TARSIUS: A System for Traffic-Aware Route Search under Conditions of Uncertainty (Demo Paper)

Itsik Hefez
Technion, Israel Institute of Technology
itsikhefez@gmail.com

Yaron Kanza
Technion, Israel Institute of Technology
kanza@cs.technion.ac.il

Roy Levin
Technion, Israel Institute of Technology
royl@cs.technion.ac.il

ABSTRACT

This demo presents TARSIUS—a system for *traffic-aware route search*. In a traffic-aware route search (TARS), the user provides start location, target location and search terms, which specify types of geographical entities that should be visited along the route. A TARS query may include additional temporal constraints and limitations on the order by which entities are visited. The goal is to find the fastest route from the start location to the target, via entities of the specified types, while taking into account variations in the travel speed, due to changes in traffic conditions. Planning a route under conditions of uncertainty requires the system to also take into account the possibility that some visited entities will not satisfy the user requirements so that the route may need to go via several entities of the same type. In the demonstration we present the system. We demonstrate a web-based user interface that facilitates the formulation of TARS queries. We show how queries are posed and evaluated over a database that contains real traffic data. Since answering a TARS query is NP-hard, we present three heuristics to the problem. Using the system, we illustrate the routes that are computed by these heuristics.

Keywords

Route search, traffic, temporal, probabilistic data

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Geographical Services such as presenting a map, searching for an address (local search) or finding a route between two locations, have become an inseparable part of the World-Wide Web (see, for example, Bing Maps, Google Maps and Yahoo! Local[1]). Such services are useful for finding geographical locations on a map. However, when initiating a geographical search, users frequently need not only to find the relevant geographical entities, but also to actually arrive at the discovered locations. In such cases, the result of a search should be a route. There are applications that can provide driving directions to a specific address. Yet, these tools are rather restricted because they are not designed to deal with complex route planning tasks where the route should go via several types of entities. Such complex task is illustrated in the following example.

EXAMPLE 1.1. *A tourist in a foreign city, say San Francisco, wants to plan a route from her current location to some destination, via four types of entities: (1) a coffee shop, (2) an ATM, (3) a shoe store, and (4) a vegetarian restaurant. She may perform four separate local searches on some geographical search engine—a different search for each type of entities. Each local search will result in a ranked list of entities, with their locations depicted on four separate maps. Yet, joining the results of the four different searches, and constructing an effective route via relevant entities, is a complex task. On the one hand, going via the entities with the highest rank is likely to increase the probability of satisfying the search requirements, however, such route may be ineffective and may travel back and forth in the city. On the other hand, choosing entities with a low rank to create a short route may reduce the chances of satisfying the user. Furthermore, taking into account traffic conditions and temporal constraints, such as a requirement to reach the restaurant at lunch time, or the need to visit an ATM before going to a coffee shop, increases the intricacy of the problem even further. Also note that a route which only goes via a single shoe store has a smaller probability to satisfy the user than a route that goes via several shoe stores, because not every shoe store has shoes that comply with the requirements (style and taste in shoes) of the user.*

Several papers have shown how to formulate route-search queries and how to answer them (e.g., [1, 5, 2, 4]), however, they did not consider traffic or time constraints. Thus, they focus on finding the shortest route.

In a Traffic-Aware Route Search (TARS) the goal is to find the *fastest route* along with the best departure time

[1]`http://www.bing.com/maps/`, `http://maps.google.com/`, `http://local.yahoo.com/`

in tasks such as the one in Example 1.1, while taking into account traffic conditions. That is, the user provides a start location, a target location, terms to specify types of entities that should be visited, and constraints. There are two types of constraints. Temporal constraints and order constraints. Temporal constraints specify limitations on the time during the day when entities should be arrived at. They may also specify an estimated stay duration at the entities. An order constraint specifies that entities of some type should only be visited after visiting an entity of some other type.

Usually, finding the fastest route is more difficult than finding the shortest route because distances between entities are constants whereas travel times vary. In many cities, the travel speed on major arteries during rush hours is significantly slower than during other hours, yet, it is difficult to calculate the effect of the traffic load, because road networks are unevenly affected by congestion [3]. The need to take into account varying traffic conditions increases the intricacy of computing a route via several types of entities more than it does for computing a route between two locations.

