

Geospatial Route Search

חיפוש מסלולים גיאוגרפיים

Ph.D. Research Proposal

Submitted by: Roy Levin

Under supervision of: Dr. Yaron Kanza

Computer Science Department
The Technion – Israel Institute of Technology
Haifa, 32000, Israel

February 24, 2010

1 Introduction

In a route-search, a user specifies his requirements in the form of a query, and the main task is to find a route that goes via geographical objects while satisfying the search specifications. In this section we present the paradigm of route search. We survey the existing state-of-the-art of the field, and we specify open problems which we intend to study in the course of our research.

1.1 The Route-Search Paradigm

Traditionally, geographic information systems were designed for expert users. In such systems, querying geographic data often required using complex formal query languages for well defined tasks. The availability of geographical data on the World-Wide Web and on mobile devices has initiated a change. This is due to the fact that services in these domains must be suited for the use of novice users and should provide answers instantly. Consequently, geographical services, such as presenting a map, searching for an address (local search) or finding a route between two locations are becoming an inseparable part of the World-Wide Web (see, for example, Google Maps, Bing Maps and Yahoo! Local¹). There is also a rapid growth in the availability of such services in GNSS-based² mobile devices such as PDAs, cellular phones and car navigation systems (which are themselves becoming inseparable). When applying a geographical search, users frequently need not only to find the appropriate geographical entities, but also to actually get to those entities. In such cases, the result of a search should be a route. Currently, users can search for geographical entities and receive driving directions to a single specific entity. For instance, a user may use a tool such as Google Maps to find a Chinese restaurant and then receive driving directions from his current location to it. However such tools are rather restrictive as they are not designed to deal with more complex scenarios, as we shall see in the following example.

Example 1 *A tourist traveling in some city wishes to visit (1) an art museum, (2) a coffee shop, (3) an ATM, (4) a shopping mall, (5) a cheap restaurant, and (6) a theater. He may open six different instances of a search tool, such as Google Maps, to perform six separate local searches for each of the required object types. Each such local search will result in a ranked list of objects and will display them on a map. However, joining the results from six different maps, into a route, is a complex task. This is because naively choosing the most highly ranked objects from each map may yield a route that travels back and forth in the city and is much longer than necessary. Moreover, there are many more factors that must be considered when constructing a route. For each of these six points of interest (POI), an effective route must account for details such as: (a) Time windows for arriving at each POI, e.g. the museum and the shopping*

¹<http://maps.google.com>, <http://www.bing.com/maps/> and <http://maps.yahoo.com>, respectively

²Global Navigation Satellite System

mall may be opened from 10:00-17:00 while the theater may only be available at night time. The user himself, may impose time windows on POIs by specifying that he wishes to eat lunch at the restaurant between 12:00 to 14:00. (b) Order constraints, e.g. the user may need to withdraw money from the ATM before reaching the theater. (c) Traffic conditions, e.g. the shortest route may not be the fastest route, however, finding the fastest route in a dynamic environment, i.e. an environment with changing traffic conditions, is difficult. (d) Multi-network environment, e.g. if the user is commuting by car he will need to find a parking lot before reaching each of these POIs. (e) Physical constraints, e.g. visit a gas station every 300 Kilometers. (f) Conditionals between objects, e.g. for some restaurants, a cash withdrawal from the ATM is necessary while others may accept credit cards. After parking his vehicle, the user needs to use sidewalks instead of roads, so his speed is reduced. The user may also wish to combine several means of transportation, such as his private car, public transportation and walking by foot.

The issues stated above are merely a subset of the considerations involved in measuring the effectiveness of a route. The route's effectiveness may also be a function of user specific preferences, i.e. finding a route that involves minimizing the expenses. A user may not desire the fastest route, but rather a route that maximizes his chance for meeting some deadline. E.g. arriving at the airport on time having completed all the prior arrangements. In such a case, arriving at the airport earlier than necessary does not provide any advantage to the user. It may also be useful to consider integrating two route queries, e.g. two friends that would like to meet at some arbitrary location and possibly continue their route together.

The example above illustrates the various difficulties that may arise when answering a complex route-search query. However, route-search is not limited to tourist scenarios. In many cases, route-search is applicable in ordinary day-to-day tasks. A person that wants to visit a gas station, withdraw money from an ATM, and visit the post office on his way to work, or a couple that desire having dinner at a fancy restaurant followed by a visit to the theater, and finding a parking spot for their vehicle, are both examples of common route-search queries.

Given datasets that consist of spatial objects representing spatial entities from the real world, we define route-search as the task of finding a list of objects, which represent a route via real world spatial entities. The route must satisfy a set of requirements that are specified by the user in the form of a route-search query. The requirements comprise constraints and goals that apply to the desired route. In scenarios where temporal constraints are involved, answering a route-search query can be viewed as planning a daily schedule, under spatial and temporal constraints, over one or more datasets. Thus, existing geographical search tools are not suitable for such tasks.

Route-search is significant in many scenarios of geographic search, and it has applications in many different areas. In addition to the cases presented above, it may be useful in cases such as evacuation of injured people, military

applications and economical application, e.g., delivering goods, a taxi searching for clients, etc.

1.2 State of the Art

In this section we present a brief summary of the research work that has been published to date on the problem of route-search. A more thorough and elaborate survey will be given in the related work section, see section (2). Note that the state-of-the-art approaches to route-search are not comprehensive in the sense that they address a rather restricted version of the route-search paradigm.

One way to answer a route-search query is by modeling the problem in terms of finding an optimal path over a weighted graph. Such types of problems have been extensively studied in the context of operational research. The best known problem of this type is the classical Traveling-Salesman Problem (TSP), that aims at finding a minimum cost Hamiltonian cycle on a given weighted graph [36]. TSP is a known NP-hard problem, and so it is unlikely to find a polynomial-time algorithm for it. There are several variations of TSP, which are also NP-hard, and have been studied thoroughly. For examples of TSP variations see [38, 8, 35, 33].

Unlike TSP, computing the shortest route between two specified locations over a road network is a task that can be accomplished optimally using fast polynomial time algorithms, e.g. the A* algorithm [24]. Recently an interest in searching, in real time, for more complex geographical routes began to emerge. This line of work can be categorized into two main types, one assuming a setting which is fully deterministic whilst the second assumes some degree of uncertainty. In both cases, the problems are formulated as variations or generalizations of the TSP. Hence, various heuristic algorithms are proposed and examined. We discuss these two approaches in the following paragraphs.

Deterministic Setting [19, 66, 18]. In this approach the dataset contains geospatial objects which represent real world entities. The objects are divided into distinct predefined sets which represent different categories, e.g. restaurants, hotels, theaters, etc. The user conducts a search over this dataset by issuing a query. The query contains a start and end locations, and a set of categories. The goal is to find the shortest route that starts and ends at the specified locations and visits exactly one object from each category. This type of problem has been studied in the context of Trip Planning Queries [19]. Note that this type of formulation is almost identical to the GTSP [38, 64, 65], it differs merely on the basis that in GTSP the start and end locations are identical. In the case where every category contains only one unique object, and the route should start and end at the same location, the problem becomes identical to the classic TSP [36]. More recent studies [66, 18] have extended this problem by adding partial order constraints. Order constraints define limitations on the route, by specifying that objects of one category are to be visited *before* objects of another, e.g. visiting an ATM before going to a post office.

Uncertain setting [61, 27, 44]. In the uncertain case the dataset contains geospatial objects which represent real world entities, however without certainty regarding the correctness of the representation. A user conducts a search over this dataset by issuing a query which includes start and end locations, and a set of k subqueries each containing search terms. The objects from the dataset are distributed into k sets, each representing a different category, which corresponds to the k subqueries. The distribution of objects to categories is based on the relevance to each of the search terms. Hence this association is uncertain. The objects in each category are assigned a value which represents the probability of the corresponding real world spatial entity to satisfy the user. For instance, for the search term "cheap restaurant" each object in the dataset is given a value representing the probability of the corresponding spatial entity to actually satisfy the user as a cheap restaurant. An answer to such a query is a list of objects representing a route that starts and ends at the given location and traverses via at least one object from each category. The probability of a route to satisfy the user is the probability of the user being satisfied with at least one object from each category. Hence, one goal is to find a route that maximizes the probability for satisfying the user. Another goal is to find a route that is as short as possible. There may be a conflict between these two goals, since a route that maximizes the probability for satisfying the user is a route that traverses via all the objects in the dataset. On the other hand, the shortest route is one that traverses via only one object from each category, and these objects are selected such that the overall distance is minimized, without considering the probability of satisfying the user. Research in the context of an uncertain settings studies different approaches for dealing with the tradeoff between maximizing the probability of satisfying the user and the need to maintain the route as short as possible. This is accomplished by examining a few alternative semantics. Some of these semantics involve finding the shortest route that visits some predefined number of objects [61] or that satisfies the user according to some predefined probability [44]. Other semantics involve maximizing the probability for satisfying the user while not exceeding some predefined route length [27, 44]. Hence in all these cases some predefined threshold is given for the purpose of resolving the tradeoff between maximizing the probability for satisfying the user, while minimizing the overall traveled distance.

The research work discussed above shows the importance of processing complex route queries in real time, and they imply that developing a practical route-search system is a feasible task. In the next section, we discuss problems that are still open with respect to route-search.

1.3 Open route search problems

The state-of-the-art is not comprehensive in the sense that there are many issues related to the route-search problem which are over simplified, or not addressed at all. In this section we will elaborate on these issues. The route-search problem can be separated into three layers, which are as follows:

1. The language layer which should allow a user to formulate his query
2. The computation layer is in charge of finding, in real time, routes that are likely to satisfy the user
3. The presenting layer deals with presenting routes to the user in a convenient manner.

Any overall solution to the route-search problem must address each of these layers. The difficulties involved in designing each of these layers is affected by cross cutting concerns. Cross cutting concerns are aspects that influence any overall solution to the route-search problem. Examples of such concerns are constraints on arrival times for different spatial entities, changing means of transportation, uncertainty about objects satisfying a user, etc. Moreover there are possible extensions to route-search. The extensions we consider in the scope of this proposal are query suggestions and advertising in the context of a route-search. Next we elaborate each of these concerns and extensions.

1.4 Route Search Layers

An overall solution to the route-search problem should properly address the three layers described above. Next we discuss the importance and difficulties in implementing each of these layers.

Language. A user wishing to express his requirements for a route, needs to use a language that is designed for that purpose. Such a language should allow a user to formulate route-search queries, that include information such as descriptions of objects to visit and restrictions applying to the actual route that traverses them. E.g. visit one object before the other, visit some object within a predefined time frame, whether traveling by car or public transportation or combining both, etc. In addition to such restrictions, the route-search language should allow the user to express his desired goals. For instance, minimizing the total travel time, or the overall cost of the route, or alternatively, maximizing the probability of successfully satisfying all the restrictions. A language for route-search must be designed to address the following requirements:

1. It should be expressive enough to allow users to express their needs, preferences and constraints.
2. It should be sufficiently structured to allow an automated system to process and cope with.
3. It should be sufficiently restrictive to allow results to be calculated within a reasonable time frame.

The question of whom the language is designed for is also of major importance. If the language is intended for an expert user, it can be a well defined formal language. On the other hand, a language for novice users, should be simple

enough to allow mastering it effortlessly. This is similar to how, on the one hand, database management systems (DBMS), which are designed for experts, use formal languages, such as SQL, for querying. On the other hand, search engines, like Google Search, use a simple, non-structured query language which can easily be mastered by novice users. Designing a proper language either for expert or novice users requires finding appropriate compromises, due to the tradeoffs between the need for simplicity and the need for high expressiveness.

Answering a route-search query. Given a dataset of spatial objects along with their various properties and a query which specifies the user's requirements, the goal is to calculate, in real time, a list of objects that represent a route via corresponding spatial entities in the real world. In fact, finding a single route may not suffice, since the user may want a few alternatives to choose from, as in modern search engines. Hence, an answer to a route-search query can be a ranked list of routes. This ranked list of routes should adhere to the requirements specified by the user. One way for the user to specify his requirements is by providing a list of constraints and a list of goals. The main difference between a constraint and a goal is that a constraint must be satisfied, e.g. reaching an entity before it closes. A goal, on the other hand, describes the ideal solution worth striving for. An example of a goal can be *finding the shortest route*, e.g. finding a route that is nearly shortest, may still satisfy the user. One problem when computing results is dealing with conflicting goals. An example of this was discussed in the previous section in the case where one goal was to maximize the probability of satisfying each of the user's requirements, and the other was to find the shortest route. Moreover, as discussed in the previous section, even when expressing a single goal, with relatively straight forward constraints, the problem of answering route-search queries is computationally as complex as the traveling salesman problem. Hence, finding a polynomial-time optimal algorithm is not likely, unless $P=NP$. The challenge is to develop real-time algorithms which satisfy the user's requirements in a best-effort fashion instead of in a guaranteed optimal fashion. There is however, a tradeoff between the complexity of algorithms and the quality of their results. Thus it is important to develop heuristics both to scenarios that require efficiency and to scenarios that require accurate results.

Presentation The manner by which the resulting routes are presented to the user is also an important issue. Merely displaying a single full-featured map with all the relevant routes depicted on it is not practical because the user would not be able to see the distinction between the different routes. The presentation of the routes should provide the information necessary for the user to decide which of the matching routes are most likely to satisfy his requirements. The user would expect the focus to be on information that is relevant to him. When we consider the aspects that affect the presentation layer, one of the problems is that each such aspect requires additional information to be conveyed. Hence filtering out redundant information while placing focus on relevant information

becomes even more important. When displaying route-search results, there are four design goals that must be taken into account: readability, clarity, completeness, and convenience. Achieving all of these goals is not a simple task and it requires trade-offs. Note that it may be important to decide, in advance, which routes are to be presented to the user. Doing so will allow optimization techniques to compute only the routes that are relevant for the display rather than computing many unnecessary routes and then filtering them out. This means that presentation issues also affect query answering.

1.5 Cross cutting concerns

There are a few crucial aspects that should be addressed in an overall solution to the route-search problem. Next we will describe some of the aspects that we intend to study. Note that some of these aspects have been studied in the past but never in the full context of the route-search problem.

1.5.1 Temporal constraints.

We limit our review of temporal constraints to two main types, as follows:

Time windows. Since, in its simplest form, an answer to a route-search query is a sequence of spatial objects representing a route in the real world, it is important to note that the corresponding spatial entities may have properties that impose restrictions on the resulting route. An important example of such properties are time windows of availability. Time windows can represent opening and closing hours of different entities. Time windows can also be imposed on spatial entities, by the user. For example the user would like to check into an hotel at night time. Moreover, the amount of time that is to be spent at each spatial entity must also be taken into account. Obviously reaching a museum five minutes before closing time is rather useless. Traveling along some route also manifests temporal constraints, as may be determined by the properties of the road network being used. E.g. a roadway network may include information on the average time required for traversing it. Note that travel times may be complex stochastic functions representing information such as traffic conditions or probabilities of delay in arrival of a train.

Order constraints. In many cases, it is necessary for a user to specify limitations on the visitation order of the desired categories. For example, a user may specify a need to visit an ATM, to withdraw cash, before reaching a post office, for the purpose of making some payment. Hence, order constraints are basically a special type of temporal constraints, e.g. category *A* must be visited before *B*. They may sometimes be derived from assigning time windows on objects to be visited. Hence, if category *A* is to be visited between 12:00 to 13:00 and category *B* between 14:00 to 15:00 an order limitation is also imposed between *A* and *B*. However, in the example of the ATM and the post office, the user wishes to make an order constraint without specifying any time windows on

neither categories. Such a need may be very common in a route-search query and needs to be addressed. As order constraints are a special case of temporal constraints, they affect the different layers of route-search for the same reasons.

Language. The route-search language should allow a user to express time window constraints and order constraints, in a convenient manner. Specifying time windows for each spatial object, separately, and providing a set of order constraints, in the form of pairs of categories is not necessarily the most convenient manner to formulate a query. Consider an example of three categories, A , B and C . Specifying that A must be visited first can be done by requiring that A be visited before both B and C . However, it may be more convenient in this case to simply state that A should be visited first. In a query containing n categories the latter formulation will save $n - 1$ restrictions that the user would otherwise need to specify explicitly.

Query evaluation. Temporal constraints add an additional dimension to the problem of route-search. When constructing a route, the travel times must be taken into account along with the availability time windows of the spatial entities along the route. Adding temporal constraints also complicates the problem. Note that finding a route that reaches all the spatial objects within their time window of availability, is a type of constraint satisfaction problem. Such problems may not have a solution at all or may have multiple solutions. Merely answering whether a solution exists or not is a problem which is NP-complete [60]. Hence, answering route-search queries in the presence of temporal constraints, may require more exhaustive search algorithms, since a simple approach may not merely be suboptimal it may not be able to find a feasible route even if such a route exists.

Presentation. When presenting a route, the time frames planned for reaching each spatial entity in the route should be conveyed in the presentation. E.g. it is desirable to express, in a convenient fashion what are the times frames of visitation planned for each spatial object along the route.

1.5.2 Uncertainty.

As in an ordinary search, the answer to a route-search query may fail to satisfy the user. Hence, we broadly refer to uncertainty here, in terms of not being able to provide a route that will satisfy the user with absolute certainty. We distinguish between three types of uncertainty as will be discussed next.

Uncertainty due to fuzzy queries. The problem is with matching spatial objects from our dataset to the user's query. The most protrude reason for this is due to the use of fuzzy terms in the user's query. For example, if the user desires to visit a fancy restaurant, how certain can we be that some spatial object indeed describes a "fancy" restaurant in reality.

Uncertainty due to a mismatch between the virtual world and the real world. The second type of uncertainty is due to inaccurate information that may be embodied in the dataset itself. This type of uncertainty is manifested as a mismatch between the objects in the dataset and the real-world entities that they represent. Such a mismatch may be the result of incorrect integration of data from heterogeneous sources. An example for this is when one source of information is used for obtaining hotel fees and another source is used for obtaining information about hotel ranking. Gathering information about objects from multiple sources is known as object-fusion. When integrating data from heterogeneous sources the object-fusion problem is not trivial, due to the lack of global identifiers [11]. The result of obtaining data about spatial objects in such a way is that the information will contain a certain degree of error [11]. In addition to object-fusion, data contained in multiple sources may be contradictory, e.g. two sources specifying a different price for an average meal as the term average meal is somewhat ambiguous.

