 cumulative mistakes. Interestingly, half of the B requests corresponding to the original 10 wrong A requests, are translated correctly; instead all but one of the C requests corresponding to the original set of wrong A requests is translated wrong. These results show that our system is robust in handling requests that are expressed in varying ways, including half of those that use keywords and a telegraphic style. Whereas we do not rely on the user to follow a prescribed grammar, expressing the sentence grammatically still gives the best results. This shows the importance of actually employing a parser, as opposed to employing simpler approaches such as chunking.

5.4 Implicit Requests (Set D)

As discussed previously, a request such as *Find the nearest bank* can be interpreted as an implicit request 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 request *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, we considered allowing requests such as these to be answered explicitly and presented a simple heuristic for classifying requests based on what type of target they were for (e.g., a trip, resource, time) (Booth *et al.*, 2009b). However, we decided that focusing on trips alone actually provides greater functionality and accuracy. Indeed these requests often count as *indirect speech acts*, where the user's intent goes beyond what can be inferred from the surface form of the request (Austin, 1962; Searle, 1975). Computational models of how to reply to these questions with an extended response rather than with a yes/no questions, for example *overanswering*, have also been studied for many years (Kaplan, 1982; Di Eugenio, 1987).

Given the robust results presented in the previous sections, it is not surprising that requests written like this can be processed correctly. Many of the requests in sets A and B are implicit requests to start with, especially if they are phrased as *yes/no* questions. In a small test of 12 additional requests, all were processed correctly. Among these 12 requests are the two requests included earlier in this section; two others are *Can I walk to the grocery store by 7:00pm?* and *Where is the nearest gas station?*[*]

5.5 Performance and Error Analysis

Whereas it is not possible to pinpoint the specific cause for each error, we can draw some general conclusions, especially as concerns the C requests. We will discuss A and B requests together, and C requests on their own.

As concerns the A and B request sets, most of the complex constructions those requests contain are interpreted correctly. For example, four A requests and five B requests are phrased as an expletive *there* question – see the A requests in the earlier example (5a) and in Example (13) below, and the B requests in the earlier Example (5b), and in Example (14) below. Eight out of these nine requests are translated correctly. The one B query which is wrong is missing intermediate stops, but this is likely due to the conjoined NPs, rather than to the expletive *there* (see below for a discussion of conjoined NPs).

(13) *Is there a way to home that includes a pharmacy?*

(14) *Driving to work, is there a shortest path that doesn't cross a river but also includes a bank to stop at?*

---

[*] Note that these 12 implicit queries are not listed at http://nlp.cs.uic.edu/nl2tranquyl/queries.txt.

There can also function as a referential pronoun. Two occurrences of referential *there* appear in A requests, as in Example (15) below, where *there* refers to (the location of) *my doctor*, and four in B requests. All of those are translated correctly, other than one B query (where the mistake is on time not on destination). We do not perform reference resolution other than for personal references (see Section 4.2). Hence, when referential *there* is processed by the Knowledge Map algorithm, no mapping concept is found, and *there* is ignored. Note that in all our examples, referential *there* always refers to a concept present in the same sentence (i.e. *my doctor* in (15)), which explains why it can be safely ignored.

(15) *Leaving from my bank, get the shortest walking path to my doctor that gets there by 4:00pm*

Other pervasive constructions include *preposed gerundive adjuncts*, *relative clauses*, and *conjunctions*. *Preposed gerundive adjuncts* are adjunct clauses that appear at the beginning of the sentence and don't have a main verb, but a gerund, like *Driving to work* in Example (14), and *Leaving from my bank* in Example (15). Two A requests include these constructions, one being Example (15); 31 B requests include them. The two A requests and 23 of the B requests produce completely correct output; for the eight wrong B requests, in only three cases does the error concern a constituent embedded in the adjunct. In those cases, the error appears to be due to other factors, not the adjunct per se, as in Example (12b), repeated here:

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

This example produces the incomplete query included here, missing the time and the *negation* as concerns *crossing*:

```
SELECT * FROM
  ALL_TRIPS(user.current_location, dStation) AS t, River AS R
    WITH MODES Auto
    WHERE "Bank" IN dStation.facilities
      AND INTERSECTS(GEOMETRY(t), R.area)
    MINIMIZE COST(t)
```

As concerns the omitted negation, this is due to an error in the dependency parse tree, where *not* is not marked as a negation as usual, but simply, as conjoined to *driving*, with *driving* and *crossing* linked by *dep*, the generic dependency that the Stanford parser uses when it cannot compute a more informative one (De Marneffe & Manning, 2008). As concerns the omitted time, almost all our requests, and specifically 47 triples of requests A, B, C, include an explicit time expression, such as *by 8:30pm*, *before 6am*, or *no later than 7:00pm*, for a total of 141 time expressions. Eleven of those time expressions are omitted in the final query, for a recall of 92%. Three times this is due to the time expression not including *pm* or *am*, as in Example (16). In these three cases, the corresponding time is not labeled as a *num* in the dependency tree, and hence it is missed by our **getRegExConcepts** algorithm (as described in Section 4.1). In a corresponding pair of A/B requests (hence, two cases), the explicit time expression *by 6:00am* is itself modified, as in *by 6:00am this morning*. In this case it is our pruning method in the algorithm in Figure 6 that is not able to deal with an NP which is in turn composed by two juxtaposed NPs. Interestingly in this case, the more succinct C request does not include *this morning* and is therefore processed correctly – the only case in which a C request is correct, but the corresponding A and / or B request is not. The precision on the 130 temporal constraints that are included in TRANQUYL queries, it is 96%, namely, 126 of those temporal expressions are translated correctly. Mistakes refer to the temporal direction of the constraint, for example translating *Leaving no earlier than 7:00pm* as `WHERE ENDS(t) < 7:00pm`, instead of as `WHERE BEGINS(t) > 7:00pm`.