In a route search, the system should take into account the inherent uncertainty regarding real-world entities. Some entities may not satisfy the user. For example, the user may not find appropriate shoes in some of the visited shoe stores, a visited restaurant may not have any available table at the time of the visit, *etc.* To cope with this, the setting is modeled as a probabilistic database. The *success probability* of an entity (*probability*, for short) specifies the likelihood that a visit at the entity will satisfy the search requirements, regarding its type. For example, the probability of a shoe store is the likeliness that the user will find appropriate shoes in this store. Such probabilities can be generated based on statistics and information-retrieval methods. The generated route should go via several entities of the same type such that the probability of success (*i.e.,* the probability that at least one of the visited entities of this type will satisfy the user) will be greater than a given threshold.

TARSIUS is a system for answering TARS queries over probabilistic data. The system works as a Web application. It has a Web-based graphical user interface that facilitates the formulation of queries. It also presents the results on real-world maps using a Web browser. The system can serve both as a complete route search service for end users and as a set of independent services that can aid in generating and displaying data using an XML API.

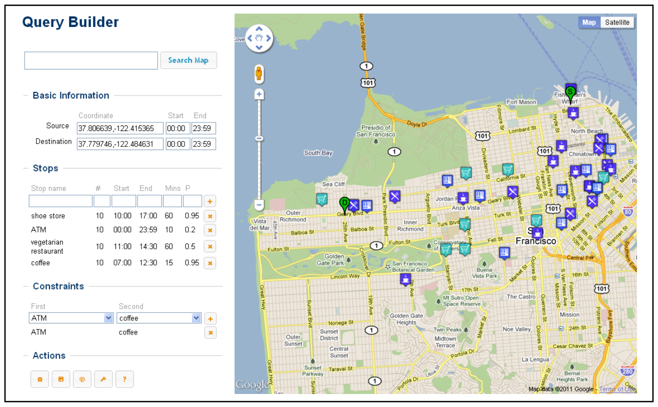
2. DATA MODEL AND QUERIES

In this section we present the data model used by the system. We also discuss TARS queries, their formulation in TARSIUS and the presentation of their results.

2.1 Data

The dataset consists of *entities* and a *road network* that connects the entities. Each entity is associated with keywords that specify its *type*. For example, some entities are associated with the keyword "ATM", other entities are associated with the keywords "shoe store", *etc.* In addition, each entity has a location, thus, an entity is presented on a map by an appropriate symbol depicted in its location.

The road network is a directed graph where the nodes represent road junctions (including intersections, traffic circles, *etc.*) and the edges represent roads. Roads are presented on a map as polygonal lines. Each edge has a *travel-time*



Basic Information

	Coordinate	Start	End
Source	37.806639,-122.415365	00:00	23:59
Destination	37.779746,-122.484631	00:00	23:59

Stops

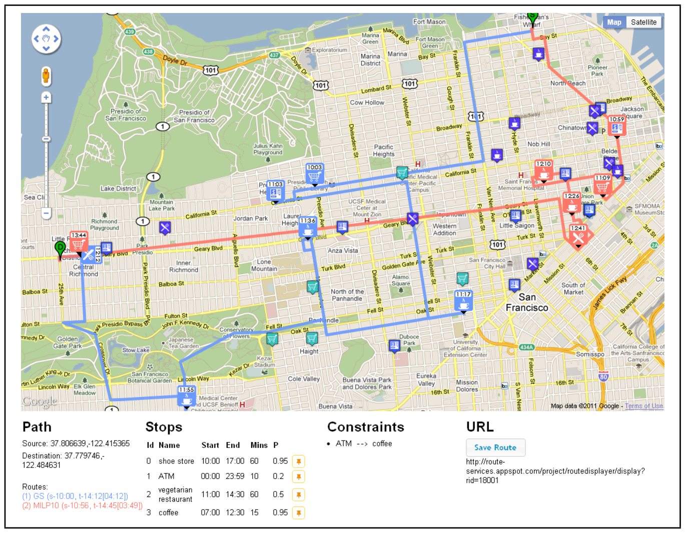
Stop name	#	Start	End	Mins	P
shoe store	10	10:00	17:00	60	0.95
ATM	10	00:00	23:59	10	0.2
vegetarian restaurant	10	11:00	14:30	60	0.5
coffee	10	07:00	12:30	15	0.95

Constraints

First	Second
ATM	coffee
ATM	coffee

Actions

Figure 1: The query of Example 1.1 issued using the query builder component.