Uncertainty due to abstractions. The third type of uncertainty is due to information that is absent in the dataset and can only be discovered upon arrival at the actual spatial entity. E.g. no vacancy at a hotel, or long lines to the cashier in a supermarket. Since the amount of information that can be contained in a dataset is limited with respect to reality, there will always be information about real-world entities, that can only be obtained by actually visiting them. This type of uncertainty is a result of abstracting the real world.

Language. When posing a route-search query, the user may wish to express how he would like to treat some of the uncertainties. For example, a user that must be on time at the airport would prefer not to follow a route with a high degree of variance in the total travel time. Hence, the language should allow the user to express various goals and constraints with respect to how to deal with uncertainty.

Query evaluation. Computing a route over data that contains a degree of error is a major challenge. The main reason for this is that the algorithm must be designed to make sensible decisions based on data which is incorrect or incomplete. It is only possible to estimate the probability of the user being satisfied with some object along the route. One element of this uncertainty has already been described in the previous section. In [61, 27, 44] they have studied a model in which an answer to a route-search query is a sequence of spatial objects which represent a route via real-world entities. As explained, there are two conflicting goals here. On the one hand there is a need to find the shortest possible route. On the other hand, there is also a need to find a route that maximizes the probability for satisfying the user. The method used for dealing with this conflict was by setting some predefined threshold which specifies the number of objects to be visited, a limit on the travel distance or a lower bound on the probability of satisfaction. The problem is that finding a proper value

for such a threshold may be difficult. Let us consider, for example, the problem of finding the route with the highest probability for satisfying the user while not exceeding an overall distance specified by a predefined threshold l [44]. The problem is that setting l too low will rule out many possible routes, and thus, force the algorithm to choose a route that has a low probability for satisfying the user. Setting l too high will yield a route that is much longer than necessary. Therefore, despite this research, the problem of resolving the conflict between minimizing the overall distance and maximizing the probability for satisfying the user remains open and requires further study.

Presentation. When presenting results, it is important to convey information about the degree of certainty pertaining to each object in the proposed route-search answer. Doing so will allow the user to make an informed decision with respect to his options. If the certainty is too low, the user may wish to issue a different query, or prefer one route over another providing that he has a choice between multiple routes. Conveying this additional information requires re-examining the tradeoffs in the design of the presentation.

1.5.3 Route-search in a multi-network environment

Another dimension of complexity that needs to be taken into account is the road network used for traversing the route in the real world. A route may vary depending on the road network that is being used. Each road network contains its own properties and constraints. For example, railroad tracks can only be used for passage when entering via a train station and within predefined time frames, which are derived from the train's schedule with a certain degree of uncertainty. E.g. information such as whether the user is driving a car or taking the train affects the resulting route. Deciding the best means of transportation to use is a matter of considering the tradeoffs between issues such as travel time convenience, cost, etc. Some of these considerations, such as convenience, for example, may be subjective, and thus, may require elaborate means of evaluation, e.g. statistical tools. One option is to let the user decide which means of transportation he prefers to use. It is also necessary, however, to let the user define his requirements using more generic terms, e.g. minimizing cost or travel time or some combination of the two, and select the best means of transportation for satisfying these requirements. Furthermore, in reality many routes are to be traversed by combinations of different means of transportation. E.g. driving by car to the train station, taking the train, then the bus, and finally arriving at the destination by foot. This too must be accounted for when querying for a route, constructing a route or when presenting a route. The following example further illustrates the problem of route-search in a multi-network environment.

Example 2 *A user wishes to spend a day at some distant city. In that city the user desires to visit a shopping mall, eat lunch at some Italian restaurant and visit a friend before returning home. The user also states that he would prefer*

to avoid traveling by bus, and that he would like to spend no more than some budget limit on this trip. For deciding which means of transportation to use, the tradeoffs mentioned earlier are relevant here. A possible choice may be to drive to a parking lot near a train station in his local city, take the train to the target city, and proceed by foot and by public transportation in the target city. Another option may be to make the entire trip by car, considering the different parking options at each point of interest. Note that in the first option, some spatial objects may be preferable to others based on their distance from the train station, while in the second option, preferable objects are those that have nearby available parking spots.

Language. The user will need to express in his query which means of transportation he wants to user, or alternatively define his goals in terms that affect this decision, e.g. specifying what he would like to maximize or minimize.

Query evaluation. A route may require changing road networks in a way that adheres to the user's constraints and goals. When commuting by car the location of the car should be an entry and exit point for the road network. It is also important to note that the amount of time the vehicle is left at a parking spot may rule out parking lots that close before the user comes up to pick up his vehicle. Moreover the time spent at a parking lot may affect the cost of the route, which may affect the decision of which means of transportation to use.

Presentation. When displaying a route, information pertaining to the road network to be used for each section of the route must be included. Such information is necessary for the user to understand by which means of transportation he should travel from one spatial entity to the next. Moreover, the user may prefer one route over another due to this information.

1.5.4 System properties

For the purpose of implementing a system capable of providing an overall solution to the route-search problem the system's running environment must be taken into account. For example, when a route-search is executed in the context of a mobile device, the user can provide feedback upon his arrival at each spatial entity. If query processing is done in a centralized server it may allow the use of statistics gathered from many users.

Language. The language should be designed in a way that suits the environment. As an example consider the design of a language suited for a mobile device versus a desktop computer. In the context of a mobile device, the language may allow the user to provide feedback upon arrival at the spatial entities. In other words, a language designed for a mobile device may be interactive differently from a language designed for a desktop computer. One of the difficulties in

designing a non-interactive language is that the user must specify all his requirements in a single query. As a result, query formulation may be too cumbersome and complex for novice users. Alternatively, a dynamic language should be designed in such a way that before the system suggests the next step, the user will provide enough information so that the final route will be reasonable in terms of satisfying the user's requirements.

Query evaluation. When designing algorithms for calculating an answer to a route search query, one distinction is between computation on the client side versus computations on the server. Matters such as the availability of computational resources, the need to process concurrent requests and the size of the dataset must be accounted for, separately, in each of these cases. For instance, when results are computed on the client side, which can also be a mobile device, efficiency is required for two main reasons. Firstly, answers should be provided promptly since the search can be initiated while the user is moving (e.g., in the case of an application in a car-navigation system). Secondly, the navigation device may have limited computation power or limited memory. Therefore, processing methods should avoid resource-intensive computations. Efficiency is also crucial when answering route-search queries on a centralized server, since the server should provide its results within a very short time period. In general, in route-search the number of possible answers can be exponential in the size of the dataset. Thus, checking all the possible routes cannot yield an efficient algorithm. Moreover, it may be desired to optimize various properties simultaneously, e.g., total travel distance and overall relevance scoring. This requires applying techniques for dealing with computationally-hard problems, such as sophisticated optimization techniques and suitable data structures. Interactivity also affects the complexity of the computation. The need to take into account the feedback options from the user increases the complexity of calculating a route. This is due to the fact that the number of possible feedbacks a user may provide is exponential in the size of the dataset.

Presentation. With respect to presenting route-search results, matters such as the size of the screen, e.g. whether a large desktop computer screen or a small screen of a mobile device, or the interface devices available to the user, be it keyboard and mouse versus a touch screen affect the manner by which results should be displayed and user feedback can be provided.

1.6 Route Search Extensions

In addition to the aspects we have just reviewed there are interesting extensions to the route-search problem. Two of which we will present next.

1.6.1 Query suggesting.

We refer to query suggesting in the context of route-search as the problem of finding a list of queries related to a query submitted by a user. The related

queries can be issued by the user to tune or redirect the route-search process. To find such a list we must take into account the user's location, in addition to the specified points of interest and his requirements from the route.

Example 3 *A user searches for a route that goes via a museum, an art gallery and a book store. The user also specifies that he is traveling by car and that he needs to be back at his hotel before nighttime. Additional points of interest such as a cathedral, a theater and perhaps a public library are related in the sense that they too are places of culture. The original query complemented by any of the above points of interest may be relevant. However, for such a query to be truly potentially relevant to our user, it must also be within the vicinity of the original route, and must satisfy the constraints defined in the query, e.g. temporal, transportation, etc.*

The main challenge is to calculate, in real time, a ranked list of route-search queries that are related to the user's query both in the contextual sense and in the geographical sense. Finding such a list is composed of finding a set of related points of interest and combining them, perhaps with the original query, in such a way that they satisfy a set of constraints and rank highly according to some evaluation function. Moreover, in some cases, an alternative query may be one that formulates the user's requirements using terms that are more generic. E.g. a user searching for a florist and a bookstore, may more generally be interested in shops that sell gift items. Therefore suggesting a related query requires knowledge pertaining to how different types of concepts are related. Furthermore, calculating a list of such related queries, in realtime, is an even harder task and presents an interesting challenge.

1.6.2 Advertising in the context of route-search.

The business model behind many online services that exist today is based on advertisement. Similarly, advertisement can serve as a solid business model for a route-search engine. Finding advertisements that may be of interest to a user performing route-search is not a simple task. It involves finding ways to compute, in real time, the best advertisements based on the user's current location, the route-search answer (hence, the objects selected to be part of the route), the points of interest specified by the user, the user's personal history, etc. Advertising in the context of route-search, is not always as simple as advertising brands of shoes to a user looking for a shoe stores. In some cases, advertisements may be selected in a more sophisticated manner. For example, advertise a nearby restaurant around lunchtime to a user that sets on a route from dawn to evening without specifying a place to eat. In addition to matching the contents of an ad to the user's query and his location, which may be rather similar to finding relevant points of interest for route recommendation, there is also the problem of selecting an ad that will maximize profit for the company the offers the route-search service, herein the provider. Maximize profit is more than just finding the ad that is most likely to interest the user, it also has to do with the fee that is to be paid by the advertiser to the provider. There may

also be a contract promising the delivery of some minimal number of ads by the provider on behalf of each advertiser. The difficulty is to compute, in realtime, a ranked list of advertisements that maximize profit for the provider, and that satisfies all the contractual constraints.

2 Related Work

In this section we will survey work that is related to the route-search problem as presented in the previous section. We will begin by discussing work that is closely related, followed by work that is related in a more general sense.

Variations and generalizations of TSP. The classical Traveling Salesman Problem (TSP), calls for a minimum cost Hamiltonian cycle on a given graph [36]. As mentioned in section (1.2), finding an answer to a route-search query is somewhat related to solving the traveling salesman problem or one of its variations. The TSP has several applications in its purest formulation or in slight modifications. These applications include planning, logistics, microchip manufacturing and even DNA sequencing. Moreover TSP is also one of the most well known NP-hard problems. As a result much work has been invested in researching algorithms and heuristics to the problem. These algorithms include:

- *Exact algorithms* which can be used to solve TSPs of up to 85,900 cities within 136 CPU years [47]. A survey of exact algorithms for TSP was done by Woeginger G. [39].
- *Approximation algorithms* for the case of symmetric TSP in a metric space have also been studied. A solution of up to a factor of $3/2$ [23] or even up to $7/6$ [12], if additional assumptions are allowed, can be reached from the optimal solution.
- *Heuristic algorithms* which include greedy algorithms and local search algorithms such as the Lin-Kernighan algorithm [67]. Many general heuristics that have been devised for combinatorial optimization, such as genetic algorithms, simulated annealing, tabu search, ant colony optimization, and others have been tried on TSP [55].

There are several other variations of TSP which are also NP-hard and have been studied thoroughly. Next, we will discuss the similarities and differences of possible semantics for route search with existing variations of TSP.

Shortest Route (SR). SR is one possible semantic presented by Kanza et al. [44]. A similar semantic is also presented in the context of answering Trip Planning Queries [19]. Under the SR semantic the goal is to find the shortest route that matches the search query. The former presents an algorithm that runs in main memory while the latter deals with the problem in the context of a spatial database. The problem of finding the shortest route is a variation of the generalized traveling-salesman problem (GTSP). In GTSP, given a partition of the nodes of a weighted graph to k clusters, the goal is to find the least-cost cycle passing through each cluster exactly once. Thus, GTSP is similar to computing the shortest route when the route begins and ends at the same spatial entity. In terms of route search, each search term specified in the search query, results in a set of matching objects (e.g., in example 1 from the introduction,

hotels, coffee shops and museums will be in three different corresponding sets). GTSP has been studied extensively over the years. It was introduced by Henry-Labordere [38, 65], and Sarivastava et al. [64] for problems that arise in computer design and in routing. Many approaches were proposed for solving GTSP, including dynamic programming [20], integer programming [49], Lagrangian relaxation [48, 57], branch-and-cut [31], genetic algorithms [68] and transforming the problem into a standard traveling salesman problem [53]. However, finding the shortest route is limited when compared to GTSP. This is due to the fact that in SR we assume that there is an edge between every two nodes and that the weights on the edges define a metric space (i.e., the weights satisfy the triangle inequality). These limitations may allow more efficient algorithms to be devised specifically for SR. SR is also somewhat related to the Generalized Minimum Spanning Tree (GMST) problem. The GMST is a generalized version of the Minimum Spanning Tree (MST) problem where the vertices in a graph G belong to m different categories. A tree T is a GMST of G if T contains at least one vertex from each category and T has the minimum possible cost (total weight or total length). Even though the MST problem is in P, it is known that the GMST is in NP. There are a few methods from the operational research and economics fields that propose heuristics for solving this problem [56]. The GMST problem is a special instance of an even harder problem, the Group Steiner Tree (GST) [72]. The main difference is that in SR the goal is to construct a route and not a tree of minimal length.

Shortest More Relevant Route (SMRR). Under the SMRR semantics, a minimal relevance threshold is given and the goal is to find the shortest route that has an overall relevance score which is no less than a given relevance threshold. This problem is somewhat related to PCTSP (Prize Collecting TSP). In PCTSP the target is to provide a route that yields a set of objects whose overall score, in our case relevance score, is no less than a given threshold, while minimizing the overall traveling distance. Balas and Martin [8] introduced the Prize-Collecting TSP as a model for scheduling the daily operations of a steel-rolling mill. Balas [6, 7] analyzed the problem from a polyhedral point of view. Fischetti and Toth [32] proposed a branch-and-bound exact algorithm based on additive bounding procedures. Bienstock et al.[14] presented an approximation algorithm with a constant bound. The main difficulty when modeling SMRR as a PCTSP problem is that in SMRR the objects are clustered into sets. These sets are the result of the object's relevance to each of the user's desired points of interest (as with SR). A feasible SMRR route consists of no more than one object from each set. Unlike with GTSP the problem is that PCTSP does not address the concept of k clusters. Therefore, it is not clear whether SMRR can be directly modeled as PCTSP.

Most-Profitable Route (MPR) [44]. Under the MRR semantics, a distance limit is given and the goal is to find the route with the highest overall relevance, under the constraint that the total distance of the route should not exceed a given distance limit. The problem of finding MRR has some similarity to the orienteering problem. In the orienteering problem, the input consists of a distance limit, a start location and a set of objects where each object has a score. The problem

is to compute a route that (a) starts at the given starting location, (b) does not exceed the given distance limit and (c) commutes via objects whose total score is maximal. The orienteering problem has been studied extensively [35] and several heuristic algorithms [69, 17, 34, 45, 52] and approximation algorithms [59] were proposed for it. As with modeling SMRR via PCTSP, due to the fact that the orienteering problem does not deal with the concept of clusters it is not clear if MRR can be directly modeled as an orienteering problem.

Additional research was also directed at solving TSP and its variations with time windows. The time windows are attached to each node in the graph indicating when the object may be visited in terms of temporality. This work is related to the temporal constraints as described in the background section. TSP (in metric space and on a line), PCTSP and OP with time windows have been studied by F. Focacci et al. [33], Bar-Yehuda et al. [10] and M. Kantor et al. [41].

In Section 1.2 we discussed recent research papers that deal with answering route related queries. Next, we will elaborate on of these papers. Li, F. et al. [19] propose *Trip Planning Queries* or *TRQ*. They use a dataset in which each point of interest is associated with a single category and they strive to compute the shortest route that passes via exactly one point of interest from each category. They show that the problem is NP-hard and present a few algorithms and analyze their approximation boundaries with respect to the optimal solution. They also discuss how one of these algorithms can perform in the context of a spatial database, e.g. using external memory. This semantic is actually equivalent to GTSP, as reviewed earlier. A very similar semantic called *the optimal sequenced route query* or *OSR*, has been proposed by Sharifzadeh et al.[66]. The semantic is identical to TPQ only with full order constraints imposed on the categories to be visited, e.g. visit a category *A*, then *B* and finally *C*. Unlike TPQ, finding an optimal answer to a OSR query is not NP-hard, it can be done using Dijkstra’s algorithm. The authors present heuristic algorithms for vector and metric spaces which run faster than Dijkstra’s algorithm. Chen et al. [18] later present the *multi-rule partial sequenced route (MRPSR) query* which unifies TPQ and OSR. They also present an improvement to the previous algorithms which is based on the A* algorithm.

Safra et al. [61] consider a problem of finding a short route via *k* location objects in an uncertain geospatial dataset. They define a probabilistic model that represents real world spatial entities and examined three heuristic algorithms that find routes via *k* desired location objects within the uncertain dataset. They show that the runtime of their algorithms is anti-proportional to the quality of their results. Nir Dolev et al. [27] propose a different semantic for route-searching in an uncertain dataset of spatial objects. They consider a semantic where the aim is finding a route that starts at a given location and traverses through as many correct objects as possible without exceeding a pre-defined distance threshold. This problem is a generalization of the Orienteering Problem (OP) which has been reviewed above. They presented three efficient OP route-search heuristics which attempt to lead the constructed route towards clusters of objects as soon as possible, under the assumption that clusters al-

low collecting many objects that are likely to be correct, from a small area. Kanza et al. [44] introduced the paradigm of a route-search query and propose three semantics for it. The first semantic discussed is finding the *shortest-route*, which is basically the same as TPQ and GTSP. They compare three efficient algorithms, namely GExt, GIns and IFH and conclude that GExt is the most efficient algorithm among the three, whereas IFH provides better results than the other two algorithms. The second semantic they discuss is the *most-reliable route*. In this case a distance limit l is given and the goal is to find a route that has the highest minimal score which doesn't exceed l . They devise and compare two algorithms for realizing this semantic.

As mentioned, the work reviewed in this section is relevant to finding an answer to a route-search query. However when combining all the different aspects discussed in Section 1.3 such as uncertainty, temporality, multi-network environment, etc. the problem changes all together. Hence, the current state of art does not offer an overall solution to the problem of answering route-search queries.