As concerns relative clauses (all introduced by *that*), there are 39 total occurrences, of which 22 in 21 A requests (one A request includes two relative clauses), and 17 in B requests – Examples (3), (6), (5a), (5b) earlier in the paper include relative clauses. Of these, there are mistakes in two A requests and two B requests, and they concern missing intermediate stops, which is what we turn to now. These are the most common mistakes across all requests, and we discuss the cause behind most of the mistakes in the C requests below. For A or B requests, it is hard to find a general explanation for what the problem may be, because these errors occur in complex requests that include relative clauses, conjunctions etc; each of these constructions may be the cause of the mistake. For example, consider the pair of A and B requests in Examples (17a) and (17b); the translation for Example (17a) does not include the intermediate stop *bank*, whereas the translation for Example (17b) does:

(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?*

Example (17a) is misparsed, with *work* analyzed as a verb, and *that* as a complementizer, as if it were ... *drive to say that ...* On the other hand, the relative clause in Example (17b) is correctly parsed.

As concerns the C requests, the six requests that include a main verb are all successfully processed, including Example (12c) above, even if the request is otherwise ungrammatical, e.g. *walk my house shortest*. 23 out of the 38 total mistakes (60%) are due to missing all or some of the intermediate stops. In 21 out of these 23 cases, the construction that triggers the failure is the PP attachment expressing the intermediate stop, such as the PP *after bank, bakery, gym* in Example (11c), as is resolved by the algorithm that computes the Knowledge Map. When the query is fragmented, those PPs are parsed as modifiers of the source or destination of the trip (*office* in Example (11c)), or of the trip itself (as in requests like *transit to my house with bank, grocery*). In the algorithm in Figure 9, the body of the second *for* loop (*for all Concepts* $c \in map$) looks for a path in the ontology between the two concepts involved in the dependency that models the PP in question; if that fails, a connection is sought for all the concepts in the immediate vicinity of other concepts. However no connection is found between concepts that are all subconcepts of *Resource* (see Figure 2), hence the PP is dropped from the mapping. The same explanation applies to four additional cases such as *restaurant before 5pm* in which the temporal PP is interpreted as a modifier of the destination, as opposed to modifying the main trip like in *drive restaurant before 5pm*. Finally, in two cases certainty is just expressed as an apposition, as in Example (18), and it is dropped as well (the apposition is probably the cause of a second mistake in the translation of Example (18), since time is omitted as well):

(18) *transit to theater with fewest transfers by 8:30pm, 95% certain*

Among other miscellaneous errors, the negation is not recognized in Example (19), the only mistake on this rather complex request – perhaps because the dependency between *expensive* and *cross* is only marked as a generic *dep*.

(19) *least expensive to my apartment not cross river by 7:00pm with 99% certainty*

5.6 Informal Evaluation

After performing a formal evaluation of NL2TRANQUYL we 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 our initial informative data collection. We solicited requests from laypersons with no experience with our 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 requests. The note contained a list of dozen requests from the evaluation corpus as well as a brief description of the project and its goals. For generating new requests, 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 everything is captured in the list below.

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

We were 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 we should take the project in the future. From this solicitation we received 42 new requests. Of these, roughly one third were immediately compatible with our system, with the caveat that certain landmarks needed to be added to the ontology. At this time we 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 requests would be understandable within our framework with small extensions. These requests 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 requests looking for attributes of the trip (e.g., *How long does it take to walk to my house?*) – the implicit trip request 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 requests as shortcomings, we take a more optimistic approach. Our 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 requests 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, as in Example (20); and multi-sentence requests that seek information beyond the data model, as in Examples (21a) and (21b):

(20) *Where can I go for dinner that is easy for Alice and Bob to meet me?*

(21a) *Find a bus to the nearest hotel... how was it rated by previous guests*
(21b) *Which bus goes to the airport? ... Is it air-conditioned?*

The final use for the results obtained in this informal evaluation is that we 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 requests that may be related to trip planning but are beyond the understanding of the system as it stands.

6 Conclusions and Future Work

As shown in our discussion and evaluation, NL2TRANQUYL provides robust handling of trip planning queries expressed in natural language. The low error rate demonstrated in the formal evaluation in conjunction with the handling of poorly and partially formed input sentences provides a solid foundation for a real-world system. This is feasible as we begin by identifying the smallest identifiable semantic units in the input and building up our knowledge from there with the ontology providing guidance of what to look for—we do not start with the assumption that the input is good or in any singularly coherent format.

There are a number of directions we are pursuing for further work on this project. The first, and we believe most important, is continuing to work with collaborators in the development of a fully functioning ITA where our NLI can provide an interface option. Among others, this would require extensions to the ontology, both as concerns its content, and access and traversal, as we mentioned in Section 3.2. Once the system is situated in a user accessible environment, we can further evaluate the efficacy of the NLI as it exists now. Additionally, we 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 (Rosé & Lavie, 2000; Misu & 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 their requests in multiple steps (i.e., sentences), which is currently not supported; as we discussed earlier, both our initial data collection, and our informal evaluation with Facebook yielded several examples of multi-steps requests.

Even with the strong performance of the system, our informal evaluation found some areas in which we can improve. One straightforward extension is to allow for more phrasings of temporal constraints, e.g., *Find a bus home that leaves in 2 hours*. In our initial evaluation we required that the time be stated explicitly, but this type of extension is straightforward to include. Likewise, since we have identified the causes of some mistakes such as PP attachments for request set C, we can devise heuristics to deal with these cases. Some of the identified language (e.g., knowledge of signalized intersections, cleanliness) require extensions to the TRANQUYL data model. An intriguing venue for future work is adapting methods for supervised or unsupervised learning of additions to the ontology, as inspired by work such as (Thompson & Mooney, 2003; Muresan, 2008).