Path

Source: 37.806639,-122.415365

Destination: 37.779746,-122.484631

Routes:
(1) GS (s-10:00, t-14:12[04:12])
(2) MILP10 (s-10:56, t-14:45[03:49])

Stops

Id	Name	Start	End	Mins	P
0	shoe store	10:00	17:00	60	0.95
1	ATM	00:00	23:59	10	0.2
2	vegetarian restaurant	11:00	14:30	60	0.5
3	coffee	07:00	12:30	15	0.95

Constraints

- ATM --> coffee

URL

http://route-services.appspot.com/project/routedisplayer/display?rid=18001

Figure 2: The routes produced by the GS and MILP algorithms when processing the query that is presented in Figure 1.

function that for each hour during the day returns the time it takes to travel along the road from the junction where it starts to the junction where it ends.

2.2 Queries

A TARS query is formulated using a graphical user interface, as depicted in Figure 1. The start location *s*, namely *source*, and the target locations *t*, namely *destination*, of the query are indicated by clicking the map or by providing appropriate addresses. The types of entities to be visited are referred to as *stops*. Each stop is defined by inserting keywords (*e.g.*, "shoe store") with their constraints. The constraints include start time and end time to specify the earliest and latest arrival time at entities of that type, the estimated stay duration at an entity and a probability threshold P. The probability threshold provides flexibility to the level of uncertainty in satisfying the user. The threshold of a stop S is near 1 when it is crucial to visit a satisfying entity of the type related to S, and it is closer to 0 when visiting a satisfying entity has a low importance. Note that increasing the threshold P may cause an increase in the number of visited entities to increase the probability of success, and vice versa. Order constraints appear as pairs of types, so that entities of the second type in the pair should only be visited after visiting an entity of the first type.

An *answer* to a TARS query is a sequence $s, e_1, \dots, e_n, t$, and a pair of functions f_a and f_d, where s is the source, t is the target location, $e_1, \dots, e_n$ are entities that refer to the specified stops, f_a and f_d assign to each entity an arrival time and departure time, respectively (except for s that only has departure time and t that only has arrival time), and such that the following conditions hold.

1. *The probability of success for each stop (type) exceeds the threshold.* That is, let $e_{i_1}, \dots, e_{i_k}$ be the entities in the sequence that refer to stop S, let $p_{i_1}, \dots, p_{i_k}$ be the success probabilities of these entities and let P be the probability threshold of S, then, the probability that at least one of the entities $e_{i_1}, \dots, e_{i_k}$ will satisfy the user should exceed P, that is, $P \leq 1 - (\prod_{j=1}^{k}(1 - p_{i_j}))$.

2. *Time constraints are satisfied.* For each entity e_i in the sequence, given that e_i refers to stop S and the time limits of S are t_1 and t_2, the *arrival time* at e_i should be between t_1 and t_2. The arrival time at an entity e_i is the sum $TT(s, e_1) + \sum_{j=1}^{i-1} TT(e_j, e_{j+1}) + \sum_{j=1}^{i-1} sd(e_j)$, where $TT(e_j, e_{j+1})$ is the travel time from e_j to e_{j+1} at the time of departure from e_j (the departure from e_j is the arrival at e_j plus the stay duration at e_j) and $sd(e_j)$ is the stay duration at e_j. That is, the travel time to e_j is the time it takes to go from s to e_1, stay for the stay duration at e_1, travel from e_1 to e_2, stay at e_2 according to the stay duration, and so on.

3. *Order constraints are satisfied.* The order of the entities in the sequence should comply with the partial order defined by the order constraints. That is, given stops S_1 and S_2 as the first and second stops of an order constraint, the entities that refer to S_2 should appear in the sequence only after the entities of S_1.

2.3 Query Evaluation

There can be many answers to a TARS query. Our goal is to find the fastest one, that is, the route for which the difference between the departure time from s and the arrival time at t is the smallest. This problem is, however, a generalization of the Traveling Salesperson Problem (TSP), and hence, it is NP-hard. Thus, we developed and implemented heuristics for it. There are currently three heuristics supported by the query processor.