Simple queries over digital maps. Another field of research that spawned as a result of the growing popularity of online mapping services such as Google Maps and Bing Maps, has to do with answering queries over road networks. This involves responding, in real time, to queries such as finding shortest routes between locations along a spatial network, as well as finding an object's k nearest neighbors, from a given set (e.g., gas stations, markets, and restaurants), where the distance is measured in terms of paths along the network [22, 58, 62, 63]. Balasubramanian [9] studied the problem of finding the shortest route between two given locations in a multi-geography environment. In a multi-geography environment, geographical information may be spread over multiple heterogeneous interconnected maps. They offer a method that achieves efficiency and scalability which are better than what has been achieved in a single-geography environment.

This research work may prove useful for efficiently calculating the answer to a route-search query. This is because calculating the overall distance of a route requires calculating distances between sets of nodes. The algorithms for nearest neighbors accomplish this task efficiently. Note, however, that this is a limited problem with respect to finding an optimal route-search answer.

GIS query languages. Over the years, several query languages for geographical data were proposed. The proposed languages can be classified into the following three categories: Natural languages, Formal languages and Visual languages.

Natural languages allow users to express their queries without the need to learn a new query language. It seems to be the most suitable approach for the end-user. However, the difficulty of this approach is that many ambiguities and errors may occur. For these reasons, actual implementations of natural-language interfaces for GIS have been very limited [15].

Formal languages, several extensions to traditional relational query languages, such as SQL and Quel, have been proposed [13, 25, 28]. These extensions were made in order to allow Database Management Systems (DBMS) to store and retrieve spatial information. However, these languages are not designed for non-expert users due to lack of conviviality. Furthermore, these query languages are aimed at providing complete and exact answers and, in some cases, are not suitable when answers must be provided instantly.

Visual languages. Languages of this category were developed to allow users to pose queries over geographical data in a graphical way. To date, two main approaches have been developed in designing visual query languages for GIS. In the first approach, the user draws the query directly on the screen, using the blackboard metaphor (See for example, Spatial-Query-By-Sketch [29]). In the second approach, the user uses commands and buttons to display metaphorical patterns (icons or symbols) in a command user interface. Important examples of this approach are the Cigales language [54, 16] and the Lvis language [4, 3]. Hybrid approaches such as GeoPQL [30] have also been proposed. Visual languages for GIS are well suited for the end-user as they offer an intuitive and incremental view of spatial queries. However, these languages provide only a limited expressive power, their efficiency is low and they are sometimes ambiguous, thus inferior, in this aspect, to formal languages [30].

The type of queries that can be handled by these languages may be of (1) alphanumerical type (retrieving information without using geometry), (2) geometrical or topological type and (3) deductive type. These queries may be very complex and, in most cases, involve the combination of several geometrical or topological operators (conjunction and/or disjunction of operators). The following three queries may serve as a representative example of these types: [46]

Q1: What are the plots of urban zones situated in a protection zone of an historical monument?

Q2: What are the urban zones including a natural zone?

Q3: What are the plots included in an urban zone and in an agricultural zone, not located in a protection zone but near a sewerage, and where the sound pollution does not exceed 20 db?

All the languages discussed above offer different approaches for posing queries over geographical datasets. Some of them can also be used to describe how results should be presented. However, none of them address the issue of specifying a need for a route or providing a route as a result of a search query.

Route presentation. Related work also exists in the field of displaying routes. Route maps depict a path from one location to another. They are a powerful tool for visualizing and communicating directions. Although creating a route map may seem to be a straightforward task, the underlying design of most route maps is quite complex. Mapmakers choose which information is most essential for following the route and they use a variety of cartographic generalization

techniques including distortion, simplification, and abstraction to emphasize this essential information. Agrawala and Stolte [1, 2] propose methods for effectively creating computer-generated route maps. Several of these algorithms have been implemented into LineDrive, a real-time system for automatically designing and rendering route maps. LineDrive is part of mapblast, a Web mapping service which was acquired by Microsoft and incorporated into MSN maps and directions. Recently, route maps in the form of driving directions, have emerged as one of the most popular applications on the Web. See for example Google Maps and Bing Maps. In contrast to hand-designed route maps, these computer-generated route maps are more precise and contain more information. Yet, in general, they are also more difficult and frustrating to use as they do not distinguish between essential and extraneous information. Hansen et al. [37] have studied the cognitive and ergonomic aspects of giving route directions. Dale et al. [26] suggest using natural language generation methods for automatic route description. Nevertheless, displaying route search results is different from displaying route maps in a few aspects. Firstly, in route search there are many routes to be displayed and not just a single route. Secondly, the routes displayed in route maps are routes from a single source to a single destination; this is different from displaying a route that travels via multiple optional destinations as in the case with route search.

GIR. In the area of geographical search, the availability of geographical information on the World-Wide Web raised a growing interest in research on Geographic Information Retrieval (GIR). Similarly to classic IR (Information Retrieval) GIR also deals with answering search queries over a repository of textual documents. GIR extends classic IR by adding the concept of location to documents and then using it for improving the relevance ranking of search results [40, 50, 51]. The domain of problems tackled by GIR is different from that of route search in two main respects. Firstly, in GIR the goal is to retrieve a set of relevant textual documents and not matching routes. Secondly, in GIR the queries are performed over a repository of documents or over the entire Web and not over specific maps.

Query suggesting. Suggesting alternative queries to a user performing search has been studied extensively in the context of information retrieval. One form of query suggestion is called query expansion (QE). Due to the fact that people often use different words to describe concepts in their queries than authors use to describe the same concepts in their documents. Hence, by reformulating the user's seed query, it is possible to improve results [71]. More recent papers have studied a personalized version of the problem within the context of a search engine on the Web [21]. Query expansion usually involves using synonyms, different morphological forms or spelling correction, to improve the search results.

A different method for suggesting alternative queries is query recommendation (QR). Given a query submitted to a search engine, the idea is to suggest a

list of related queries. The related queries are based in previously issued queries, and can be issued by the user to the search engine to tune or redirect the search process. The method for suggesting the related queries is usually based on a clustering process over data extracted from the query log [5, 73].

These problems are within the domain of IR, and hence, are fundamentally different from suggesting alternative routes to a user. This line of research does not deal with the user's location nor with the fact that each suggested query should yield a route that meets the user's requirements in some way.

Constraints satisfaction. Constraint satisfaction problems or CSPs are mathematical problems defined as finding a value assignment for a set of variables such that these values satisfy a number of constraints or limitations defined using those variables [60]. This is relevant to route-search since a route that satisfies a set of constraints defined in the user query is required. In its full generality, constraint satisfaction is an NP-complete problem [70]. Hence, methods that can solve such problems generally cannot be applied to efficiently find an answer to a route-search query. The problems reviewed in this section in the context of TSP and its variations, are *constraint-optimization problems* or COPs which are similar to CSPs. The difference is that with COPs the constraints must be maintained while satisfying some objective function as well. Route-search is more closely related to COP than to CSP.

3 Research Objectives

Our objective is realize the route-search paradigm as described in the introduction. In the next subsections we present our plans and objectives.

3.1 Layers

We consider the problem of route-search as composed of three separate layers.

- The language layer for allowing users to issue expressive route-search queries.
- The algorithmic layer for calculating an answer to a route-search query.
- The presentation layer for presenting route-search results.

We now present our plans for dealing with these layers.

Query Language. The first question pertaining to the route-search query language is the type of user it is designed for. A query language designed for expert users should be formal and expressive, whilst when designed for a layman user, the query language should be simple and flexible. We intend to focus mostly on a query language for layman users, since it seems that most of the natural domains in which route-search can be applied in are those servicing layman users. Therefore our efforts will focus on designing a query language which is easy to master yet powerful in terms of expressiveness. In the open problems section (see Section 1.3), we reviewed two types of query languages. The first is interactive, e.g. it is applied in the context of a mobile device and it allows the user to specify requirements pertaining to his route while being on the move. The second is non-interactive and requires the user to specify all his requirements before starting his travel. An interactive language will be simpler to master by a novice user since there is no need to specify all requirements in advance. As a result there is no need to plan the route to its details in advance, but rather express requirements in a more agile manner. This approach will add additional uncertainty to the model, since the query being issued is incomplete. Another issue which is due to queries being issued by layman users is that the query may be incomplete or use fuzzy predicates such as *near*, *easy to reach* or *cheap*. This requires developing an adequate probabilistic model. We intend to develop a mechanism that will interpret fuzziness and incompleteness to probabilistic measures. The probability assigned to each spatial object should properly represent the the likelihood of the user being satisfied with the corresponding real-world spatial entity. This mechanism can make use of probabilistic dependencies between objects. The probability assigned to an object may change when some fact is revealed about another object. For example, the probability that there are available rooms in one hotel may be affected by information about the vacancy in neighboring hotels. The probability of a route satisfying a user should be a function of the probabilities assigned to each of the

objects along the route, but also account for requirements not yet stated by the user. When providing a route, there is a need to make probabilistic inferences about requirements that have not been stated from requirements which have been. Such inferences can be made by examining statistical data pertaining to mutual information between different types of requirements generally made by users.

Query Answering. In this line of research, our objective will be to construct algorithms for answering route-search queries. As in an ordinary search, it is usually desired to provide to users the ability to choose from different options. One way to do so is to return several routes as the result of a search. A different approach is to allow users to provide some input on spatial entities being visited, in an interactive fashion. A route can thus be constructed more effectively based on such input. The algorithms we intend to study, are to be applied over a probabilistic model which encompasses the different uncertainties pertaining to the dataset and the user’s query. To improve the runtime performance of our algorithms, we intend to examine methods for using buffers and caches for storing information that may speedup query answering. The information may include precomputed routes according to the search history or pre-calculations pertaining to certain key point locations that routes usually contain. After implementing our algorithms, we intend to measure their effectiveness by testing them over real-world data and synthetically-generated data (when real data is not available). The comparison will be done using benchmarks that we will develop in the course of our research. In this respect, one of our goals is to construct an exact algorithm, that will be used to find an optimal solution with respect to a model of the route-search problem. As discussed in previous sections, any reasonable modeling of route-search results in a problem which is NP-hard. Therefore, we expect an exact algorithm to run in exponential time. As a result, such an algorithm can only be run on very small datasets. It still may be useful as a benchmark for comparing other algorithms to it, and measuring their effectiveness. Hence, by comparing the results of a fast heuristic algorithm to those of an exact algorithm by running them both on a small dataset we can determine a measure for the effectiveness of the heuristic algorithm.

Visualization. Displaying route search results is, in many aspects, more complicated than displaying the results of an ordinary Web search. On the one hand, each route should be displayed in a manner which is descriptive enough to allow the user to actually navigate his way in the real world. On the other hand, since the number of routes to display is likely to be very large, displaying too much information for each route will overload the user with too many details and will be difficult to follow. Hence presenting all the routes that satisfy a single comprehensive query does not seem to be a reasonable solution. Therefore, we plan to use an interactive approach. Instead of displaying all the possible routes on a single map and having the user examine and select the route that best meets

his requirements, upon reaching some entity, the user will be given relevant information that will assist him to choose the next entity to visit. Deciding which information to display is not straightforward even when using the interactive approach, however, it is not as difficult as presenting, in a convenient fashion, all necessary information on all the feasible routes. Our research will be aimed at studying methods for interactively presenting route-search results.

3.2 Aspects

The aspects discussed in Section 1.3 affect the design of the above layers. We intend to start with a simplified version of the route-search problem. Once we find a satisfactory solution to it and properly design the three layers accordingly, we can introduce another aspect from the open problems. By doing so we are defining a problem that more accurately represents the full-featured route-search problem, as presented in the first section. Hence, by continuing with this iterative process of gradually offering a solution to a route-search problem which contains more and more aspects from the open problems, we are bound to come closer to realizing the route-search paradigm. We plan to begin by examining the route search formulation proposed by Kanza et al. [44]. Next we intend to proceed in the following order:

1. Find a better way to deal with uncertainty
2. Add order constraints between categories that are to be visited
3. Add time windows of availability to every object in the dataset
4. Deal with a multi-network environment

Next we elaborate on each of these steps.

3.2.1 Dealing with uncertainty.

Route search in an uncertain setting was described in Section 1.3. As mentioned, to date, the approach for maximizing the probability of success was to use a lower or upper bound thresholds. We plan to resolve this tradeoff in a novel manner. Instead of providing a full route that satisfies the query, the user is given a route in an interactive manner, based on feedback he provides for each visited spatial entity. The assumption here is that the user is carrying the search device with him, e.g. he is conducting the search using a mobile device. The feedback relates to whether or not the user is satisfied with the real-world spatial entity represented by the corresponding spatial object. If the user is satisfied, he can proceed with the route as planned. Otherwise the route can be reconstructed dynamically. A route can be evaluated solely based on the overall distance. Once the user reaches the end of the route, it can be evaluated based solely on the overall travel distance. Calculating the shortest route that takes all possible user feedback into account is computationally difficult because the number of possible feedbacks is exponential in the size of the dataset. Hence,

for this task we intend to develop interactive heuristic algorithms. Interactivity allows processing to take place while the user is on his way to his next entity. Since the travel time may be rather significant, it allows for algorithms that are more exhaustive to be applied during this period. In addition, to show the effectiveness of this approach, we intend to compare the overall distances of routes that have been computed offline, e.g. without user feedback, to routes that have been computed interactively and show that the interactive approach provides shorter routes.

Order constraints. The first step in dealing with temporality is allowing users to express order constraints. We intend to study an interactive route-search that uses order constraints for specifying by which order geographical entities should be visited. Our assumption, in this case, is that the number of categories a user may request in his query is constant, i.e. it is proportional to the size of the query and is bound by a constant. Hence, applying an exhaustive algorithm that selects the next object to be visited by examining every feasible permutation of categories, should yield a reasonable solution in terms of time complexity.

Time windows for visiting objects. A further extension can be achieved by allowing time windows of availability to be given for each spatial object. Such time windows may be part of the information of the dataset or they may be specified by the user. We intend to enrich route-search with order constraints, to deal with time constraints, as well. This means enriching the query language to allow a user to specify time windows for visiting objects and to extend the algorithm to deal with interactive route-search when both order and time constraints are present. Adding time constraints increases the complexity of the route-search problem and requires developing novel algorithms.

Multi-network environment. A multi-network environment is a combination of several road networks. Each road network should properly represent all the relevant information about its roads. For instance, a railroad network should include information about the time windows of availability of the different trains and possibly information about the probability of a delay. We plan to define a structure for each such road-network and to interconnect them in specific nodes. For example, a parking lot represents a possible interchange from a roadway network to a network of sidewalks. Constructing an algorithm that keeps track and makes use of information such as where the vehicle was last parked, is more difficult and will require additional study.

3.3 Route-search extensions.

We presented two possible extensions to the route search problem. One was finding ways to suggest alternative routes to help the user tune his search, and the other was finding suitable advertisements to present to a user conducting a

route-search. These ideas are in a rather preliminary state and we have yet to devise a clear approach for them.

3.4 Building a system.

To demonstrate an overall solution to the route-search problem, we also plan to build a route-search framework. Building a framework has four main goals. First, the implementation will provide an inclusive affirmation to the practicality of our approach. Second, the system will provide a platform for testing the effect of different data structures on the performances of our algorithms. Third, as part of the system we will examine ways of using buffers and caches for storing in memory and on the disk information that may speedup query answering. Fourth, a system can serve as a convenient platform for additional future research in this field.

4 Preliminary Results

With respect to the plan laid out in the previous section (see Section 3.2.1), our preliminary work involves resolving the tradeoff between finding a route that maximizes the probability for satisfying the user with respect to his query and minimizing the overall route traveled distance. As mentioned, this conflict is resolved by adopting an interactive approach to route search. Our results include the publication of *An Interactive Approach to Route Search* [42] (see appendix A), followed by a complementing work which introduces order constraints [43] (see appendix B), as described in Section 3.2.1. The latter will be submitted for review. We also began working on further extending the problem to include time windows of availability for each spatial object (see appendix C), as discussed in Section 3.2.1. For further details about our results with respect to the work above we refer the reader to the appendix of this research proposal.

References

- [1] M. Agrawala and C. Stolte. A design and implementation for effective computer-generated route maps. *In AAAI Symposium on Smart Graphics, Stanford*, March 2000.
- [2] M. Agrawala and C. Stolte. Rendering effective route maps: improving usability through generalization. *In Proceedings of Siggraph, Los Angeles, CA*, 2001.
- [3] M.A. Aufaures-Portier. A high level interface language for GIS. *Journal of Visual Languages and Computing*, Vol. 6, N. 2:167–182, 1995.
- [4] M.A. Aufaures-Portier and C. Bonhomme. A high level language for spatial data management. *Third Int. Conf. on Visual Information Systems*, LNCS N. 1614:325–332, 1999.
- [5] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. *In International Workshop on Clustering Information over the Web (ClustWeb, in conjunction with EDBT), Creete, Greece*, pages 588–596, 2004.
- [6] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
- [7] E. Balas. The prize collecting traveling salesman problem: Ii. polyhedral results. *Networks*, 25:199–216, 1995.
- [8] E. Balas and G. Martin. Roll-a-round: Software package for scheduling the rounds of a rolling mill. *Copyright Balas and Martin Associates, 104 Maple Heights Road, Pittsburgh, PA*, 1986.
- [9] V. Balasubramanian, D. V. Kalashnikov, S. Mehrotra, and N. Venkatasubramanian. Efficient and scalable multi-geography route planning. *13th International Conference on Extending Database Technology, EDBT*, 2010.
- [10] R. Bar-Yehuda, G. Even, and S. Shahar. On approximating a geometric prize-collecting traveling salesman. *In Proceedings of the European Symposium on Algorithms*, 2003.
- [11] C. Beeri, Y. Kanza, E. Safra, and Y. Sagiv. Object fusion in geographic information systems. *In Proceedings of the 13th International Conference on Very Large Data Bases Toronto (Canada)*, page 816827, 2004.
- [12] P. Berman and M. Karpinski. 8/7-approximation algorithm for (1,2)-tsp. *Proc. 17th ACM-SIAM SODA*, ECCS TR05-069:641–648, 2006.
- [13] R. Berman and M. Stonebraker. Geo-quel: a system for the manipulation and display of geographic data. *ACM SIGGRAPH Computer Graphics*, 11(2):186–191, 1977.