In conclusion, we 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.

Appendix A. Instructions for original corpus collection

The email below was sent by the first author to the set of students, colleagues and friends described in Section 1.2. Please note that all the recipients used to work and / or live in Chicago and surrounds, and hence all the references below were familiar to them.

> One of the goals [of my work] is to provide a NLI for the ITA in order to allow users to query the available information in a natural and intuitive manner. One important question centers around the language or grammar used by users – what types of questions are asked and how do they get asked? It is for this task that I am asking for a small portion of your time to help.
>
> I have described a number of potential situations in which the ITA would be useful. For each of these I would appreciate it if you would give three example questions that you would ask the ITA. For at least half of them, please make them multi-part questions. We are assuming that the types of information (context, road, traffic, congestion, point of interest) exist. Please be yourself in these situations; use your own preferences, personality, and even willingness to spend. This is about what you would want to know. Assume that the ITA returns a ranked set of results. The ITA should know at least some basic things about you in terms of your user profile (e.g. has a car?, willing to take a bus?, etc.).
>
> Some examples:
>
> You are stuck in traffic on 90/94 at 2:00 pm on a Saturday.
>
> 1. How much longer will it take me to get to the Fullerton exit?
>   (a) How is the congestion two miles up the road?
>   (b) Is there a faster route?

2. Give me the fastest route to Fullerton and Western.

You are meeting a friend at Grant Park after class, and plan to see some of the sites of the city.

1. How can I get to Grant Park from here for less than $8?
    (a) Which is the fastest option?
2. How do I get from here to Grant Park? (assuming there is a default mode of transportation)
3. Where is the closest taxi?
4. Will there be a ride share from the park to the Sear's Tower around 4:00 pm?

And onto the situations. Feel free to simple reply to and edit this email to add your questions. The responses from all volunteers will be aggregated and any reference to you will be removed. If you have any additional questions, comments, or concerns please contact me.

Situations:

1. You have just arrived at O'Hare Airport and need to find transportation to your hotel downtown.
2. You are stuck in traffic near 35th St. on 90/94 at noon pm on a Saturday while going to meet your friends for lunch.
3. You and your date finished dinner in the near north side and are looking for something else to do.
4. You are meeting a friend at Grant Park after class, and plan to see some of the sites of the city.
5. You are on UIC's campus and need to get to the Sears' Tower as soon as possible for a meeting.
6. You stayed in the lab later than normal and your ride home left without you.

References

Androutsopoulos, I., Ritchie, G.D., & Thanisch, P. 1995. Natural Language Interfaces to Databases–an introduction. *Journal of Language Engineering*, **1**(1), 29–81.

Austin, John L. 1962. *How to Do Things With Words*. Oxford: Oxford University Press.

Bangalore, S., & Johnston, M. 2009. Robust Understanding in Multimodal Interfaces. *Computational Linguistics*, **35**(3), 345–397.

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

Bernsen, Niels Ole, & Dybkjær, Laila. 2002. A multimodal virtual co-driver's problems with the driver. *In: Proceedings of the ISCA Tutorial and Research Workshop on Spoken Dialogue in Mobile Environments*. Kloster Irsee, Germany: International Speech Communication Association.

Bernstein, A., Kaufmann, E., & Kaiser, C. 2005 (December). Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine. *Pages 45–50 of: 15th Workshop on Information Technology and Systems (WITS)*.

Black, Alan W., Burger, Susanne, Conkie, Alistair, Hastie, Helen, Keizer, Simon, Lemon, Oliver, Merigaud, Nicolas, Parent, Gabriel, Schubiner, Gabriel, Thomson, Blaise, Williams, Jason D., Yu, Kai, Young, Steve, & Eskenazi, Maxine. 2011 (June). Spoken Dialog Challenge 2010: Comparison of Live and Control Test Results. *Pages 2–7 of: Proceedings of SIGDial 2011, the 12th Conference of the ACL Special Interest Group in Discourse and Dialogue*.

Booth, Joel. 2011. *Modeling and Querying Multimodal Urban Transportation Networks*. Ph.D. thesis, University of Illinois at Chicago.

Booth, Joel, Sistla, Prasad, Wolfson, Ouri, & Cruz, Isabel F. 2009a. A data model for trip planning in multimodal transportation systems. *Pages 994–1005 of: 12th International Conference on Extending Database Technology (EDBT)*.

Booth, Joel, Di Eugenio, Barbara, Cruz, Isabel F., & Wolfson, Ouri. 2009b (September). Query Sentences as Semantic (Sub) Networks. *Pages 89–94 of: The 3rd IEEE International Conference on Semantic Computing (ICSC)*.

Branavan, S.R.K., Zettlemoyer, Luke, & Barzilay, Regina. 2010. Reading between the Lines: Learning to Map High-Level Instruction s to Commands. *Pages 1268–1277 of: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics.

Branavan, S.R.K., Kushman, Nate, Lei, Tao, & Barzilay, Regina. 2012a. Learning High-Level Planning from Text. *Pages 126–135 of: Proceedings of the 50th Annual Meeting of the Association for Com putational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea: Association for Computational Linguistics.

Branavan, S.R.K., Silver, D., & Barzilay, R. 2012b. Learning to Win by Reading Manuals in a Monte-Carlo Framework. *Journal of Artificial Intelligence Research*, **43**, 661–704.

Brunner, Levin, Schulz, Klaus U., & Weigel, Felix. 2006. Organizing Thematic, Geographic and Temporal Knowledge in a Well-founded Navigation Space: Logical and Algorithmic Foundations for EFGT Nets. *International Journal of Web Services Research*, **3**(4), 1–31.