GS (Greedy Search). This heuristics begins with a path that goes directly from the source s to the target t. Initially, it finds the *relevant* entity that can be added between s and t while causing the smallest increase to the travel time. An entity is considered relevant if it *(1)* complies with the search terms of one of the stops whose success probability is still below the threshold, *(2)* satisfies the order and time constraints with respect to the place in the route where it is inserted, and *(3)* it has not been visited yet. Iteratively, relevant entities are added to the partial route while minimizing in each addition the increase in travel time. This process continues until either the route satisfies the query or there are no more entities that can be added. An optimal departure time from s, according to the time limits on s, is also calculated using a greedy approach.

1-PGS (1-Pinned Greedy Search). 1-PGS is based on GS. It forces GS to consider each relevant entity as part of the answer. It does so by calling GS with an initial partial route of the form s, e, t, instead of applying GS with an initial route s, t. We refer to the entity e as the *pinned entity.* Iteratively, 1-PGS examines all the possible relevant entities as the pinned entity, and it returns the route with the minimal overall travel time among all the routes it generated for the pinned entities.

MILP (Mixed Integer Linear Programming). This algorithm uses heuristics to formulate the TARS query as a Mixed-Integer Linear-Programming problem whose solution is an answer to the query. The solution is computed using a solver, however, since solving a mixed-integer linear-programming problem is NP-Hard, the running time of the solver is limited. Even with the time limit, MILP usually provides a better answer than the previous two heuristics.

Note that a more thorough formulation and analysis of the algorithms is beyond the scope of this paper.

3. SYSTEM DESIGN

The design of TARSIUS is based on a SOA architecture in which each component is an interoperable Web service with a rigorously defined functionality. The architecture is depicted in Figure 3. We refer to the services that operate within a Web browser as the *front-end-tier* components and to those that operate outside the browser as *back-end-tier* components. The front-end services, which are in charge of handling the user interface, are mostly written in Javascript. The back-end services are implemented using the .NET Framework and are deployed on a Windows IIS Web server. Next, we provide more details on these tiers.

3.1 Front end

The front end is deployed within the Google cloud framework, (*i.e,* Google App Engine[2]). It is composed of two main components—*query builder* and *route displayer.*

The query builder provides an easy-to-use user interface for constructing queries. This component uses the Google Maps API[3] to present a map and provide interaction with it. Inserted queries can be saved, in XML format, and saved queries can be uploaded. The queries can be submitted to the server for processing. When a query should be submitted, the query builder sends to the server a list of algorithm names to apply, and a URL to indicate where the server should redirect the browser to upon completion.

The route displayer receives a query, a submap and a set of routes, in XML format. It depicts the received routes on the map using the Google Maps API. Each route is presented in a different color along with representative symbols of all the relevant entities via which it traverses. Symbols of specific entities that are visited by a route are enlarged, given the same color as the route and have their arrival time displayed.

3.2 Back end

The back end contains two components that are provided as Web services. The first is the *query processor* and the second is a *traffic repository.*

[2]`http://code.google.com/appengine/`

[3]`http://code.google.com/apis/maps/documentation/javascript/`

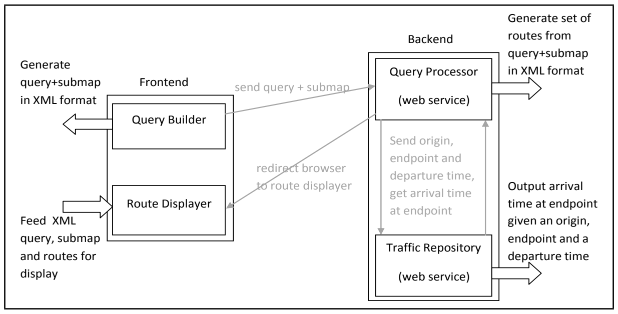

Generate query+submap in XML format
Frontend
Query Builder
Route Displayer
Feed XML query, submap and routes for display
send query + submap
redirect browser to route displayer
Backend
Query Processor (web service)
Send origin, endpoint and departure time, get arrival time at endpoint
Traffic Repository (web service)
Generate set of routes from query+submap in XML format
Output arrival time at endpoint given an origin, endpoint and a departure time

Figure 3: System architecture

The query processor computes a set of resulting routes. It receives as the input *(1)* a list of algorithm names, *(2)* a query, and *(3)* a submap. Each resulting route is annotated with the name of the algorithm that was used to produce it. The result can either be returned in the form of XML or HTML document, where the latter also includes the given query and the submap. The HTML document redirects the browser to the route-displayer component sending it the query, submap and a list of routes, each associated to the algorithm that generated it.