- [14] D. Bienstock, M. Goemans, D. Simchi-Levi, and D. Williamson. Math. program. In *B.L. Golden and A.A. Assad, editors, Vehicle Routing: Methods and Studies, North-Holland, Ser. A*, 59:413–420, 1993.
- [15] G. Cai, H. Wang, and A. MacEachren. Communicating vague spatial concepts in human-gis interactions: A collaborative dialogue approach. *Conference on Spatial Information Theory, Kartause Ittingen, Switzerland*, pages 304–319, 2003.
- [16] D. Calcinelli and M. Mainguenaud. Cigales, a visual language for geographic information system: the user interface. *Journal of Visual Languages and Computing*, Vol. 5, N. 2:113–132, 1994.
- [17] I. Chao, B. Golden, and E. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88:464–474, 1996.
- [18] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The multi-rule partial sequenced route query. *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, 2008.
- [19] F. Cheng, D. Hadjieleftheriou, M. Kollios, and G. Teng. On trip planning queries in spatial databases. *Proceedings of the 9th International Symposium on Spatial and Temporal Databases*, 3633:273290, 2005.
- [20] A. G. Chentsov and L. N. Korotayeva. The dynamic programming method in the generalized traveling salesman problem. *Mathematical and Computer Modeling*, 25(1):93–105, 1997.
- [21] A. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *the Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 7–14, 2007.
- [22] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. *VLDB, Trondheim, Norway*, pages 865–876, September 2005.
- [23] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1976.
- [24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.
- [25] G. Costagliola, G. Tortora, M. Tucci, and M. Busillo. Gisql - a query language interpreter for geographical information systems. *IFIP Third Working Conference on Visual Database Systems, Lausanne, Switzerland*, pages 247–258, 1995.

- [26] R. Dale, S. Geldof, and J.-P. Prost. Using natural language generation in automatic route description. *Journal of Research and practice in Information Technology*, 37(1):89–105, 2005.
- [27] N. Dolev, Y. Kanza, and Y. Doytsher. Efficient orienteering-route search over uncertain spatial datasets. *In FIG Working Week - Integrating Generations, Stockholm (Sweden)*, June 2008.
- [28] M. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.
- [29] M. Egenhofer. Query processing in spatial-query-by-sketch. *Journal of Visual Languages and Computing*, Vol. 8, N. 4:403–424, 1997.
- [30] F. Fernando and R. Maurizio. Resolution of ambiguities in query interpretation for geographical pictorial query languages. *Journal of Computing and Information Technology*, CIT 12, N. 2:119–126, 2004.
- [31] M. Fischetti, J. Salazar-Gonzalez, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.
- [32] M. Fischetti and P. Toth. An additive approach for the optimal solution of the prize-collecting travelling salesman problem. *In B.L. Golden and A.A. Assad, editors, Vehicle Routing: Methods and Studies, North-Holland*, pages 319–343, 1988.
- [33] F. Focacci, M. Milano, and Andrea Lodi. Solving tsp with time windows with constraints. *Proceedings of the international conference on Logic programming, Las Cruces, New Mexico, United States*, 4(2):p.515–529, November 1999.
- [34] B. Golden, Q. Wang, and L. Liu. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics*, 35:359–366, 1988.
- [35] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval research Logistics*, 34:307–318, 1987.
- [36] G. Gutin and A.P. Punnen (eds.). *The Traveling Salesman Problem and Its Variations*. 609-662, 2004.
- [37] S. Hansen, K.-F. Richter, and A. Klippel. Landmarks in opens - a data structure for cognitive ergonomic route directions. *4th International GI-Science Conference, Germany*, 2006.
- [38] A. Henry-Labordere. The record balancing problem - a dynamic programming solution of a generalized traveling salesman problem. *Revue Francaise D Informatique DeRecherche Operationnelle*, 2:43–49, 1969.

- [39] Woeginger G. J. Exact algorithms for np-hard problems: A survey. *Combinatorial Optimization Eureka, You Shrink! Lecture notes in computer science*, Springer, 2570:185207, 2003.
- [40] C. B. Jones, A. I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The spirit spatial search engine: Architecture, ontologies and spatial indexing. *In Proceedings of the 3rd International Conference on Geographic Information Science, Adelphi (MD, USA)*, pages 125–139, 2004.
- [41] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 4(2):629–635, 1992.
- [42] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv. An interactive approach to route search. *ACM SIGSPATIAL GIS*, 2009.
- [43] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv. Interactive route search in the presence of order constraints. *Technical report <http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2010/CS/CS-2010-02>*, 2009.
- [44] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher. Heuristic algorithms for route-search queries over geographical data. *16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems Irvine, CA, USA*, 2008.
- [45] P. Keller. Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, 41:224–231, 1989.
- [46] J. Kennedy and P. Barclay. A survey of query languages for geographic information systems. *Proceedings of the 3rd International Workshop on Interfaces to Databases, Napier University, Edinburgh, 8-10 July 1996*, July 1996.
- [47] Applegate D. L., Bixby R. E., Chevval V., and Cook W. J. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, ISBN 978-0-691-12993-8, 2006.
- [48] G. Laporte, H. Mercure, and Y. Nobert. Finding the shortest hamiltonian circuit through n clusters: A lagrangian approach. *Congressus Numerantium*, 48:277–290, 1985.
- [49] G. Laporte and Y. Nobert. Generalized traveling salesman problem through n-sets of nodes - an integer programming approach. *INFOR*, 21(1):61–75, 1983.
- [50] R. Larson and P. Frontieria. Geographic information retrieval (gir): searching where and what. *In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield (UK)*, page page 600, 2004.

- [51] R. Larson and P. Frontiera. Spatial ranking methods for geographic information retrieval (gir) in digital libraries. *In Proceedings of the 8th European Conference on Research and Advanced Technology for Digital Libraries, Bath (UK)*, pages page 45–56, 2004.
- [52] A. Leifer and M. Rosenwein. Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73:517–523, 1994.
- [53] Y. Lien, E. Ma, and B. W. S. Wah. Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem. *Information Sciences*, 74(1-2):177–189, 1993.
- [54] M. Mainguenaud and M.A. Protier. Definition of cigales: a gis query language. *Int. Conf. DEXA, Springer Verlag Publ.*, pages 275–280, 1990.
- [55] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, ISBN 3-540-22494-7, 1998.
- [56] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the generalized minimum spanning tree. *Problem. Networks*, 26:231241, 1995.
- [57] C. E. Noon and J. C. Bean. A lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):623–632, 1991.
- [58] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. *VLDB, Berlin, Germany*, pages 802–813, September 2003.
- [59] R. Ramesh, Y. Yoon, and M. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2):155–165, 1992.
- [60] Dechter Rina. *Constraint processing*. Morgan Kaufmann, ISBN 1-55860-890-7, 2004.
- [61] E. Safra, Y. Kanza, N. Dolev, Y. Sagiv, and Y. Doytsher. Computing a k-route over uncertain geographical data. *In Proceedings of the 10th International Symposium on Advances in Spatial and Temporal Databases (SSTD), Boston (MA, USA)*, 4605:276–293, 2007.
- [62] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.
- [63] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. *In SIGMOD’08 Vancouver, BC, Canada*, 2008.
- [64] S. S. Sarivastava, S. Kumar, R. C. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *Journal of the Canadian Operational Research Society*, 7:97–101, 1969.

- [65] J. P. Saskaena. Mathematical model for scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, 8:185–200, 1970.
- [66] M Sharifzadeh, M Kolahdouzan, and C Shahabi. The optimal sequenced route query. *The VLDB journal*, pages 765–787, 2008.
- [67] Lin Shen and Kernighan B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498516, 1973.
- [68] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational research*, 174:38–53, 2006.
- [69] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.
- [70] Moshe Y. Vardi. Constraint satisfaction and database theory: a tutorial. *Symposium on Principles of Database Systems archive Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART*, pages 76 – 85, 2000.
- [71] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. *In the Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11, 1996.
- [72] Wu Bang Ye and Chao Kun-Mao. *Spanning Trees and Optimization Problems*. Chapman and Hall/CRC, ISBN 1-58488-436-3, 2004.
- [73] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. *International World Wide Web Conference archive Proceedings of the 15th international conference on World Wide Web*, pages 1039–1040, 2006.

Appendix A

An Interactive Approach to Route Search

An Interactive Approach to Route Search

Yaron Kanza*
Technion
kanza@cs.technion.ac.il

Roy Levin*
Technion
royl@cs.technion.ac.il

Eliyahu Safra
safraeli@gmail.com

Yehoshua Sagiv†
Hebrew University
sagiv@cs.huji.ac.il

ABSTRACT

In a *probabilistic route search*, there is a start location, a target location, and search queries Q_1, \dots, Q_n . Each Q_i has an answer set A_i consisting of geo-spatial objects and their probabilities. The probability of an object $o \in A_i$ specifies the likelihood that o satisfies Q_i . The goal is to compute a route that is short and yet has a high probability of satisfying all the Q_i . This paper investigates interactive route search. Upon arrival at each object, the user provides feedback specifying whether the object satisfies its corresponding query. The goal is to compute the next object to be visited, based on the feedback. Several heuristic algorithms are given and compared experimentally.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

Geographic information system, route, path, search, queries, probabilistic data, heuristic algorithms

1. INTRODUCTION

The goal of a route search is to find a path that passes through several objects of different types that are relevant to the user. For each type, the user specifies a search query

*The work of these authors was supported by the German-Israeli Foundation for Scientific Research & Development (Grant 2165-1738.9/07) and by a grant from the Goldstein UAV and Satellite Center at the Technion.

†The work of this author was supported by the German-Israeli Foundation for Scientific Research & Development (Grant 973-150.6/2007).

that identifies objects of that type that may satisfy her needs (e.g., a seafood restaurant with moderate prices). In addition, there is a start location s (e.g., the user's office) and a target location t (e.g., her home). The goal is to find the shortest route from s to t that visits one object of each type.

Usually, the objects returned by a search query are not certain answers. That is, there is some probability (of less than 100%) that an object actually satisfies the user's needs. For example, when the user arrives at a gift shop, she may not find a gift that she likes. This uncertainty is modeled by assigning a probability to each object that is returned as an answer to a search query.

Since objects have probabilities, an effective route may have to include more than one object of each type, in order to guarantee a sufficiently high level of success. Thus, the desire to have the shortest possible route conflicts with the necessity to guarantee a high probability of satisfying the user's needs. Semantics and algorithms for route search over probabilistic data were investigated in earlier work [3].

In this paper, we deal with *interactive route search*. That is, upon arriving at each object on her route, the user informs the system whether the object satisfies her needs. Based on that feedback, the system computes the next object to be visited by the user. Interactive route search fits the ubiquitousness of hand-held devices in our technologically driven era. Moreover, computing a route interactively is likely to reduce the distance covered by the user.

In earlier work on route search, they either dealt with non-probabilistic datasets (e.g., [1, 2, 5, 7, 8]) or gave non-interactive algorithms (e.g., [6, 4]). Consequently, the results of earlier work cannot be used to solve the interactive route-search problem that we address in this paper. Our contribution is in giving the first algorithms for this problem and showing their effectiveness.

In some cases, the user may want to impose order constraints on the types of objects to be visited. For example, she wants to draw cash from an ATM before going to a restaurant. We will deal with order constraints in a forthcoming paper.

2. ROUTE-SEARCH QUERIES

A *geo-spatial dataset* is a collection O of objects. Each object represents a real-world geographical entity and its location is the same as that of the entity. An object may have additional spatial and non-spatial attributes. Height and shape are examples of spatial attributes. Address and name are examples of non-spatial attributes. We assume that locations are points and are unique, that is, different

objects have different locations. Distances between objects are not necessarily Euclidean; rather, they can be computed from a given road network. We use $dist(o, o')$ to denote the distance from o to o' .

A *search query* specifies a collection of objects that are of interest to the user. We do not consider a specific syntax or semantics of search queries. For example, a search query Q_i could be stated as a list of keywords (e.g., **Vegetarian Japanese Restaurant**) and some constraints (e.g., **rank** ≥ 3). Our only assumption is that the evaluation of Q_i returns an *answer set* A_i comprising objects and their probabilities. The probability of an object $o \in A_i$ specifies the likelihood that o satisfies Q_i . Discussing how the probabilities are computed is beyond the scope of the paper.

Route-search queries are generated by combining several search queries that specify different types of objects that the user would like to visit. We represent a route-search query as a triplet $R = (s, t, \mathcal{Q})$, where s is a *start location*, t is a *target location*, and $\mathcal{Q} = \{Q_1, \dots, Q_m\}$ is a set of search queries. We assume that the answer set A_1, \dots, A_m of Q_1, \dots, Q_m , respectively, are pairwise disjoint (this entails no loss of generality).

3. INTERACTIVE ALGORITHMS

We now describe interactive algorithms for solving a route-search query $R = (s, t, \mathcal{Q})$, where $\mathcal{Q} = \{Q_1, \dots, Q_m\}$ and A_1, \dots, A_m are the answer sets of Q_1, \dots, Q_m , respectively. All the algorithms operate over the objects of A_1, \dots, A_m , and they compute a route by iteratively increasing a partial sequence σ . Initially, the partial sequence comprises only the start location, namely, $\sigma = s$. On each iteration, the algorithm computes the next object o_k to be visited; thus, o_k is added at the end of σ . When arriving at o_k , the user provides a feedback, denoted by $o\text{-sat}(o_k)$. If o_k actually satisfies the corresponding search query (i.e., the Q_i , such that $o_k \in A_i$), then $o\text{-sat}(o_k)$ is **true**; otherwise $o\text{-sat}(o_k)$ is **false**.

Let $\sigma = s, o_1, \dots, o_k$ be the partial sequence visited thus far. The set of *unsatisfied queries* of R with respect to (abbr. w.r.t.) σ , denoted by $q\text{-unsat}_R(\sigma)$, comprises all the Q_i , such that σ has no object that satisfies Q_i ; that is,

$$q\text{-unsat}_R(\sigma) = \{Q_i \mid Q_i \in \mathcal{Q} \text{ and } \neg \exists o (o \in \sigma \wedge o \in A_i \wedge o\text{-sat}(o))\}.$$

Each of our algorithms chooses the next object to be visited from the set of *candidate objects* w.r.t. σ , denoted by $candidates_R(\sigma)$. This set comprises the objects that neither have been visited nor belong to answer sets of search queries that have already been satisfied; that is,

$$candidates_R(\sigma) = \{o \mid \exists Q_i (Q_i \in q\text{-unsat}_R(\sigma) \wedge o \in A_i \wedge o \notin \sigma)\}.$$

Observe that if $A_j \cap candidates_R(\sigma) = \emptyset$ for some $Q_j \in q\text{-unsat}_R(\sigma)$, then σ cannot be completed to a route that satisfies all the search queries. In addition, if $q\text{-unsat}_R(\sigma) = \emptyset$, then all the queries of \mathcal{Q} have been satisfied, and hence, t is the next destination and the computation ends.

3.1 Naive and Oriented Greedy Heuristics

The *naive greedy heuristic* chooses the candidate object that is closest to the current location l (where l is either s or some o_k). The naive greedy heuristic is simple and efficient.

However, it suffers from the drawback of ignoring the location of the target t . Consequently, it may compute a route that drifts far away from t and is unnecessarily long, due to the distance from the last object to t . A possible solution is to choose the next object o' based on the combined distance of o' from both the current location and t . This approach is likely to compute a route in the general direction toward t . But the route might progress too fast toward t , that is, within a few steps, the route will reach objects near t , even when there are many relevant objects in the vicinity of s and only a few near t .

The *oriented greedy heuristic* assuages the above problems by choosing the next object o' so that it will be near the current location as well as close to the straight line from s to t . In order to do so, this heuristic chooses the candidate object o' that minimizes the sum of distances $dist(l, o') + dist(s, o') + dist(o', t)$, where l is the current location. In the sequel, the oriented greedy heuristic is called the *Greedy* algorithm.

3.2 Optimistic Approach

The greedy approach has a local scope. Namely, it picks the next object o' without taking into account the likelihood that this choice would eventually lead to the shortest route that satisfies all the remaining queries. The *optimistic approach* takes a global view by looking for the shortest route (from the current location to t) that might satisfy all the remaining search queries. This approach is optimistic in the sense that it ignores the probabilities and assumes that objects of the answer sets definitely satisfy their corresponding queries.

The first step is to compute the shortest route from s to t that passes through one object of each answer set. Unfortunately, this is an NP-hard problem [4]. Therefore, we use the infrequent-first heuristic (IFH) of [4] for this task. The user travels along the route computed by IFH until she encounters the first object o_k that does not satisfy its corresponding search query. When that happens, the optimistic algorithm uses IFH to compute a new route (from the current location o_k to t) that satisfies all the remaining search queries. Observe that the new route passes through exactly one object of each set $A_{j_i} \cap candidates_R(\sigma)$, where $1 \leq i \leq d$ and A_{j_1}, \dots, A_{j_d} are the answer sets corresponding to the search queries of $q\text{-unsat}_R(\sigma)$.

3.3 Minimizing the Expected Distance (MED)

The optimistic approach employs a best-case scenario by assuming that objects definitely satisfy their corresponding search queries. A more realistic approach is to use an average-case analysis. The main idea is to choose the next object based on the expected, rather than the shortest, distance that still remains to be traveled. To formalize this notion, let s be the current location and consider an object o . The following is a recursive definition of the expected distance to be covered, given that o is the next object to be visited. We use ℓ_s and ℓ_f to denote the expected distances from o to the target location, depending on whether o succeeds (i.e., satisfies its corresponding query) or fails, respectively. Thus, given that o is the next object, the expected distance from the current to the target location is the following sum.

$$dist(s, o) + prob(o) \cdot \ell_s + (1 - prob(o)) \cdot \ell_f \quad (1)$$

In the MED approach, the next object o to be visited is one that minimizes the above sum.

Computing the expected distance for an object o is not easy. It may involve an exponential number of possible routes, and for each one, we need to keep the history in order to avoid visiting the same object more than once. Hence, we use heuristics that estimate the expected distance, rather than compute it precisely.

For the sake of efficiency, the MED algorithm considers only k candidate objects, namely, the top- k according to the oriented greedy heuristic (w.r.t. the current location). The parameter k is provided by the user and in our experiments was usually equal to four. The MED algorithm estimates the distances ℓ_s and ℓ_f using IFH. Thus, the overall estimation obtained by Equation (1) is crude, because it only takes into account the probability of o , but not those of the objects visited after o . Nonetheless, we do it in this way for efficiency considerations.

3.4 Letting The Probability Affect the Route

When computing a route, most of the above algorithms consider only the distances between objects, but ignore the probabilities. One way to add the effect of the probabilities is by changing the distance function as follows. For every two objects o_1 and o_2 , the distance function $dist_p(o_1, o_2)$ is defined to be $dist(o_1, o_2)/prob(o_2)$. We can now use $dist_p$ instead of $dist$. This increases the distance to objects with a low probability of success in a manner that is inversely proportional to the probability.

4. EXPERIMENTS

In order to examine the effectiveness and efficiency of our methods, we tested them over both syntactic and real-world data in a variety of cases. Due to lack of space, however, we only describe the experiments over the real-world data and discuss the main conclusions.