Budanitsky, Alexander, & Hirst, Graeme. 2006. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, **32**(1), 13–47.

Buehler, D., Vignier, S., Heisterkamp, P., & Minke, W. 2003 (Miami, FL, USA). Safety and Operating Issues for Mobile Human-Machine Interfaces. *Pages 227–229 of: International Conference on Intelligent User Interfaces (IUI)*.

Chen, D.L., Kim, J., & Mooney, R.J. 2010. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, **37**(1), 397–436.

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

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

De Marneffe, Marie-Catherine, & Manning, Christopher D. 2008. *Stanford Typed Dependencies Manual*.

Di Eugenio, Barbara. 1987. Cooperative Behaviour in the FIDO System. *Information Systems*, **12**, 295–316.

Dillenburg, J.F., Wolfson, Ouri, & Nelson, P.C. 2002 (September). The Intelligent Travel Assistant. *Pages 691– 696 of: IEEE 5th Annual Conference on Intelligent Transportation Systems (ITS)*.

Dzikovska, Myroslava O., Allen, James F., & Swift, Mary D. 2008. Linking semantic and knowledge representations in a multi-domain dialogue system. *Journal of Logic and Computation*, **18**(3), 405–430.

Ferrández, Oscar, Spurk, Christian, Kouylekov, Milen, Dornescu, Iustin, Ferrández, Sergio, Negri, Matteo, Izquierdo, Ruben, Tomás, David, Orasan, Constantin, Neumann, Guenter, Magnini, Bernando, & Vicedo, J. L. 2011. The QALL-ME framework: A specifiable-domain multilingual question answering architecture. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, **9**(2), 137–145.

Fonseca, F.T, Egenhofer, M.J, C.A. Davis, Jr., & Borges, K.A.V. 2000. Ontologies and knowledge sharing in urban GIS. *Computers, Environment and Urban Systems*, **24**(3), 251 – 272.

Garcia, Vanessa Lopez, Motta, Enrico, & Uren, Victoria. 2006. AquaLog: An ontology-driven question answering system to interface the semantic web. *Pages 269–272 of: North American Chapter of the Association for Computational Linguistics Conference on Human Language Technology (NAACL-HLT).*

Geutner, P., Steffens, F., & Manstetten, D. 2002 (June). 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).*

Grasso, Michael A., Ebert, David S., & Finin, Timothy W. 1998. The integrality of speech in multimodal interfaces. *ACM Transactions on Human-Computer Interaction (TOCHI)*, **5**(4), 303–325.

Gruenstein, A., & Seneff, S. 2007. Releasing a Multimodal Dialogue System into the Wild: User Support Mechanisms. *Pages 111–119 of: Proceedings of SIGDial 2007, the 8th Workshop of the ACL Special Interest Group in Discourse and Dialogue*.

Gruenstein, Alex. 2008 (June). City Browser: A Web-Based Multimodal Interface to Urban Information. *In: ACL Workshop on Mobile Language Processing Demo (ACL-HLT).*

Hemphill, Charles T., Godfrey, John J., & Doddington, George R. 1990. The ATIS Spoken Language Systems Pilot Corpus. *Pages 96–101 of: DARPA Speech and Natural Language Workshop*. Morgan Kaufmann.

Hirst, Graeme. 1981. Discourse-oriented anaphora resolution in natural language understanding: a review. *Computational Linguistics*, **7**(2), 85–98.

John, B., Salvucci, D., Centgraf, P., & Prevas, K. 2004 (July). Integrating models and tools in the context of driving and in-vehicle devices. *Pages 130–135 of: International Conference on Cognitive Modeling*.

Johnston, Michael, Bangalore, Srinivas, Vasireddy, Gunaranjan, Stent, Amanda, Ehlen, Patrick, Walker, Marilyn, Whittaker, Steve, & Maloor, Preetam. 2002. MATCH: an architecture for multimodal dialogue systems. *Pages 376–383 of: Annual Meeting on Association for Computational Linguistics (ACL).*

Jurafsky, Daniel, & Martin, James H. 2009. *Speech and Language Processing*. Second edn. Pearson Education – Prentice Hall.

Kaplan, S. Jerrold. 1982. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, **19**(2), 165–187.

Kate, R.J., & Mooney, R.J. 2007. Learning language semantics from ambiguous supervision. *Pages 895–900 of: AAAI-2007, Proceedings of the Twenty-Second Conference on Artificial Intelligence*. AAAI Press.

Kaufmann, E., Bernstein, A., & Zumstein, R. 2006. Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. *Pages pp. 980–981 of: 5th International Semantic Web Conference (ISWC).*

Kaufmann, E., Bernstein, A., & Fischer, L. 2007 (June). NLP-Reduce: A "naïve" but Domain-independent Natural Language Interface for Querying Ontologies. *In: 4th European Semantic Web Conference (ESWC).*

Kaufmann, Esther, & Bernstein, Abraham. 2008. How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users? *Pages 281–294 of: 6th International and 2nd Asian Semantic Web Conference (ISWC/ASWC).*

Klein, Dan, & Manning, Christopher D. 2003 (July). Accurate Unlexicalized Parsing. *Pages 423–430 of: 41st Annual Meeting of the Association for Computational Linguistics (ACL).*

Lester, James C., & Porter, Bruce W. 1997. Developing and Empirically Evaluating Robust Explanation Generators: the KNIGHT Experiments. *Computational Linguistics*, **23**(1), 65–102. Special Issue on Empirical Studies in Discourse.

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

Liang, Percy, Jordan, Michael, & Klein, Dan. 2009. Learning Semantic Correspondences with Less Supervision. *Pages 91–99 of: Proceedings of ACL.* Suntec, Singapore: Association for Computational Linguistics.