The traffic repository manages information on historical traffic data and provides the travel time on each road, for each departure time. This web service receives triplets of origin, target and the time of departure from the origin. It returns an estimated arrival time at the target location based on the historical traffic data. In a naive solution, the server holds for each pair of relevant entities a list of pairs, of departure and arrival times (ascending according to the departure time). The problem with such solution is that the size of the data is quadratic in the number of entities, while the number of entities is expected to be large. Hence, to improve the scalability and significantly reduce the size of the repository, we apply the following solution.

The area for which data is stored in the repository is divided into cells. The repository stores the travel times, for different times during the day, between every pair of cells. Ideally, the cells are chosen in such a way that within them the travel speed is homogeneous—that is, there is the same traffic congestion on all the roads of the cell. Now, given origin, end location and departure time, the travel time is computed by combining the travel time within the cell of the origin, the travel time within the cell of the target and the travel time between the cells. The travel time within cells is estimated based on sampled travel times in that cell, and linear interpolation (we compute the ratio of the distance between the entities to the distance between the endpoints of a nearby sampled road segment, and then we multiply the ratio by the travel time on that segment.)

4. DEMONSTRATION

The demonstration presents TARSIUS by showing the formulation of TARS queries, query evaluation and the presentation of the results for queries over a repository of traffic data, for the city of San Francisco.

To generate the data for the traffic repository, we used actual live traffic data. We used the Bing Maps API[4] to get real travel times. This API receives the locations of a source and a destination, and it returns the fastest travel time from the source to the target at the time of the search, taking into account live traffic data. We used this API to collect historical traffic data. For each pair from an arbitrarily selected chosen list of 50 entities within the city of San Fransisco, we sampled the travel time between them for different departure times. The measures were conducted in intervals of approximately 10 minutes for 24 hours.

The entities were retrieved using the Yahoo! Local Search tool. This tool not only retrieved the entities, it also ordered them according to their rank. We assigned success probabilities to the entities based on their ranking order in the search results. So, if an entity e_1 precedes an entity e_2 in the search result then e_1 is assigned a higher probability than e_2. The assigned probabilities are constructed in the range $[0.4, 0.9]$ using the distribution function $e^{-\gamma\cdot(i-1)} - (1 - p_h)$, where $\gamma = -\frac{ln(1+p_h-p_l)}{n_r-1}$, $p_h = 0.9$, $p_l = 0.4$, $n_r = 10$, and $1 \le i \le 10$ is the position of the entity in the search result. This represents a behavior that is similar to the well known "long tail" phenomenon in search.

Our demonstration illustrates typical system usages.[5] One example that can be viewed online[6] is of a simple query, containing start and end locations and a request to visit a gas station on the route between them. Such query illustrates an example of a TARS query for driving via a gas station on the way to work and spending 10 minutes there. A more intricate example is based on Example 1.1 from Section 1.[7]

We also show how to select the algorithms to be run on the server. Figure 1 illustrates the state of the query builder before redirecting the request to the query processor. Figure 2 shows the results obtained by the query processor when issuing the GS and MILP algorithms.

Finally, we show how the different SOA components can be used as standalone components: (1) when creating a collection of queries with their relevant submaps, using the query builder; (2) when comparing between different TARS heuristics, by examining the routes each heuristic produced when applied on a collection of stored queries and submaps; and (3) when using the query processor as a standalone component that generates an XML document representing an answer to a TARS query.

5. REFERENCES

[1] L. Feifei, C. Dihan, H. Marios, K. George, and T. Shang-Hua. On trip planning queries in spatial databases. In *SSTD*, volume 3633, pages 923–923, 2005.

[2] C. Haiquan, K. Wei-Shinn, S. Min-Te, and Z. Roger. The partial sequenced route query with traveling rules in road networks. *GeoInformatica*, 15:541–569, 2011.

[3] A. Hill and W. Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *JORS*, pages 343–351, 1992.

[4] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv. Interactive route search in the presence of order constraints. *VLDB*, 2010.

[5] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB journal*, pages 765–787, 2008.

[4] `http://msdn.microsoft.com/en-us/library/cc966826.aspx`

[5] See `http://db64.cs.technion.ac.il/tars/` for an online demo.

[6] `http://www.youtube.com/watch?v=j88qsPFiT-4`

[7] `http://www.youtube.com/watch?v=ZcA0xr-huSc`