The real-world data that we used in our experiments is part of a digital map, of the city Tel-Aviv, that has been generated by the Mapa company. In our tests, we used the ‘‘Point Of Interest’’ (POI) layer of the map. The objects in this layer represent many different types of geographical entities. We extracted from the map 628 objects of seven different types (20 cinemas, 29 hotels, 31 pedestrian bridges, 54 post offices, 136 pharmacies, 169 parking lots and 189 synagogues). In the experiments, we tested route-search queries R where the number of search queries in \mathcal{Q} is between three to seven.

In order to simulate interactive scenarios, the satisfaction of each visited object was chosen randomly, when the object was visited, according to the probability of the object. Since we wanted to prevent extreme cases, we ran every query 100 times, where in each run, different random choices were made for the objects, and the results were averaged.

The Greedy algorithm (i.e., oriented greedy) and the Optimistic algorithm have two versions—one that uses the actual distances between objects and another that uses weighted distances (in the weighted version, for each edge of the network, the actual distance is divided by the probability of the target object). We denote the actual and weighted versions of Greedy by $aGre$ and $wGre$, in correspondence, and by $aOpt$ and $wOpt$ for Optimistic. In our experiments, $wOpt$ always provides better results than $aOpt$, and $wGre$ almost always provides better results than $aGre$. The reason for

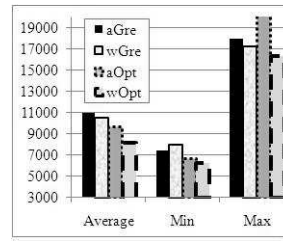


Figure 1: Normally distributed probabilities

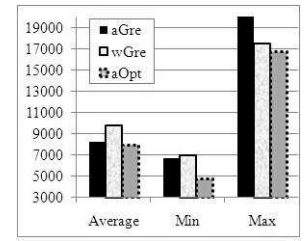


Figure 2: Five answer sets with high probabilities and two with low probabilities

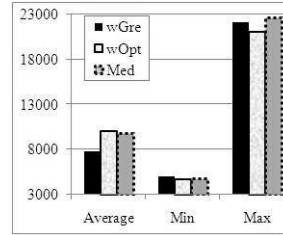


Figure 3: All probabilities are low

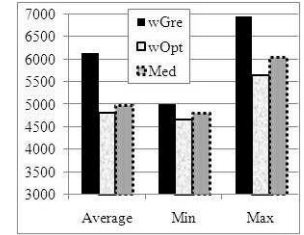


Figure 4: All probabilities are high

this is that the weighted versions prevent the user from going to many objects whose probability is low. Instead, the user goes directly to objects with high probability, i.e., to objects whose chance of satisfying the query is high. The results are presented in Figure 1.

The graph in Figure 1 was produced by combining the results of three different route-search queries, all three comprise seven search queries, but each one with a different pair of start and end points. For each query, the algorithms $aGre$, $wGre$, $aOpt$ and $wOpt$ were run 100 times over the Tel-Aviv dataset. The probabilities of the objects in the dataset were normally distributed with mean of 69.7 percent and a standard deviation of 9.98 percent.

Figure 1 shows the average length of the produced routes, for the 300 runs (100 runs per query). It also shows the length of the shortest route (the columns with the caption Min) and the length of the longest route (the columns with the caption Max) in these 300 runs. The graph shows that the average length of the computed routes is smaller for $wGre$ than for $aGre$, and the same holds for $wOpt$ and $aOpt$. Note that this does not guarantee that for a single run of a single query, the route produced by $wGre$ will be shorter than the route produced by $aGre$, and similarly for $wOpt$ and $aOpt$. The Min and Max columns provide an indication to this.

One case where $aGre$ is better than $wGre$ is when the objects of large answer sets have high probabilities while objects of small answer sets have low probabilities. This is because being near an object of a small answer set happens less frequently than being in proximity to an objects of a large answer set. However, not giving precedence to visiting infrequent objects may lead to the need to visit such objects when none of them is in proximity to the current location. The results of the algorithms in such case are shown by the graph in Figure 2. This graph shows the results of queries

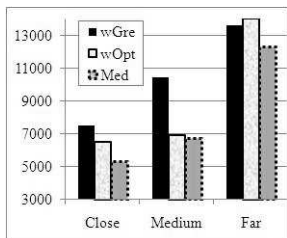


Figure 5: Different start and end locations, four answer sets have high probabilities and three have low probabilities

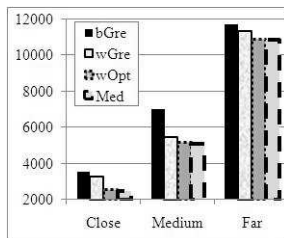


Figure 6: Different start and end locations, normally distributed probabilities

over a dataset where for five large answer sets the probability of objects is high (0.9) and for two small answer sets, the probability of the objects is low (0.3). It can be seen that in such case *aGre* is better than *wGre*.

When all the probabilities are low, the Greedy algorithm is, on the average, better than the other algorithms. The reason for this is that when the probabilities are low, the route should go via many objects since most of the objects fail to satisfy the query. The Optimistic approach and MED try to find a short route to t and only visit several objects on the way. These algorithms reach t and then need to go back to unvisited objects. The Greedy algorithm, in comparison, goes from an object to the next nearest object and, thus, visits many objects on the way to t . This reduces the number of cases where the algorithm needs to go back to unvisited objects. Figure 3 shows the results of an experiment where the probabilities are low. In this experiment, all the objects of all the seven datasets have a probability of 0.3. It can be seen that the average length of a route produced by *wGre* is smaller than the average length of the routes computed by *wOpt* or *MED*. (Note that in this experiment, all the objects have the same probability and, thus, *aGre* is the same as *wGre*, and *aOpt* is identical to *wOpt*.)

When all the probabilities are high, the Optimistic algorithm outperforms the Greedy and MED algorithms. This is because for high probabilities the objects satisfy the query in most of the cases, and thus, most of the time the shortest pre-answer succeeds to satisfy all the queries. The Optimistic algorithm, which follows the shortest pre-answer, computes in all these cases routes that are at least as short as the routes computed by Greedy or MED. An extreme case is when all the objects have a probability that is equal to 1. In this case, the shortest pre-answer, which the Optimistic algorithm computes, is the optimal solution, *i.e.*, a route that is shorter than any other possible route. Figure 4 shows the results of an experiment over a dataset in which all the objects have a high probability of 0.9. Indeed, in this case the average length of the routes computed by *wOpt* is smaller than the average length of the routes computed by *wGre* or *MED*.

In almost all cases, MED computes results that are at least as good as the other algorithms. We have seen that in the results of many experiments and we present one of these experiments in Figure 5. In this experiment, the route-search query was executed over a dataset in which the objects of four answer sets received very high probabilities and the ob-

jects of three answer sets received low probabilities. The test was conducted for queries where s and t are near each other (the columns with the caption Close), far from each other (the columns with caption Far), and neither too close to nor too far away from each other (the columns whose caption is Medium). In all of these cases, *MED* is better than *wOpt* and is much better than *wGre*.

We have also compared our algorithms to Naive Greedy. The results are presented in Figure 6, where Naive Greedy is called *bGre*. In this experiment, the algorithms were executed over a dataset with three answer sets, where the probabilities are normally distributed with mean of 69.7 percent and a standard deviation of 9.98 percent. The experiment tested queries with various distances between the start location s and the target location t . The experiment shows that in all the cases, *wGre*, *wOpt* and *MED* are much better than the Naive Greedy.

5. CONCLUSION

We presented three algorithms for interactive route search. The Greedy algorithm provides the best results for the case where all the probabilities of the objects are low. The Optimistic algorithm provides the best results when all the probabilities are high. In all the other cases, including the case where some objects have high probabilities and some objects have low probabilities, the MED algorithm provides the best results. All the algorithms are practical and efficient; in particular, the time needed to compute the next object is a few milliseconds, several tens of milliseconds and a hundred milliseconds for the Greedy, Optimistic and MED algorithms, respectively.

An important generalization of probabilistic route search is when there are order constraints on the types of objects to be visited (e.g., an ATM should be visited before a restaurant). We have already developed algorithms for this generalization and will introduce them in a forthcoming paper.

For future work, we plan to improve the estimation of the expected distance in the MED algorithm, without sacrificing efficiency. We also intend to investigate the problem of computing route-search queries in the presence of time constraints and traffic conditions.

6. REFERENCES

- [1] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann. The multi-rule partial sequenced route query. In *GIS*, pages 1–10, 2008.
- [2] X. Huang and C. Jensen. In-route skyline querying for location-based services. In *W2GIS*, pages 120–135, 2004.
- [3] Y. Kanza, E. Safra, and Y. Sagiv. Route search over probabilistic geospatial data. In *SSTD*, pages 153–170, 2009.
- [4] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher. Heuristic algorithms for route-search queries over geographical data. In *GIS*, pages 1–10, 2008.
- [5] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.
- [6] E. Safra, Y. Kanza, N. Dolev, Y. Sagiv, and Y. Doytsher. Computing a k -route over uncertain geographical data. In *SSTD*, pages 276–293, 2007.
- [7] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi. Optimal sequenced route query. *VLDBJ*, 17(8):765–787, 2008.
- [8] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis. Constrained shortest path computation. In *SSTD*, pages 181–199, 2005.

Appendix B

Interactive Route Search in the Presence of Order Constraints

Interactive Route Search in the Presence of Order Constraints

Yaron Kanza
Technion
kanza@cs.technion.ac.il

Roy Levin
Technion
royl@cs.technion.ac.il

Eliyahu Safra
ESRI
safraeli@gmail.com

Yehoshua Sagiv
Hebrew University
sagiv@cs.huji.ac.il

ABSTRACT

A *route search* is an enhancement of an ordinary geographic search, where instead of merely returning a set of entities, the result is a route that starts in a given location, ends in a specified location, and goes via entities that are relevant to the search. The input to the problem consists of several search queries, and each query defines a type of geographical entities. When visited, some of the entities succeed in satisfying the user while others fail to do so; however, only the probability of success is known prior to arrival. The main task in a route search is to find a route that visits at least one satisfying entity of each type. In an *interactive search*, the route is computed in steps. In each step, only the next entity of the route is provided to the user, and after each visit of an entity, the user provides a feedback specifying whether the entity is indeed relevant to the search and satisfies her.

This paper investigates interactive route search in the presence of order constraints. These constraints specify that some types of entities should be visited before others. We present several heuristic algorithms for interactive route search for two cases: (1) when the constraints define a complete order, and (2) when the constraints define a partial order. The main challenge in this work is to utilize the feedback in order to compute a route that is shorter and has a higher degree of success, compared to routes that are computed by non-interactive algorithms. We also discuss how to compare the results of the algorithms and introduce suitable measures for doing so. Experiments on real-world data illustrate the efficiency and effectiveness of our algorithms.

Keywords

Geographic information system, route, path, search, probabilistic data, heuristic algorithms, interactive algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Frequently, a user actually wants to visit the entities found in a geographic search that she performs. This requires providing the user not only with entities that satisfy the search conditions, but also with a route that leads to these entities. The need for a route is intensified when several geographical searches are joined to render a combined search task. Forming a route in this case is a rather difficult task due to the need to decide which object should be taken from the result of each search, how to order these objects, and whether to take more than one object from each result. The next example illustrates this.

EXAMPLE 1.1. Suppose that on her way from the office to a business meeting, Alice needs to fill up her gas tank, draw cash from an ATM, buy a new battery for her laptop, and go by a place where there is an Internet connection, in order to check her email. Suppose that Alice can conduct a simple geographic search using her cellular phone or car navigation system. She will be able to locate some nearby ATMs, some close gas stations, coffee shops that provide Internet connection, and electronics stores. However, combining the results of these searches into an efficient route that eventually leads to the location of the meeting can be a hard task.

A route search, as the one in the example above, is a task of computing a route that starts at a given location, which is usually the location of the user, ends at a specified location and goes via geographic entities of certain types. The geographic entities are considered as the *user needs* and are specified by search queries.

One of the difficulties when computing a route is dealing with *uncertainty*, namely, entities that are returned by the search queries, but actually do not satisfy the user needs. In the example above, Alice may find an electronics store that is located near the place of the meeting. Yet, upon arrival at the store, she may discover that the store does not have the specific battery she needs. She may also find out that on her way, she passed close to some other electronics stores;

however, now there is no such store near her and she needs to lengthen her travel or go back to a place she already visited.

For dealing with uncertainty caused by entities that do not satisfy the user needs, we use a probabilistic model. In this type of model, each object has a *probability of success* which is the probability that the entity will satisfy the user needs. The probabilities can be generated from collected statistics. Such statistics, for instance, may show that most of the people who search for an ATM are satisfied with the result of the search. In this case, ATMs will receive high probabilities. The statistics may show that in only 80 percent of the cases, people who search for a restaurant eventually order food in some restaurant that has been discovered in the search. In such a case, a restaurant entity will receive a probability of 0.8. User profiling can be used for adjusting the probabilities to specific users. For instance, the economic status of a user may increase the probability of some restaurants and decrease the probability of others.

When computing a route over probabilistic data, there are two conflicting goals. One goal is that the route will be as short as possible. The other goal is that the route will go via objects that have the highest possible probabilities of satisfying the user needs. Semantics and algorithms for route search over probabilistic data were investigated in a previous work [5]. This paper deals with *interactive route search*.

In an interactive route search, initially the user poses a route-search query; however, instead of providing to the user just one complete and unchanging route, the system creates the route gradually while interacting with the user. In each step, the system provides the next geographical entity on the route. The user goes to the entity and provides to the system feedback on whether the entity has satisfied her. The feedback is used when computing the rest of the route. Note that in this approach, the system can also present to the user a complete planned route, and modify the presented route whenever a feedback that changes the plan is received.

EXAMPLE 1.2. *Consider the search task of Example 1.1. Suppose that the first entity Alice receives is a nearby Internet Cafe. Alice will go to the place and will provide a feedback to the system on whether she has been able to read her email. If the answer is positive, the rest of the route does not need to visit an Internet Cafe. If the answer is negative, the computation of the route continues and is required to satisfy the need for going by a place that provides Internet connection.*

For probabilistic datasets, computing routes iteratively can produce shorter routes than non-interactive evaluation. For instance, if a route goes via an entity of type T and the entity satisfies the user, there is no need to go by other entities of T . In the non-interactive approach, for comparison, it may be required to plan the route to go via several entities of type T so that if one will fail, another one may succeed. Thus, the iterative approach can shorten the length of the produced route.

Computing a route iteratively over a probabilistic dataset so that the route will be as short as possible is a difficult task. In the non-interactive case, the problem is NP-hard [5]. The interactive case is difficult as well for the following reason. For each entity it is required to consider the consequences of both a success in satisfying the user and a failure to do so. Thus, although a single object is chosen in each step, the

choice can be affected by an exponential number of success scenarios.

Order constraints are used for specifying the order by which some types of entities should be visited. For instance, in the scenario of Example 1.1, Alice may need to visit an ATM and an electronics store before going to an Internet Cafe. The order constraints may define a *complete order* that specifies for each pair of types which one should be visited first. It may also define a *partial order* that specifies the visit order for some pairs of types, but does not specify it for the others.

In the presence of order constraints, the route-search algorithms need to guarantee that the objects are visited in an order that satisfies the constraints. Thus, the constraints are an additional factor that makes it harder to devise algorithms for route search. The case of a partial order is more difficult to handle, because one has to consider all the complete orders that are consistent with the given partial order. In the case of a complete order, there is just one order to consider and that makes the problem conceptually and computationally easier.

Earlier works dealt either with non-probabilistic datasets (e.g., [1, 2, 7, 13, 14]), or with the non-interactive version of the problem (e.g., [9, 6]). Those results cannot be used to solve the interactive route-search problem that we address in this paper. Recently, interactive route search over probabilistic data has been introduced and investigated in [4]. However, the work of [4] does not consider order constraints. As mentioned earlier, in the presence of order constraints, the problem requires more intricate algorithms. Our contribution lies in giving the first algorithms for interactive route search in the presence of order constraints, and showing their effectiveness. Additionally, one of these algorithms has proven to be dominant in all of our experiments. In comparison, each of the algorithms of [4] is better than their other algorithms under different circumstances.

The paper is organized as follows. In Section 2, we present our framework and formally define interactive route search with order constraints. In Section 3, we present interactive algorithms for route-search queries with order constraints. Some of the algorithms of this section handle the case of a complete order, while others deal with a partial order. In Section 4, we present the results of experiments with these algorithms. Finally, we conclude in Section 5.

2. PROBABILISTIC ROUTE SEARCH

In this section, we present our framework, we formally define the concept of *interactive route search*, and we explain how order constraints affect route-search queries.

2.1 Geo-spatial Datasets

A *geo-spatial dataset* consists of a collection O of geo-spatial objects and a graph G of a road network that connects the objects. Each object represents a real-world geographical entity and its location is the same as that of the entity. An object may have additional spatial and non-spatial attributes. Height and shape are examples of spatial attributes. Address and name are examples of non-spatial attributes. We assume that locations are points and are unique, that is, different objects have different locations. For objects that are represented by a polygonal shape and do not have a specified point location, an arbitrary point inside them is chosen to be the point location. In the sequel,

“object” and “entity” are synonyms, although technically an object is a representation of a real-world entity.

Each edge in the graph G represents a segment of a real-world road and it has a length. The length of an edge is the length of the road segment it represents. That is, an edge with length ℓ between two objects o_1 and o_2 represents a real-world road with length ℓ connecting o_1 and o_2 . We use $length(o_1, o_2)$ to denote the length of this edge.

A path in G from node o_1 to node o_2 is a sequence of nodes o_1, o_2, \dots, o_m , such that every two adjacent nodes o_i and o_{i+1} are connected by an edge of G . The length of the path is the sum of the lengths of its edges, namely, $\sum_{i=1}^{m-1} length(o_i, o_{i+1})$. The *distance* between two objects o and o' is the length of the shortest path that connects them. We denote this distance by $dist(o, o')$. Efficient methods for computing the distance between objects over a road network were given by Samet *et al.* [11] and by Shahabi *et al.* [12].

2.2 Search Queries

Users employ *search queries* to specify the entities that they would like to visit. A search query consists of a set of keywords and a set of constraints. The keywords and the constraints determine which entities are likely to be relevant to the user.¹

EXAMPLE 2.1. Consider a query Q_{jv} that comprises the set of keywords $\{\text{Restaurant}, \text{Japanese}, \text{Vegetarian}\}$ and the constraint $\text{rank} \geq 3$. This query searches for Japanese restaurants that serve vegetarian food and have a rank that is not below three.