Liang, Percy, Jordan, Michael I., & Klein, Dan. 2013. Learning Dependency-Based Compositional Semantics. *Computational Linguistics*, **39**(2).

Lopez, V., Motta, E., & Uren, V. 2006 (June). PowerAqua: Fishing the Semantic Web. *Pages 393–410 of: 3rd European Semantic Web Conference (ECSW).*

Lorenz, Bernhard, Ohlbach, Hans Jürgen, & Yang, Laibing. 2005. *Ontology of transportation networks*. Tech. rept. REWERSE Project IST-2004-506779.

Marçal de Oliveira, Káthia, Bacha, Firas, Mnasser, Houda, & Abed, Mourad. 2013. Transportation ontology definition and application for the content personalization of user interfaces. *Expert Systems with Applications*, **40**(8), 3145 – 3159.

Meisel, William. 2013. The Personal-Assistant Model: Unifying the Technology Experience. *Pages 35–45 of:* Neustein, Amy, & Markowitz, Judith A. (eds), *Mobile Speech and Advanced Natural Language Solutions*. Springer.

Meng, F., & Chu, W. 1999. *Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation*. Tech. rept. University of California at Los Angeles, Los Angeles, CA, USA.

Misu, Teruhisa, & Kawahara, Tatsuya. 2006. Dialogue strategy to clarify user's queries for document retrieval system with speech interface. *Speech Communication*, **48**(9), 1137 – 1150.

Mnasser, Houda, Khemaja, Maha, Marçal de Oliveira, Káthia, & Abed, Mourad. 2010 (May). A public transportation ontology to support user travel planning. *Pages 127–136 of: International Conference on Research Challanges in Information Science (RCIS).*

Moldovan, Dan, Clark, Christine, Harabagiu, Sanda, & Hodges, Daniel. 2007. COGEX: A semantically and contextually enriched logic prover for question answering. *Journal of Applied Logic*, **5**(1), 49 – 69.

Muresan, Smaranda. 2008. Learning to map text to graph-based meaning representations via grammar induction. *Pages 9–16 of: Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing*. COLING Workshop.

Neustein, Amy, & Markowitz, Judith A. (eds). 2013. *Mobile Speech and Advanced Natural Language Solutions*. Springer.

Niaraki, Abolghasem Sadeghi, & Kim, Kyehyun. 2009. Ontology based personalized route planning system using a multi-criteria decision making approach. *Expert Systems with Applications*, **36**(2, Part 1), 2250 – 2259.

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

Prince, E. 1981. Toward a Taxonomy of Given-New Information. *Pages 223–255 of:* Cole, P. (ed), *Radical Pragmatics*. Academic Press.

Raux, Antoine, Langner, Brian, Bohus, Dan, Black, Alan W., & Eskenazi, Maxine. 2005 (June). Let's go public! taking a spoken dialog system to the real world. *Pages 885–888 of: Interspeech*.

Rosé, Carolyn Penstein, & Lavie, Alon. 2000. Balancing Robustness and Efficiency in Unification-Augmented Context-Free Parsers for Large Practical Applications. *In:* Junqua, Jean-Claude, & van Noord, Gertjan (eds), *Robustness in Language and Speech Technology*. Kluwer Academic Press.

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

Searle, John R. 1975. Indirect Speech Acts. *In:* Cole, P., & Morgan, J.L. (eds), *Syntax and Semantics 3. Speech Acts*. Academic Press.

Sharman, R., Kishore, R., & Ramesh, R. (eds). 2007. *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*. Integrated Series in Information Systems, no. 14. Springer–Verlag.

Staab, Steffen, & Studer, Rudi (eds). 2009. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer–Verlag.

Stoyanchev, Svetlana, Liu, Alex, & Hirschberg, Julia. 2012. Clarification Questions with Feedback. *In: Proceedings of Interspeech*.

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

Teller, Jacques, Keita, Abdelkader, Roussey, Catherine, & Laurini, Robert. 2007. Urban Ontologies for an improved communication in urban civil engineering projects. *Cybergeo: Revue Européenne de Géographie= Cybergeo-European Journal of Geography*.

Thompson, C.A., & Mooney, R.J. 2003. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research*, **18**(1), 1–44.

United States Research and Innovative Technology Administration. 2011. *ITS Strategic Research Plan, 2010-2014: Executive Summary.*

Vilar, David, Xu, Jia, D'Haro, Luis Fernando, & Ney, Hermann. 2006 (May). Error Analysis of Statistical Machine Translation Output. *Pages 697—702 of: 5th International Conference on Language Resources and Evaluation (LREC).*

Walker, Marilyn A., Litman, Diane J., Kamm, Candace A., & Abella, Alicia. 1997 (July). PARADISE: A Framework for Evaluating Spoken Dialogue Agents. *Pages 271–280 of: Annual Meeting of the Association for Computational Linguistics (ACL).*

Wang, Chong, Xiong, Miao, Zhou, Qi, & Yu, Yong. 2007 (June). PANTO: A Portable Natural Language Interface to Ontologies. *Pages 473–487 of: European Semantic Web Conference (ESWC).*

Wang, Junli, Ding, Zhijun, & Jiang, Changjun. 2005. An Ontology-based Public Transport Query System. *Page 62 of: SKG'05, First International Conference on Semantics, Knowledge and Grid.* IEEE.

Webber, B.L. 1978. *A Formal Approach to Discourse Anaphora.* Ph.D. thesis, Harvard University, Cambridge, MA, USA.

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

Zettlemoyer, Luke, & Collins, Michael. 2009. Learning Context-Dependent Mappings from Sentences to Logical Form. *Pages 976–984 of: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP.* Suntec, Singapore: Association for Computational Linguistics.

Zhong, Ting, Xu, Bo, & Wolfson, Ouri. 2008. Disseminating real-time traffic information in vehicular ad-hoc networks. *Pages 1056–1061 of: Intelligent Vehicles Symposium, 2008 IEEE.* IEEE.

Table 1: Weights for String Matches

Target Concept c	**Distance Measure**	**Weight**
Concept Label	Equal	1
Colloquial Label	Equal	1
Concept Label	Stemmed Equal	0.25
Colloquial Label	Stemmed Equal	0.25
Concept Label	Edit Distance: 1	0.15
Colloquial Label	Edit Distance: 1	0.15

Table 2: Weights for WordNet Matches

Node n	**Relation in WordNet**	**Weight**
Lemma	Synonym	.25
Lemma	Hypernym	.05
Lemma	Hyponym	.05
Lemma+POS	Synonym	.50
Lemma+POS	Hypernym	.10
Lemma+POS	Hyponym	.10

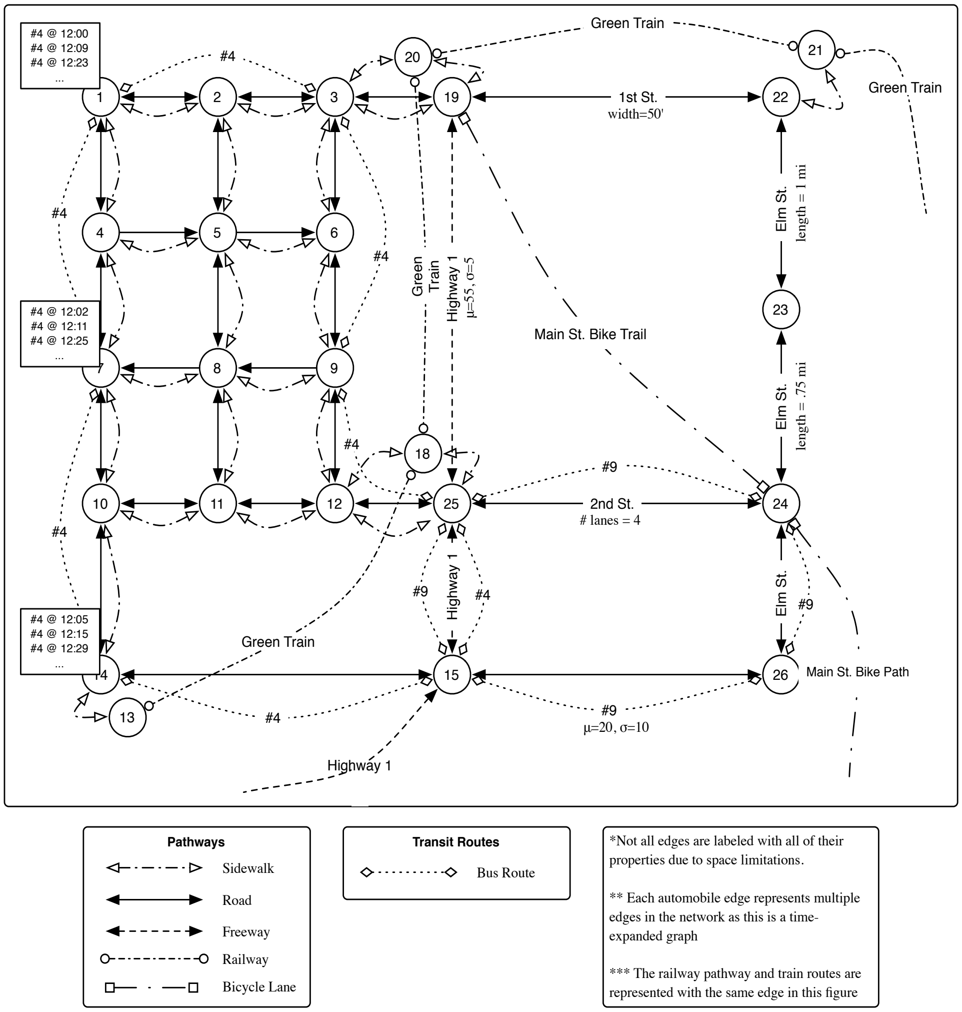
Table 3: Precision and recall for different types of statements (set A)

	Precision	**Recall**	**F-Score**
Select (ALL_TRIPS)	0.95	0.95	0.95
With Modes	1.00	1.00	1.00
With Stop_Vertices	0.88	0.79	0.83
With Certainty	1.00	1.00	1.00
Where	0.94	0.90	0.92
Optimize	1.00	1.00	1.00
Cumulative	0.97	0.95	0.96

Table 4: Precision and recall for WHERE clauses (set A)

	Precision	**Recall**	**F-Score**
Temporal	0.92	0.86	0.89
Facility	0.96	0.92	0.94
Geographic	1.00	1.00	1.00
Cost	1.00	1.00	1.00
Transfers	1.00	1.00	1.00
Cumulative	0.96	0.92	0.94

List of Figures



*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 1: An example network graph