The objects that are relevant to such a query can be determined by taking into account several factors: the number of keywords that appear in the attributes of each object, the “importance” of these keywords and the “importance” of the attributes in which they appear. Ordinary search methods, such as TF-IDF, Okapi BM25 [3, 8] and others [10], can be applied to determine relevancy. The constraints can be used to specify exact conditions on specific attributes.

Relevancy, however, does not mean certainty. That is, a relevant object is likely to satisfy the user’s needs, but there is no guarantee. For example, a search engine may easily locate electronics stores, but it is typically impossible to guarantee the availability of a specific item (e.g., a battery for a laptop). Therefore, we use probabilities to specify the likelihood that relevant objects actually satisfy the user’s needs. Formally, the result of a search is represented as a *probabilistic dataset*, namely, each object is assigned a value $0 \leq p \leq 1$, called *probability of success* (or *probability*, for short). The probability of an object o specifies what is the likelihood that o represents an entity that actually satisfies the user’s needs, rather than just having some relevancy to her query. For example, if the query is Q_{jv} , then an object whose attributes contain the words “Japanese,” “Restaurant” and “Vegetarian” is more likely to satisfy the user’s needs than an object that only contains some of these words. We denote the probability of an object o by $prob(o)$.

There are many different ways to determine the probability of success for objects in the result of a search. Such probabilities can be based on a statistical analysis of a large collection of queries to which users have provided a feedback

¹An exact syntax and semantics of search queries is not needed for this paper.

on how satisfied they have been with the answers. From such statistics, heuristics and rules that provide an estimation of the probabilities can be derived. However, the details of how to determine probabilities are beyond the scope of this paper.

2.3 Route-Search Queries

Route-search queries are generated by combining several search queries that specify different types of entities through which the route should go. We use Q (typically with a subscript) to denote a search query, such as the one given in Example 2.1. We denote by \mathcal{Q} a collection of several search queries that together constitute one component of a route-search query, as we explain later.

An *order constraint* on a route-search query \mathcal{Q} is a pair (Q_1, Q_2) , where Q_1 and Q_2 are distinct search queries of \mathcal{Q} . Intuitively, this pair specifies that the user must visit an entity of the answer to Q_1 that satisfies her needs prior to arriving at an entity of the answer to Q_2 . Users can add order constraints to a route-search query by specifying a set of such pairs.

Let C be a set of order constraints over \mathcal{Q} . The *precedence graph*, denoted by G_C , is a directed graph whose nodes are the search queries of \mathcal{Q} and whose directed edges are the pairs of C . When there is a path in G_C from some query Q_1 to a query Q_2 , we say that Q_1 *precedes* Q_2 and we denote this by $Q_1 \prec Q_2$. We say that C is a *valid* set of constraints if G_C is acyclic. It is easy to see that when there is a cycle in C , it is impossible to satisfy all the order constraints, because there are two queries such that each one should precede the other.

We say that G_C defines a *complete order* over \mathcal{Q} if it contains a Hamiltonian path, that is, a directed path that goes via all the elements of \mathcal{Q} . Otherwise, we say that G_C defines a *partial order*.

In a *route-search query*, the user specifies a *start location* s , a *target location* t , a set \mathcal{Q} of search queries, and a set C of valid order constraints. Hence, we represent a route-search query as a 4-tuple $R = (s, t, \mathcal{Q}, C)$.

EXAMPLE 2.2. Consider again Example 1.1. A suitable route-search query for Alice should include (1) the location s of her office, (2) the location t where the meeting should be held, and (3) the following four search queries: $Q_1 = \{\text{gas station}\}$, $Q_2 = \{\text{ATM}\}$, $Q_3 = \{\text{laptop battery}\}$, and $Q_4 = \{\text{Internet Cafe}\}$. The order constraints (Q_2, Q_4) and (Q_3, Q_4) specify that Alice should visit an ATM and an electronics store before going to the Internet Cafe. Note that there is no order constraint that involves Q_1 which means that a gas station can be located anywhere on the route.

Consider a route-search query $R = (s, t, \mathcal{Q}, C)$, where \mathcal{Q} is the set $\{Q_1, \dots, Q_m\}$ of search queries. The *result* of Q_i , denoted by A_i , comprises the objects of the database that are relevant to Q_i . We assume that the sets A_1, \dots, A_m are pairwise disjoint. In other words, distinct search queries of \mathcal{Q} refer to different types of objects. For example, one search query is about hotels, another is concerning restaurants, etc. A *pre-answer* to R is sequence s, o_1, \dots, o_k, t that starts at s , ends at t and goes via objects of the results A_1, \dots, A_m , such that every A_i has at least one object in the sequence. The objects are visited in an order that conforms to the constraints of C . That is, for all o_{i_1} and o_{i_2} , where $i_1 < i_2$, it holds that if o_{i_1} belongs to A_{j_1} and o_{i_2} belongs to A_{j_2} ,

then Q_{j_2} does not precede Q_{j_1} (i.e., in G_C there is no path from Q_{j_2} to Q_{j_1} , so $Q_{j_2} \not\prec Q_{j_1}$).

The *length* of the route is the sum of the distances between consecutive objects, that is,

$$\text{dist}(s, o_1) + \sum_{i=1}^{k-1} \text{dist}(o_i, o_{i+1}) + \text{dist}(o_k, t).$$

2.4 Interactive Search

Answering route-search queries is traditionally done by computing a complete route from s to t that has a high probability-of-success and a short length [5]. An interactive search is different from the traditional approach in the following aspect. After visiting an entity, the user provides a feedback on whether the entity actually satisfies the corresponding search query, and only then does the system determine the next entity to be visited. In other words, instead of computing a complete route in advance, the route is computed incrementally. At each step, the system provides to the user a single object, which is the next one on the route. After visiting the geographical entity that corresponds to the object, the user sends to the system information on whether the entity satisfies her needs, and based on that feedback, the next object on the route is computed. Alternatively, the system may give to the user a complete route (that visits relevant objects of the search queries that still have to be satisfied). The system may change this route when the feedback warrants doing so.

The computation of the route is influenced by the order constraints. When the user visits an entity that meets her needs, the corresponding search query is deemed satisfied. In each step, the user can visit an entity only if the corresponding object o is an answer to a search query Q_i , such that all the queries that precede Q_i have already been satisfied. When all the queries have been satisfied, the user goes to the target location t and the search ends. Recall that when there are m search queries in \mathcal{Q} , there is a need to visit exactly m entities that satisfy the user.

Note that if all the objects of some answer set A_j have already been visited, and none has satisfied the user, then there is no way to satisfy R . In this case, a failure message should be sent to the user and a new search should be initiated.

When the order defined by C is complete, then in each step the user can visit only objects of one answer set. Hence, answering a route-search query when the constraints define a complete order is simpler than in the case of a partial order.

Our goal is to develop algorithms for interactive route search that compute routes that are as short as possible.

3. ALGORITHMS

In this section, we describe interactive algorithms for route search. Each algorithm has two versions: one is for queries whose constraints define a complete order, and the second version is for queries whose constraints define a partial order. All the algorithms operate over the objects in the answer sets A_1, \dots, A_m of the search queries of \mathcal{Q} , and they compute a route by iteratively increasing a partial sequence σ . Initially, the partial sequence comprises only the start location, namely, $\sigma = s$. On each iteration, the algorithms provide to the user the next object o_k to be visited; thus, o_k is added at the end of σ . When arriving at o_k , the user provides a feedback regarding whether o_k actually satisfies the corresponding search query (i.e., the query Q_i , such that

$o_k \in A_i$). The feedback determines whether the objects of A_i are still relevant to the search and whether Q_i is satisfied.

For each object o in the sequence σ , we denote by $o\text{-sat}(o)$ the feedback received for o . When this feedback is **true**, it means that the object satisfies the corresponding search query. Otherwise (i.e., in the case of a **false** feedback), the object does not satisfy the query.

On each iteration, an object is chosen from the answers to the queries that have not yet been satisfied. Next, we formally define the set from which the object is chosen. Consider a route-search query $R = (s, t, \mathcal{Q}, C)$, where $\mathcal{Q} = Q_1, \dots, Q_m$. Let $\sigma = s, o_1, \dots, o_k$ be the partial sequence computed so far. The *unsatisfied queries* of R are all the queries Q_i , such that σ has no object that satisfies Q_i . In other words, we use $q\text{-unsat}_R(\sigma)$ to denote the set of these queries and define

$$q\text{-unsat}_R(\sigma) = \{Q_i \mid Q_i \in \mathcal{Q} \text{ and } \neg \exists o(o \in \sigma \wedge o \in A_i \wedge o\text{-sat}(o))\},$$

where A_i is the answer set for Q_i .

In each iteration, the sequence σ is extended by providing to the user the next object of the route. The added object is chosen from a set of *candidate objects*, denoted by $\text{candidates}_R(\sigma)$, that consists of all objects o , such that o has not yet been visited and its addition to σ complies with the order constraints. In order to compute the set of candidate objects, consider the precedence graph G_C that is generated from the order constraints C . Let G_{unsat} be the induced subgraph of G_C w.r.t. the unsatisfied queries of $q\text{-unsat}_R(\sigma)$. That is, G_{unsat} is obtained from G_C by removing all the satisfied queries and their incident edges. Let \mathcal{Q}_0 be the set of nodes of G_{unsat} with no incoming edges (i.e., queries that have no preceding query in G_{unsat}). Then, o is a candidate object if o does not appear in σ and is an answer to some query of \mathcal{Q}_0 . When \mathcal{Q}_0 is the empty set, all the queries have been satisfied (i.e., $q\text{-unsat}_R(\sigma) = \emptyset$) and the route must continue to the end location t . If there is no candidate object and \mathcal{Q}_0 is not empty, then the route-search query R cannot be satisfied. Note that when C defines a complete order on \mathcal{Q} , then in each iteration (except for the last one where \mathcal{Q}_0 is empty), \mathcal{Q}_0 contains exactly one query and thus all the candidate objects are answers to the same query.

3.1 Naive Greedy Heuristic

The *naive greedy heuristic* is a simple method that serves as our baseline; namely, more elaborate algorithms will be compared to it. In each iteration, this heuristic chooses the candidate object that is closest to the current location l . Note that l is s in the first iteration, and l is some o_k in subsequent iterations. Formally, when $q\text{-unsat}_R(\sigma) = \emptyset$, all the queries of \mathcal{Q} have been satisfied, and hence, t is the next location and the computation ends. When $q\text{-unsat}_R(\sigma) \neq \emptyset$, the naive greedy heuristic chooses a candidate object o' that is nearest to l , namely, $o' \in \text{candidates}_R(\sigma)$ and

$$\text{dist}(l, o') = \min \{ \text{dist}(l, o) \mid o \in \text{candidates}_R(\sigma) \}.$$

3.2 Oriented Greedy Heuristic

The naive greedy heuristic is simple and efficient. However, it suffers from the drawback of ignoring the location of the target t . Consequently, it may compute a route that drifts far away from t and is unnecessarily long, due to the distance from the last object to t . A possible solution is to

choose the next object o' based on the combined distance of o' from both the current location and t . This approach is likely to compute a route in the general direction toward t . But the route might progress too fast toward t , that is, within a few steps, the route will reach objects near t , even when there are many relevant objects in the vicinity of s and only a few near t .

The *oriented greedy heuristic* is aimed at solving the above problems by choosing the next object o' so that it will be near the current location as well as close to the straight line from s to t . In order to do so, the algorithm computes for each candidate object o' , the sum of distances $dist(l, o') + dist(s, o') + dist(o', t)$, where l is the current location. Then the algorithm chooses a candidate object that minimizes this sum.

3.3 Optimistic Approach

The main weakness of the greedy approach is that it picks the next object o' without taking into account how likely it is to complete the route from o' to t by traveling the shortest possible distance. In other words, to obtain a better algorithm, we should also consider the distance of the route that starts at o' , passes through objects that satisfy the remaining search queries and ends at t . The *optimistic approach* does that by computing at each iteration a complete route with respect to the search queries that still have to be satisfied. We now describe how it works.

The algorithm computes the shortest pre-answer, that is, as short a route as possible from the start location to the end location via one object from each answer set A_1, \dots, A_m . The user follows this route till an object fails to satisfy its corresponding search query. When that happens, the algorithm computes a new route from the current location to t that goes via one object of each A_i , such that Q_i has not yet been satisfied.

This approach is “optimistic” in the sense that at each step, the route is computed under the assumption that all the relevant objects satisfy their corresponding queries. If this assumption holds, the shortest pre-answer is indeed the optimal solution. Next, we explain in more detail the two versions of this approach: for queries with constraints that define a complete order, and for queries where the order is partial.

3.3.1 Optimistic Approach for Complete Order

For route-search queries $R = (s, t, \mathcal{Q}, C)$ whose constraints define a complete order, we can efficiently compute the shortest pre-answer. Without loss of generality, suppose that the constraints define the order Q_1, \dots, Q_m over the queries of \mathcal{Q} (i.e., objects of Q_1 should be visited first, then objects of Q_2 , after that objects of Q_3 , and so on.) Consider the answer sets A_1, \dots, A_m of R . The algorithm *DistanceToTarget* of Figure 1 computes for each $o \in A_i$ ($1 \leq i \leq m$), the minimal distance of a route that starts at o and for $j = i + 1, \dots, m$, passes through one object of each A_j in the order of increasing j , and finally arrives at t . We denote this minimal distance by $dist-t(o)$ and refer to it as the *distance-to-target* of o .

The algorithm *DistanceToTarget* iterates through the answer sets in reverse order, that is, from A_m to A_1 . For all objects o of A_m , the loop of Lines 1–2 computes $dist-t(o)$, which is simply the distance from o to t . Line 3 iterates through the remaining answer sets. The loop of Lines 4–5

DistanceToTarget (A_1, \dots, A_m, t)

Input: Target location t , answer sets A_1, \dots, A_m ordered according to the order defined by C

Computes: For each object $o \in A_i$, the minimal distance of a route when starting at o , continuing to an object of A_{i+1} , then to an object of A_{i+2} and so on until getting to an object of A_m and ending at t .

```

1: for each  $o \in A_m$  do
2:    $dist-t(o) \leftarrow dist(o, t)$ 
3: for  $i = m - 1$  downto 1 do
4:   for each  $o \in A_i$  do
5:      $dist-t(o) \leftarrow \min_{o' \in A_{i+1}} (dist(o, o') + dist-t(o'))$ 

```

Figure 1: Computing the distance-to-target values

computes $dist-t(o)$ for all objects o of A_i using the values computed for A_{i+1} . In particular, $dist-t(o)$ is the minimum of the sum $dist(o, o') + dist-t(o')$ over all $o' \in A_{i+1}$.

Figure 2 gives the optimistic algorithm for route-search queries $R = (s, t, \mathcal{Q}, C)$ where C defines a complete order. The algorithm computes a route that satisfies the search queries Q_i in the order of increasing i . In each iteration, it suffices to compute only the next object to be visited, rather than a whole route. Line 1 sets the current location to s . The loop of Line 2 iterates through the answer sets A_i in the order of increasing i . For each A_i , the loop of Line 4 iterates over objects of A_i until it finds one that satisfies Q_i . In Line 5, the algorithm picks the object o of A_i that appears on the shortest pre-answer (w.r.t. Q_i, \dots, Q_m) from the current location to t . In Line 6, the user is informed to travel to o and provides her feedback. Line 7 sets the current location to that of o . The test of Line 8 checks whether o satisfies Q_i . If the test is positive, then Line 9 sets *found* to **true**, which means that the while loop of Line 4 terminates and the algorithm proceeds to the next iteration of Line 2. Otherwise (i.e., o does not satisfy Q_i), the object o is removed from A_i and another iteration of the loop of Line 4 is done. If A_i becomes empty before finding an object that satisfies Q_i , the algorithm terminates in Line 13 after notifying the user that the route cannot be completed. When the loop of Line 2 terminates (without reaching Line 13), the user is informed to travel to the target location.

3.3.2 Optimistic Approach for Partial Order

In the case of a complete order, computing the distance-to-target values is rather straightforward, because the shortest route from the current location to t is unique. In the case of a partial order, the shortest route may vary depending on the types of objects that have already been visited. As an example, suppose that there is no order constraint that involves Q_i ; that is, an object of A_i may appear anywhere on the route. If the current location is an object $o \in A_j$, where $j \neq i$, then we should consider (at least) two distinct shortest routes from o to t ; one of those routes visits an object of A_i while the other does not. In other words, the distance-to-target value of o depends on whether an object of A_i has already been visited or not. Thus, we should compute the distance-to-target value of o for each possible *history*, namely, each sequence of queries that have already

Ordered Optimistic $((s, t, \mathcal{Q}, C), D)$

Input: Start location s , target location t , search queries Q_1, \dots, Q_m ordered according to C , a dataset D

Output: A route that satisfies the search queries Q_i in the order of increasing i , based on feedback from the user

```
1:  $u\text{-location} \leftarrow s$ 
2: for  $i = 1$  to  $m$  do
3:    $found \leftarrow false$ 
4:   while  $A_i \neq \emptyset$  and not  $found$  do
5:      $o \leftarrow \operatorname{argmin}_{o \in A_i} (dist(u\text{-location}, o) + dist\text{-}t(o))$ 
6:     provide  $o$  to the user and receive a feedback
7:      $u\text{-location} \leftarrow$  the location of  $o$ 
8:     if  $o$  satisfies  $Q_i$  then
9:        $found \leftarrow true$ 
10:    else
11:       $A_i \leftarrow A_i - \{o\}$ 
12:    if not  $found$  then
13:      return "the route cannot be completed"
14: provide the target destination  $t$  to the user
```

Figure 2: Optimistic algorithm when C defines a complete order

been satisfied.