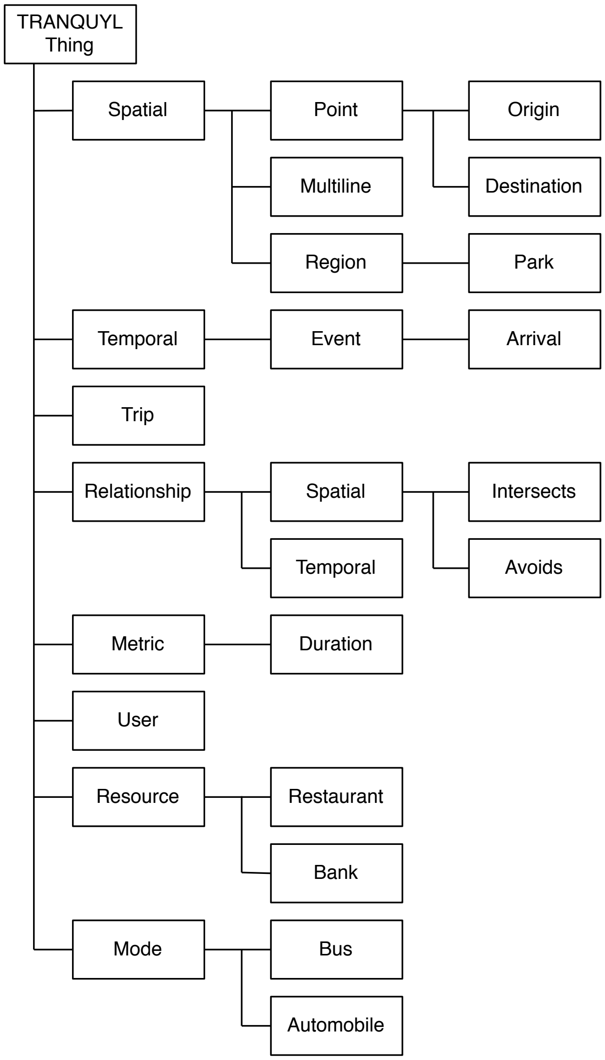


Figure 2: A small subset of the 80 classes in the TRANQUYL class hierarchy

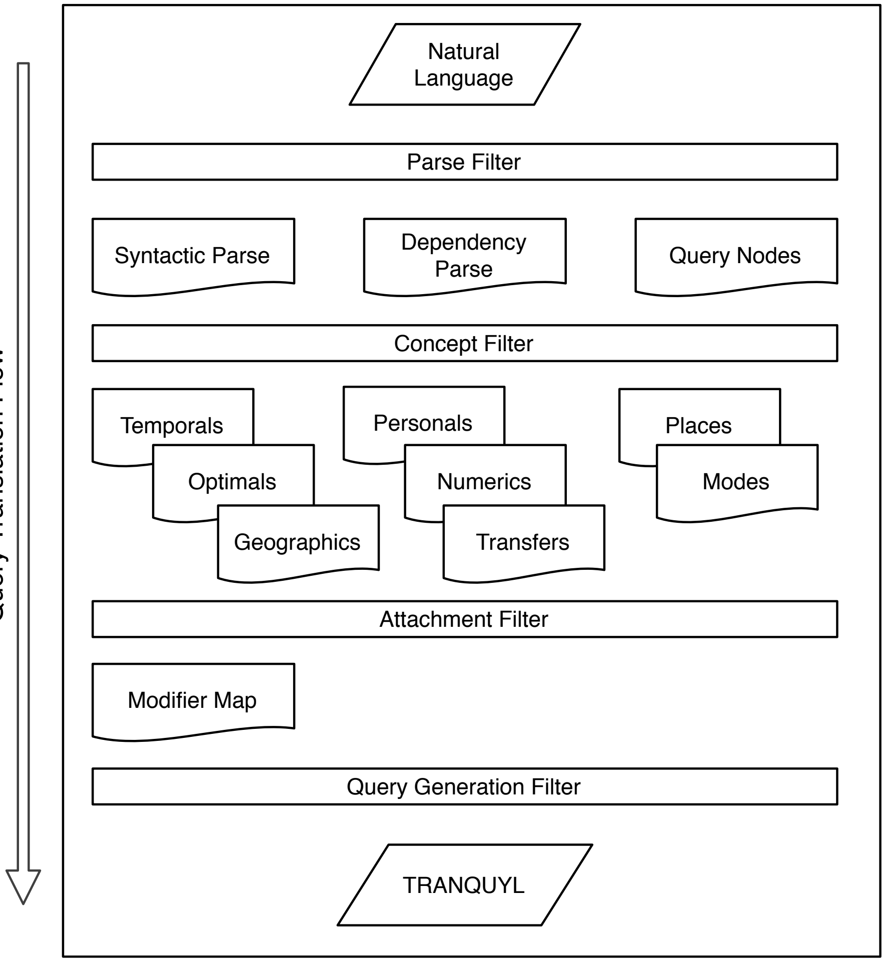


Figure 3: Overview of the NL2TRANQUYL translation pipeline

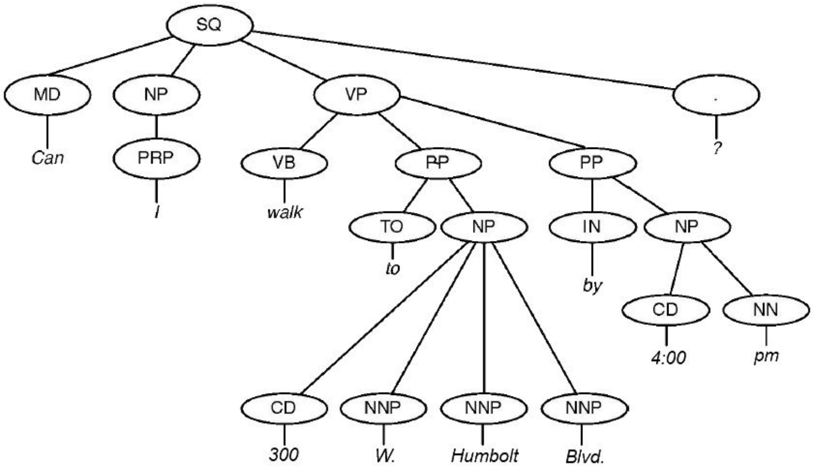


Figure 4: Example constituency parse tree

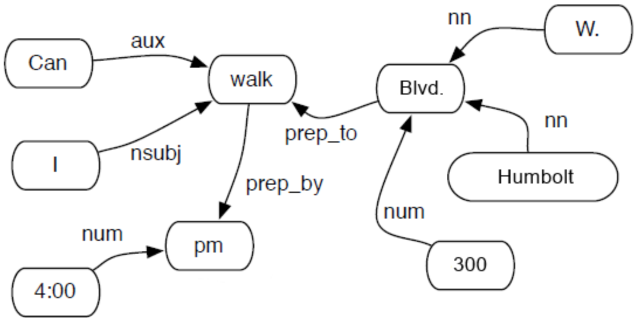


Figure 5: Example dependency parse graph

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

Figure 6: Algorithm 1: Parsing and Concept Identification

```
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
         $V_c \leftarrow V_c$ + matchScore($m, c, n$)
      end for
   end for
   $c_{max} \leftarrow \text{argmax}_c V_c$
   if $V_c > threshold$ then
      $map$.addMapping($n \rightarrow c_{max}$)
   end if
end for
return $map$
```

Figure 7: Algorithm 2: getOntologyConcepts

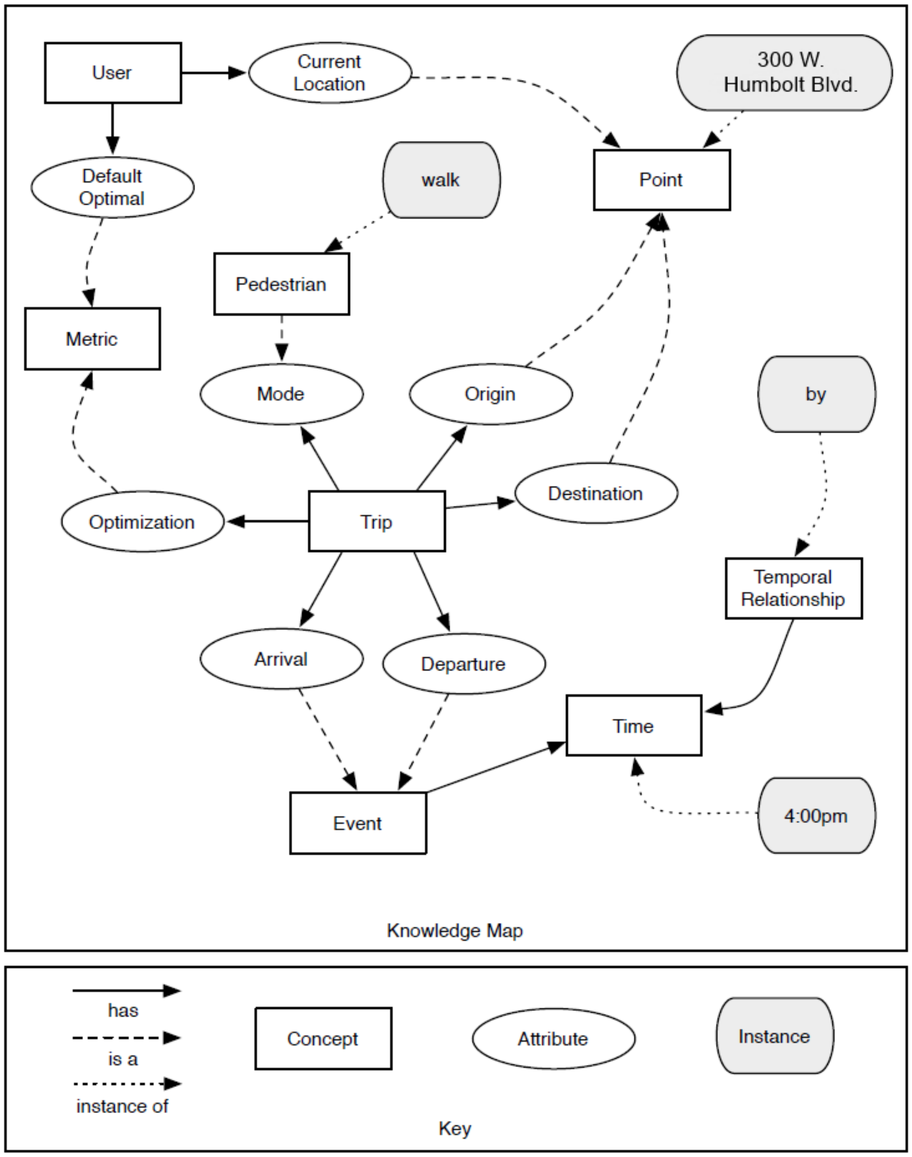


Figure 8: Example knowledge map

```
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[] $modTypes \leftarrow \{‘amod', ‘advmod', ‘nn', ‘infmod', ‘num',' prep'\}$
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 $Concept$ $c_{w[i]} \in windowConcepts$ do
            if $ont$.existsPath($c$,$c_{w[i]}$) then
               $mods$.makeConnection($c$,$c_{w[i]}$)
            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
```

Figure 9: Algorithm 3: Knowledge Map Generation

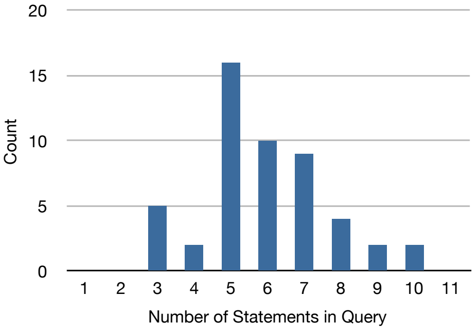


Figure 10: TRANQUYL query length distribution