Formally, we first construct the set \mathcal{O}_C of all the complete orders over \mathcal{Q} that conform to the constraints of C . Next, consider an object $o \in A_i$. We have to compute for o a distance-to-target value for each sequence Q_{i_1}, \dots, Q_{i_f} of distinct search queries, such that $Q_{i_1}, \dots, Q_{i_f}, Q_i$ is a prefix of some element of \mathcal{O}_C . We do it by considering every suffix Q_{i_g}, \dots, Q_{i_m} , such that $Q_{i_1}, \dots, Q_{i_f}, Q_i, Q_{i_g}, \dots, Q_{i_m}$ is in \mathcal{O}_C . We compute the distance-to-target value of o w.r.t. the complete order $Q_{i_1}, \dots, Q_{i_f}, Q_i, Q_{i_g}, \dots, Q_{i_m}$ using the algorithm of Figure 1. The actual distance-to-target value of o w.r.t. the sequence Q_{i_1}, \dots, Q_{i_f} is the minimum over all the possible suffixes. This computation is based on the assumption that all the objects that correspond to the queries of a possible suffix Q_{i_g}, \dots, Q_{i_m} are available, namely, none of them has already been visited and failed. However, this is not necessarily true, because the partial order implied by the constraints of C may allow objects corresponding to some Q_j ($j \neq i$) to be visited either before or after o . Therefore, the computed distance-to-target value is only an estimation. In summary, we create for each object o an *estimated-distance table* (EDT) that maps sequences of search queries to distance-to-target values. Finally, observe that if two sequences consist of exactly the same queries, then the same value is computed for both. Hence, the entries of an EDT are subsets of \mathcal{Q} rather than sequences. The following example illustrates what are the entries of an EDT.

EXAMPLE 3.1. Consider a route-search query where $\mathcal{Q} = \{Q_1, Q_2, Q_3, Q_4, Q_5\}$ and $C = \{Q_1 \prec Q_2, Q_2 \prec Q_3, Q_2 \prec Q_4, Q_3 \prec Q_5, Q_4 \prec Q_5\}$. There are two complete orders to consider: Q_1, Q_2, Q_3, Q_4, Q_5 and Q_1, Q_2, Q_4, Q_3, Q_5 . Now, for an object o_2 in the result of Q_2 , the EDT has a single entry, which maps the set $\{Q_1\}$ to the shortest distance among the following two routes: (1) the shortest pre-answer from o_2 to t with respect to the complete order Q_2, Q_3, Q_4, Q_5 , and (2) the shortest pre-answer from o_2 to t with respect to the

complete order Q_2, Q_4, Q_3, Q_5 .

For an object o_3 in the result of Q_3 there are two entries in the EDT, one is for the set $\{Q_1, Q_2\}$ and the other is for the set $\{Q_1, Q_2, Q_4\}$.

The optimistic approach starts the processing of a route-search query by constructing an EDT for every object. The route is computed in stages as follows. Let $\sigma = s, o_1, \dots, o_k$ be the sequence of objects visited thus far (note that initially $\sigma = s$). We use $q\text{-sat}_R(\sigma)$ to denote the set of queries that have been satisfied by σ (i.e., $q\text{-sat}_R(\sigma) = \mathcal{Q} - q\text{-unsat}_R(\sigma)$). For an object o that has an entry for $q\text{-sat}_R(\sigma)$ in its EDT, let $d_\sigma(o)$ be the value of that entry. The next object to be visited is the one that minimizes the sum $dist(o_k, o) + d_\sigma(o)$, among all objects o that have an entry for $q\text{-sat}_R(\sigma)$ in their EDT.

3.4 Letting The Probability Affect the Route

When computing a route, the greedy algorithms and the optimistic algorithms consider only the distances between objects, but ignore the probabilities. One way to add to these algorithms the effect of the probabilities is by changing the distance function as follows. For every two objects o_1 and o_2 , the distance function $dist_p(o_1, o_2)$ is defined to be $dist(o_1, o_2)/prob(o_2)$. We can now use $dist_p$ instead of $dist$. This increases the distance to objects with a low probability of success in a manner that is inversely proportional to the probability.

3.5 Minimizing the Expected Distance (MED)

The optimistic approach employs a best-case scenario. That is, the next object to be visited is the first one on the shortest route that passes through one object of each answer set A_i , such that Q_i has not yet been satisfied. A more realistic approach is to use an average-case analysis. The main idea is to choose the next object based on the expected, rather than the shortest, distance that still remains to be traveled. To formalize this notion, let s be the current location and consider an object o . The following is a recursive definition of the expected distance to be covered, given that o is the next object to be visited. There are some expected distances ℓ_s and ℓ_f from o to the target location,² depending on whether o succeeds (i.e., satisfies its corresponding query) or fails, respectively. Thus, given that o is the next object, the expected distance from the current to the target location is the following sum.

$$dist(s, o) + prob(o) \cdot \ell_s + (1 - prob(o)) \cdot \ell_f \quad (1)$$

In the MED approach, the next object o to be visited is one that minimizes the above sum.

Computing the expected distance for an object o is not easy. First, there could be an exponential number of pre-answers that need to be considered. Second, we should avoid pre-answers that visit the same object more than once, which means that when constructing the pre-answers, we should keep the entire history (i.e., the visited objects) of each one—doing so for an exponential number of pre-answers is impractical. Hence, we use heuristics that estimate the expected distance, rather than compute it precisely.

²More precisely, the user travels until she either arrives at t or discovers that one of her search queries cannot be satisfied. The expected distance is computed by considering all the routes that the user may travel and the probability of each one.

<p>$MED((s, t, \mathcal{Q}, C), D, \prec)$</p> <p>Input: Start location s, target location t, search queries Q_1, \dots, Q_m ordered according to C, a dataset D, an order \prec over D</p> <p>Output: The next object to be visited</p> <pre> 1: if \mathcal{Q} is empty then 2: return t 3: call $ComputeExpLen(o, E, (s, t, \mathcal{Q}, C), D, \prec)$ 4: $curr \leftarrow s$ 5: for $i = 1$ to m do 6: $found \leftarrow false$ 7: while not $found$ do 8: if $A_i = \emptyset$ then 9: return "the route cannot be completed" 10: $o \leftarrow \underset{o \in A_i}{\operatorname{argmin}}(dist(curr, o) + E[o])$ 11: provide o to the user and get a feedback 12: $curr \leftarrow o$ 13: if o does not satisfy Q_i then 14: remove o from A_i 15: else 16: $found \leftarrow true$ </pre>
--

Figure 3: MED for route-search queries in which C defines a complete order

3.5.1 MED for Complete Order

In this section, we describe the version of MED for route-search queries with a complete order. The algorithm estimates the expected distance given that objects must be visited in the order dictated by the constraints. It employs a heuristic that enforces a total order on the objects of the dataset D , thereby limiting the number of examined routes.

The algorithm is presented in Figure 3. Line 3 uses a subroutine that returns an array E , such that for all objects o , the entry $E[o]$ is an estimation of the expected distance covered by the shortest route that satisfies all the remaining search queries, starting with the one that corresponds to o . The computation of E is described later on. The loop of Line 5 iterates over the answer sets A_i in the order of increasing i . The loop of Line 7 iterates until it finds an object of A_i that satisfies Q_i ; if eventually none is found, then the algorithm terminates in Line 9. Line 10 chooses the next object o to be the one that minimizes the sum of the distance from the current location to o plus the expected length of a route from o to t . If o satisfies its corresponding query, then the algorithm proceeds to the next iteration of Line 5; otherwise, o is deleted from A_i and another iteration of Line 7 is done.

Next, we describe how to compute the estimation $E[o]$ for all objects $o \in D$. First, we define the order \prec over the objects of D as follows. If $o_1 \in A_i$, $o_2 \in A_j$ and $i < j$, then naturally $o_1 \prec o_2$, because objects of A_i must be visited before objects of A_j . In order to define \prec for objects of the same answer set A_i , we partition the straight line from s to t into $m + 1$ equal intervals. Let the sequence of points $p_0, p_1, \dots, p_m, p_{m+1}$ describe this partition, where $p_0 = s$ and $p_{m+1} = t$. In other words, the intervals $[p_i, p_{i+1}]$ ($0 \leq i \leq m$) cover the straight line from s to t , they are disjoint and have the same length. Objects of the same answer set

A_i are ordered according to their distance from p_i . That is, $o_1 \prec o_2$ if o_1 and o_2 are both in A_i and $dist(o_1, p_i) < dist(o_2, p_i)$. In case of a tie (i.e., $dist(o_1, p_i) = dist(o_2, p_i)$), the order between o_1 and o_2 is defined arbitrarily.

The rationale for the above definition is to prefer objects that are closer to the line from s to t and, in particular, objects whose distance from s is linearly proportional to their position on a possible route. In other words, the goal is to choose the next object so that it will be in the direction toward to t , but not too close to t , in order to avoid routes that unnecessarily go back and forth.

When estimating the expected length of a route, we should take into account the possibility that some search queries are not satisfied by any object. To do it properly, we define for each answer set A_i a *penalty* that amounts to the length of a route that goes through all the objects of A_i (which must be done when Q_i cannot be satisfied). That is, $penalty(A_i) = \sum_{j=1}^{k-1} dist(o_j^i, o_{j+1}^i)$, where the route o_1^i, \dots, o_k^i passes exactly through all the objects of A_i from the smallest to the largest, according to the order \prec .

Recall that $E[o]$ denotes our estimation of the expected length of the shortest route from an object o of A_i to t , such that the search queries Q_1, \dots, Q_m are satisfied. $E[o]$ is given by

$$E[o] = prob(o) \cdot (dist(o, o_s) + E[o_s]) + (1 - prob(o)) \cdot (dist(o, o_f) + E[o_f]) \quad (2)$$

where o_s and o_f are defined as follows. If o succeeds, then an object of A_{i+1} should be visited next. Therefore, we choose o_s from the objects of A_{i+1} so that the sum $dist(o, o_s) + E[o_s]$ is minimized, except that o_s is t if $i = m$ (note that by definition, $E[t] = 0$). If o fails, then another object of A_i should be visited next. To avoid an exponential computation, we choose o_f from the objects of A_i that are larger than o according to \prec . In particular, o_f is picked out so that the sum $dist(o, o_f) + E[o_f]$ is minimized; however, if o is the last object of A_i (according to \prec), then we replace the sum $dist(o, o_f) + E[o_f]$ with $penalty(A_i)$, because none of the objects of A_i satisfies Q_i .

The algorithm $ComputeExpLen$ computes all the entries of E by traversing the objects of D from the largest to the smallest, according to \prec , and using Equation (2). The pseudo-code is presented in Figure 4.

3.5.2 MED for Partial Order

The adaptation of MED to partial orders is obtained in a way similar to how it was done in the case of the optimistic approach. An estimation of the expected distance has to be computed for every pair (o, S) , such that o is an object in the answer set of some query and S is a subset of \mathcal{Q} that represents a possible history, namely, the search queries of S have already been satisfied before arriving at o .

Formally, let S be a subset of \mathcal{Q} and Σ be a sequence Q_{i_1}, \dots, Q_{i_m} of distinct search queries. Recall that \mathcal{O}_C is the set of all the complete orders implied by the constraints of C . We say that Σ is *consistent* with S if the following holds.

1. Σ is a suffix of some element of \mathcal{O}_C ;
2. No search query appears in both S and Σ ; and
3. Every search query of \mathcal{Q} appears in either S or Σ .

ComputeExpLen ($E, (s, t, \mathcal{Q}, C), D, \prec$)

Input: Route-search query (s, t, \mathcal{Q}, C) , a dataset D , an order \prec over the objects of D

Output: Array E such that for all $o \in D$, the entry $E[o]$ is an estimation of the expected distance from o to t

```

1: let  $o_{k_m}^m \succ \dots \succ o_1^m$  be the objects of  $A_m$ 
2:  $E[o_{k_m}^m] \leftarrow \text{prob}(o_{k_m}^m) \cdot \text{dist}(o_{k_m}^m, t) +$ 
    $(1 - \text{prob}(o_{k_m}^m)) \cdot \text{penalty}(A_m)$ 
3: for  $j = k_m - 1$  downto 1 do
4:    $o \leftarrow \underset{\{o \in A_m \wedge o_j^m \prec o\}}{\text{argmin}} (\text{dist}(o_j^m, o) + E[o])$ 
5:    $E[o_j^m] \leftarrow \text{prob}(o_j^m) \cdot (\text{dist}(o_j^m, t) +$ 
    $(1 - \text{prob}(o_j^m)) \cdot (\text{dist}(o_j^m, o) + E[o])$ 
6: for  $i = m - 1$  downto 1 do
7:   let  $o_{k_i}^i \succ \dots \succ o_1^i$  be the objects of  $A_i$ 
8:    $o \leftarrow \underset{o \in A_{i+1}}{\text{argmin}} (\text{dist}(o_{k_i}^i, o) + E[o])$ 
9:    $E[o_{k_i}^i] \leftarrow \text{prob}(o_{k_i}^i) \cdot (\text{dist}(o_{k_i}^i, o) + E[o]) +$ 
    $(1 - \text{prob}(o_{k_i}^i)) \cdot \text{penalty}(A_i)$ 
10:  for  $j = k_i - 1$  downto 1 do
11:     $o_{A_i} \leftarrow \underset{\{o_{A_i} \mid o_{A_i} \in A_i \wedge o_j^i \prec o_{A_i}\}}{\text{argmin}} (\text{dist}(o_j^i, o_{A_i}) +$ 
    $E[o_{A_i}])$ 
12:     $o_{A_{i+1}} \leftarrow \underset{o_{A_{i+1}} \in A_{i+1}}{\text{argmin}} (\text{dist}(o_j^i, o_{A_{i+1}}) + E[o_{A_{i+1}}])$ 
13:     $E[o_j^i] \leftarrow \text{prob}(o_j^i) \cdot (\text{dist}(o_j^i, o_{A_{i+1}}) + E[o_{A_{i+1}}]) +$ 
    $(1 - \text{prob}(o_j^i)) \cdot (\text{dist}(o_j^i, o_{A_i}) + E[o_{A_i}])$ 

```

Figure 4: Computing the expected distances

We say that Σ is an i -sequence if Q_i is the first search query in Σ .

Consider an object $o \in A_i$. The array E (of estimations) of expected distances has an entry for every pair (o, S) , such that $S \subseteq \mathcal{Q}$ and there is some i -sequence Σ that is consistent with S . The value $E(o, S)$ is computed as follows.

Let Σ be an i -sequence that is consistent with S . We apply the algorithm *ComputeExpLen* of Figure 4 w.r.t. the complete order Σ (while ignoring objects corresponding to search queries that do not appear in Σ). We do it for every i -sequence that is consistent with S . The minimum value computed for o , over all the i -sequences that are consistent with S , is assigned to $E(o, S)$.

The above description of how to compute $E(o, S)$ is not the most efficient way of doing it. In fact, it suffices to apply the algorithm *ComputeExpLen* of Figure 4 once for each complete order of \mathcal{O}_C . (Recall that \mathcal{O}_C is the set of complete orders implied by the constraints of C .) If we do it in this way, then we actually compute values of the form $E[o, \Gamma]$, where $\Gamma \in \mathcal{O}_C$. Let $\Gamma \langle o \rangle$ denote the suffix of Γ that starts at the Q_i corresponding to o . $E(o, S)$ is the minimum over all $E[o, \Gamma]$, such that $\Gamma \langle o \rangle$ is consistent with S .

More specifically, for each object o , we need to divide all the $E[o, \Gamma]$ into subsets, such that in each one all the Γ have the same search queries appearing before Q_i . Thus, each subset corresponds to one $E(o, S)$, where S is the set of search queries that appear before Q_i in all the Γ of the subset. $E(o, S)$ is assigned the minimum value in its corresponding subset.

As earlier, σ denotes the route traveled thus far, and

$q\text{-sat}_R(\sigma)$ is the set of queries that have already been satisfied. The algorithm MED for partial orders is similar to that of Figure 3. The main difference is that the next object to be visited is the one that minimizes the sum $\text{dist}(\text{curr}, o) + E[o, q\text{-sat}_R(\sigma)]$ over all objects o , such that $E[o, q\text{-sat}_R(\sigma)]$ is defined, o has not yet been visited and its corresponding search query still has to be satisfied. If there is no such o , then the algorithm has failed to find a route. As usual, if the route σ has satisfied all the search queries, then the user should travel to the target location t .

3.6 Phantom Objects

The optimistic approach computes the exact minimal distance (using the algorithm *DistanceToTarget* of Figure 1) in the case of a complete order. As noted earlier, in the case of partial orders, the optimistic approach computes only an estimation of the minimal distance. The reason for that is that it takes into account the search queries that have already been satisfied, but not the possibility that some of the visited objects have failed. The values of the minimal distances are computed in a preprocessing step. So, when they are actually used during the construction of a path, it could be that a specific value is based on using an object that has already been visited and failed; hence, this value is only an estimation. We say that a *phantom object* is used if the choice of the next object is based on a value of the minimal distance that incorporates an object that has already been visited. The phenomenon of phantom objects can also occur in the MED algorithm for partial orders.

A simple solution to the effect of phantom objects is to do the following in each step of computing the next object to be visited. If the most-recently visited object has failed, then discard it and recalculate the estimations before determining the next object. We refer to the versions of Optimistic and MED that perform recalculation of the estimations as *Recalculating Optimistic* and *Recalculating MED*, respectively.

This solution is detrimental to the efficiency of these algorithms. Fortunately, our experiments show that phantom objects are rare and recalculating the estimations decreases the length of the produced route only by a very small amount.

3.7 The Complexity of Computing a Step

We now analyze the complexity of the different algorithms. For interactive algorithms, the time complexity of computing an entire route is unuseful because the algorithms are delayed by the need to wait for feedbacks from the user. So instead, we use the following two complexity measures. The *preprocessing complexity* is the time complexity of the computation that is required for providing the first object of the route. The *step complexity* is the time complexity of computing the next object on the route after at least one object has been computed. We analyze our algorithms according to these two measures. In our analysis, we assume that there are n objects in D and these objects are partitioned into m answer sets.

The Naive Greedy and the Oriented Greedy algorithms require no preprocessing. The computation of the first object on the route has the same time complexity as the computation of any other object on the route. In each step, all the objects of the dataset D are examined. Thus, these algorithms can be easily implemented to have $O(n)$ preprocessing complexity and $O(n)$ step complexity.

The Optimistic algorithm for the case of a complete order has a preprocessing step of computing the distance-to-target values. For each object of D , a value is computed by examining the distance from it to all the objects of the next set. Thus, the preprocessing has $O(\frac{n^2}{m})$ time complexity. The computation of an object is done by choosing an object from a set of at most n objects. Hence, the step complexity is $O(n)$.

In the case of a partial order, there can be $m!$ possible orders, and hence the preprocessing has time complexity $O(\frac{n^2}{m}m!)$. The step complexity requires checking n objects and considering at most 2^m entries in the EDT of each object. Thus, the step complexity is $O(n2^m)$.

The algorithm MED for the case of a complete order has a preprocessing step of computing the expected-distance values for the objects. First, the objects of D are sorted. The sort has $O(n \log n)$ time complexity. An expected distance is computed for each object and this is done by considering about $\frac{n}{m}$ objects of some answer set. Thus, the preprocessing complexity is $O(n \log n + \frac{n^2}{m})$. The step complexity requires choosing an object from an answer set, therefore, the step complexity is $O(n)$.

For constraints that define a partial order, MED needs to create EDTs and use them in each step. The preprocessing complexity requires considering $m!$ orders, and hence, it is $O(m!(n \log n + \frac{n^2}{m}))$. The step complexity is $O(n2^m)$, as for Optimistic with a partial order.

Note that in practical scenarios, the number of queries, m , is relatively small. It is reasonable to assume that in most practical cases, users will pose route-search queries of no more than ten search queries. Thus, even though the preprocessing complexity and the step complexity are exponential in m , in the case of partial orders, in practice our algorithms provide answers in an acceptable time. The experiments in the next section confirm this.

4. EXPERIMENTS

In order to examine the effectiveness and efficiency of our methods, we tested them over real-world data in a variety of cases. We conducted many experiments and we present here only the results of typical cases.

4.1 Setting

The real-world data that we used in our experiments is part of a digital map, of the city Tel-Aviv, that has been generated by the Mapa company. A fragment of that map is presented in Figure 5. In our tests, we used the ‘‘Point Of Interest’’ (POI) layer of the map. The objects in this layer represent many different types of geographical entities. We extracted from the map 628 objects of seven different types (20 cinemas, 29 hotels, 31 pedestrian bridges, 54 post offices, 136 pharmacies, 169 parking lots and 189 synagogues). In the experiments, we tested route-search queries R where the number of search queries in \mathcal{Q} is between three to seven.

In order to simulate interactive scenarios, the satisfaction of each visited object was chosen randomly, when the object was visited, according to the probability of the object. Since we wanted to prevent extreme cases, we ran every query 100 times, where in each run, different random choices were made for the objects, and the results were averaged.

4.2 Examples of Specific Routes

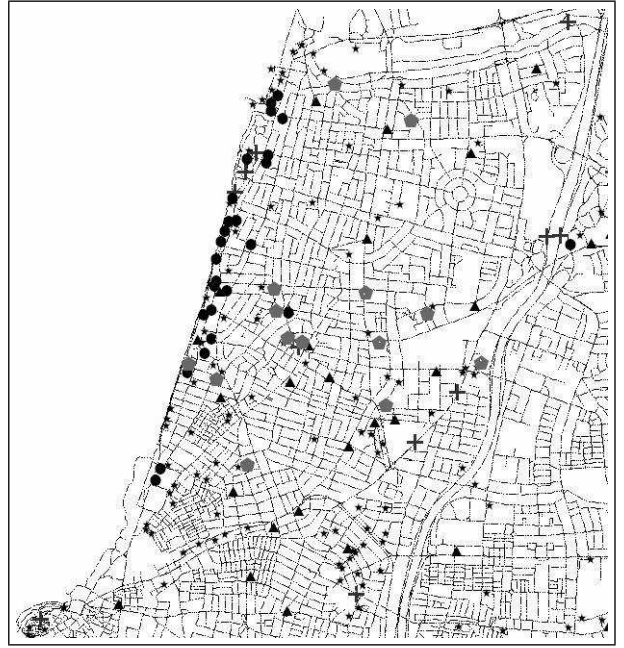


Figure 5: Map of Tel-Aviv (fragment)

We present two cases that illustrate some of the differences between our algorithms. In these two cases, we used real-world datasets, and we run our algorithms so that the results will reflect the actual behavior of the algorithms. For simplicity of presentation, there are no order constraints in the two examples of this section.

The first case compares the greedy algorithm to MED, and it shows why in many cases MED outperforms greedy. It is presented in Figure 6 .

EXAMPLE 4.1. *In this example, a route search with five queries is considered. The objects to be visited are depicted by plus, star, triangle, circle and square icons. The route provided by the greedy algorithm is depicted with a solid line and some locations where the user provided a feedback are depicted as a number inside a circle. The route that MED computed is depicted with a dashed line, and the locations where the user provided a feedback are shown as a number inside a square.*

The result of one of the search queries consists of a single object, and it is depicted using a black square at the bottom left corner of the figure. Since there is only one such square, the route must go via this location.

MED ‘‘plans’’ the entire travel, and thus, it goes from the start location to the location of the black square. (This is also marked by the number 2 inside a square). Then, MED continues directly to the target location going via the other objects it needs to visit.

The greedy algorithm goes to objects that are near the line that connects s and t . It leads the user to the locations depicted by 1, 2 and 3 in a circle. The greedy approach leads toward t till there is only one query left to satisfy—the query whose answer is the black square. This forces the route to lead back in a direction opposite to t , visit the black square and continue to t . Going back and forth due to lack of planning causes the greedy to be inefficient in such case.

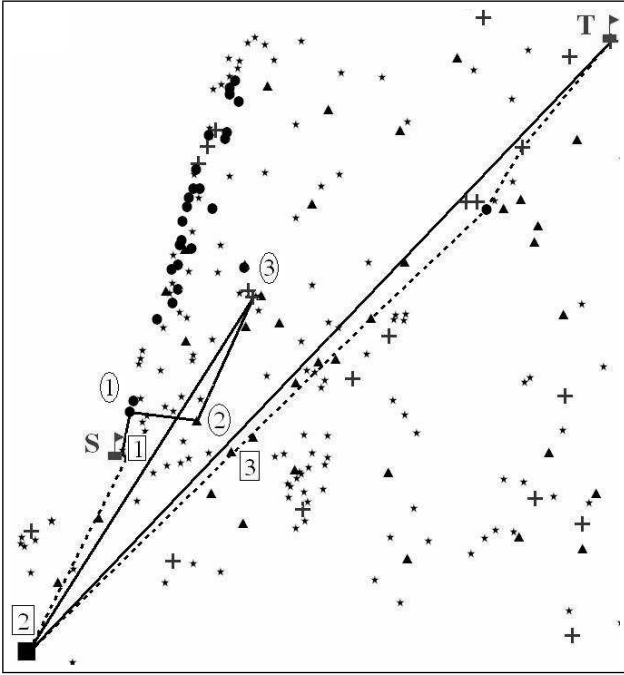


Figure 6: A scenario where the route computed by Greedy (solid line) is significantly longer than the route computed by MED (dashed line).

The second example compares Optimistic to MED.

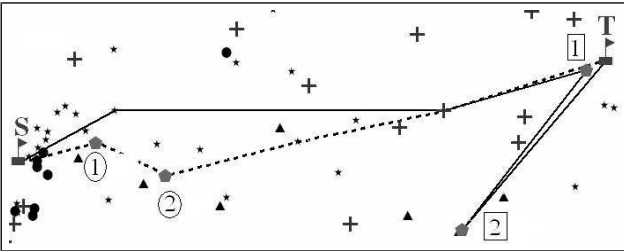


Figure 7: A scenario where the route computed by Optimistic (solid line) is significantly longer than the route computed by MED (dashed line).

EXAMPLE 4.2. The scenario depicted in Figure 7 illustrates the superiority of MED over Optimistic. In this scenario, the route-search query consists of three search queries whose results are depicted by plus, star and pentagon icons, respectively.

The pentagons represent cinemas. In this scenario, cinemas have a probability of 0.7. There is a cinema near t . Optimistic computes the shortest route and reaches that cinema (see the number 1 in square near that cinema). However, in many cases this cinema fails to satisfy the user. In these cases, the route continues to a cinema that is far from t (there is an icon of the number 2 inside a square near that cinema). So, in this scenario, Optimistic generates a route that frequently goes back and forth.

MED, on the other hand, consider the case that the cinema near t will fail and hence, it visits cinemas on the way

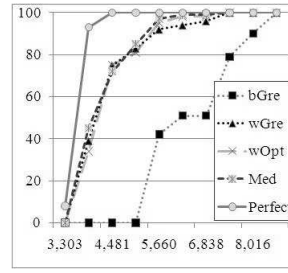


Figure 8: No order.

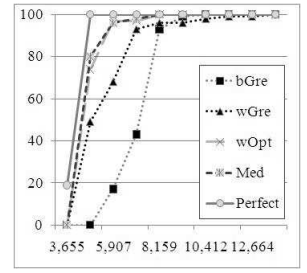


Figure 9: Partial order.

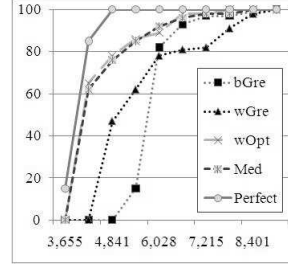


Figure 10: Complete order.

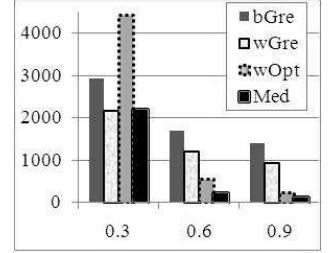


Figure 11: Comparison to the perfect result.

from s to t (these cinemas are marked by 1 and 2 in a circle). If cinema 1 fails, the route continues to 2 with only a small increase in the total length, whereas for the route of Optimistic, when the first cinema fails the increase in the length of the route is large.

4.3 Effectiveness

We conducted a series of experiments to examine the effectiveness of our algorithms. We tested the effect of different parameters on each algorithm. In the experiments we compared MED, Optimistic, Oriented Greedy and Naive Greedy. For Optimistic and Oriented Greedy we experimented with the version that is affected by the probabilities, i.e., the version that uses $dist_p$ instead of $dist$. In this section, we denote the Optimistic by wOpt, the weighted oriented Greedy by wGre, and the Naive Greedy by bGre.

Comparing the algorithms one to the other only provides a relative indication of their effectiveness. For a non-relative comparison, we included in our experiments an algorithm we call Perfect. Perfect computes the shortest route while having the satisfaction conditions of all the objects before the first step. Since Perfect has information that no interactive algorithm has, the route computed by Perfect is the best any interactive algorithm could hopefully compute. Obviously, in actual scenarios such an algorithm does not exist; however, in our experiments we had all the information on the objects, and hence, we were able to use it. We compare the results of our algorithms to the results of Perfect to show that our algorithms are effective in general and not just relatively.

The experiments whose results are presented in Figure 8, Figure 9 and Figure 10 examine the effect of order constraints on the effectiveness of the algorithms. In each of these graphs, the x-axis shows lengths. For each length ℓ , the y-axis presents the percentage of routes that were created interactively and had a length of at most ℓ . The percentage

was achieved by running each route-search query 100 times, while simulating interaction with the user. When comparing two interactive algorithms on such a graph, the better algorithm is the one whose curve is higher because the routes it produces are expected to be shorter.

In the experiments whose results are presented in Figure 8, Figure 9 and Figure 10, probabilities were normally distributed³ with mean 0.7 and standard deviation 0.1. The route-search queries in this experiment comprise five search queries, i.e., need to go via objects of five types.

Figure 8 shows the results of the algorithms for the case where there are no order constraints. There are 120 complete orders in this case. Figure 9 shows the results for the case where there is a partial order. There are 20 complete orders in this case. The case of a complete order is presented in Figure 10.

The results show that MED outperforms the other algorithms in almost all of the cases. Optimistic (wOpt) is almost as good as MED, and both of them are almost as good as Perfect, which shows that they are indeed effective. The Greedy algorithms bGre and wGre are less effective than MED and wOpt.

Figure 11 presents the results of comparing the algorithms to Perfect. For each algorithm, it shows the average difference between the length of the route computed by the algorithm and the length of the route computed by Perfect. The results are shown for the cases where the means of the probabilities of the objects are 0.3, 0.6 and 0.9, respectively. Not surprisingly, for high probabilities the algorithms provide closer results to Perfect than for low probabilities. This experiment also shows that MED is the most effective in all cases. Optimistic is effective when the probabilities are high, but it is not effective when the probabilities are low. This is because it applies an “optimistic” assumption and when the probabilities are low, this assumption is incorrect. The greedy approach wGre is relatively good when the probabilities are low, because in this case, most of the visited objects fail to satisfy the user, so not planning and going to the nearest object is a good strategy for this case.

Figure 12, Figure 13 and Figure 14 show the results of the different algorithms for a search over three datasets, where the probabilities are normally distributed with mean 0.3, 0.6 and 0.9, respectively. In this experiment, the route-search query comprised three search queries, thus, the objects are partitioned into three categories. This experiment provides an additional affirmation to the effectiveness of MED.

In Section 3.6, we presented the problem of phantom objects and claimed that a possible solution is to recalculate the estimation of the minimal distance after every negative feedback. We denote by rMED the algorithm Recalculate MED and by rwOpt the algorithm Recalculate Optimistic. We claimed that recalculation has almost no effect on the effectiveness of the algorithms. The results of an experiment that supports this claim are presented in Figure 15. The test shows this by comparing the results of MED, rMED, wOpt and rwOpt to the results of Perfect. It is done over datasets in which the probability is normally distributed with means 0.3, 0.6 and 0.9, respectively. Each column is the average over three different start and target locations, and for 100 different interactive runs. It can be seen that there is almost

³The actual distribution is close to normal since we do not allow objects to receive a probability lower than zero or greater than one.

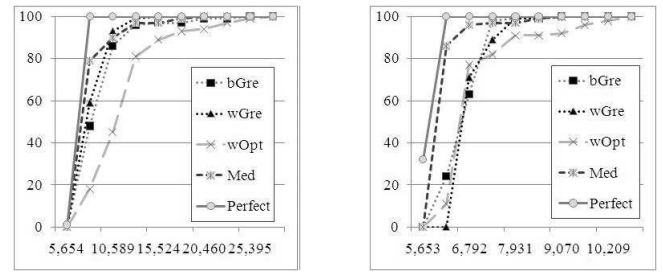


Figure 12: Mean 0.3.

Figure 13: Mean 0.6.

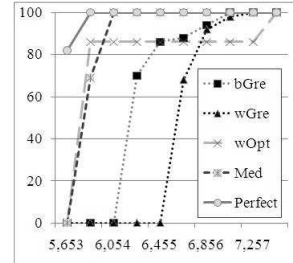


Figure 14: Mean 0.9.

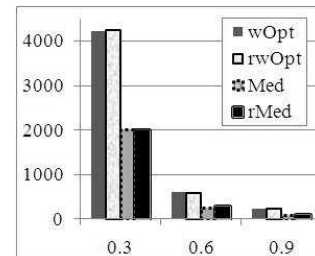


Figure 15: Comparison of the results with and without recalculating prior to every step.

no difference between MED and rMED. Similarly, there is almost no difference between wOpt and rwOpt.

4.4 Efficiency

All the algorithms compute the next object on the route within less than a milli-second. (Except for the Recalculating versions of MED and Optimistic.) The difference in the efficiency of the algorithms is in the preprocessing time they require. When users initiate a route search, they may want the first object to be provided instantly, and thus, the efficiency of the preprocessing is important in many cases.

Table 1 presents the pre-processing times of the different algorithms. It shows that the greedy algorithms are the most efficient. MED is the least efficient because it requires a relatively long preprocessing step. It can be seen that the preprocessing requires significantly less time for a complete order than for no order. In general, the efficiency of the preprocessing is inversely proportional to the number of possible orders that comply with the order constraints.

5. CONCLUSION

We investigated the problem of interactive route search in the presence of order constraints. We examined two cases.

Table 1: Pre-processing times, in milliseconds, for 5 search queries, over a dataset of 419 objects.

Algorithm	Full Order	Partial Order	No Order
bGre	0.6	22	115
wGre	1.6	34	167
wOpt	145	3015	16,217
Med	244	5146	26,207

In one case, the constraints define a complete order over the types of entities that should be visited, and in the other they only define a partial order. For each case, we presented three algorithms, having in mind two goals: computing an effective route (i.e., a route that is as short as possible) and doing it efficiently (i.e., finding the next object on the route as quickly as possible). The Greedy algorithm is the most efficient, yet the route it computes is the least effective. The MED algorithm, in contrast, provides the most effective route; however, its efficiency is the lowest. The Optimistic algorithm is a compromise that provides a route with effectiveness and efficiency that are between those of MED and Greedy. The differences between the running times of the three algorithms are just in the preprocessing phase. The time needed to find the next object is about the same in all of them (less than 1 millisecond). If efficiency is important, then the best may be a hybrid approach that determines the first object using the Greedy algorithm, and then switches to the MED (or Optimistic) algorithm in order to find subsequent objects. The time it takes the user to get to the first object is more than enough for completing the preprocessing. Thus, the hybrid approach is both efficient and effective.

For future work, we plan to consider dynamic route-search queries in which routes can be affected by feedbacks from other users. For example, if Alice provides a feedback that some ATM does not work, there is no reason to send Bob there, even if he is already on the way. Another challenge is to answer queries that consider the availability of transportation. To that end, we intend to investigate the problem of computing route-search queries in the presence of time constraints and traffic conditions.

6. REFERENCES

- [1] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann, *The multi-rule partial sequenced route query*, GIS, 2008, pp. 1–10.
- [2] X. Huang and C.S. Jensen, *In-route skyline querying for location-based services*, W2GIS, 2004, pp. 120–135.
- [3] S. Jones, S. Walker, and S.E. Robertson, *A probabilistic model of information retrieval: Development and comparative experiments (parts 1 and 2)*, Information Processing and Management **36** (2000), no. 6, 779–840.
- [4] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv, *An interactive approach to route search*, GIS, 2009.
- [5] Y. Kanza, E. Safra, and Y. Sagiv, *Route search over probabilistic geospatial data*, SSTD, 2009, pp. 153–170.
- [6] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher, *Heuristic algorithms for route-search queries over geographical data*, GIS, 2008, pp. 1–10.
- [7] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.H. Teng, *On trip planning queries in spatial databases*, SSTD, 2005, pp. 273–290.
- [8] S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford, *Okapi at trec-3*, TREC-3 (Gaithersburg, USA), 1994, pp. 109–126.
- [9] E. Safra, Y. Kanza, N. Dolev, Y. Sagiv, and Y. Doytsher, *Computing a k-route over uncertain geographical data*, SSTD, 2007, pp. 276–293.
- [10] G. Salton and M.J. McGill, *Introduction to modern information retrieval*, McGraw-Hill, 1983.
- [11] H. Samet, J. Sankaranarayanan, and H. Alborzi, *Scalable network distance browsing in spatial databases*, ACM SIGMOD, 2008, pp. 43–54.
- [12] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, *A road network embedding technique for k-nearest neighbor search in moving object databases*, GeoInformatica **7** (2003), no. 3, 255–273.
- [13] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, *Optimal sequenced route query*, VLDBJ **17** (2008), no. 8, 765–787.
- [14] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis, *Constrained shortest path computation*, SSTD, 2005, pp. 181–199.

Appendix C

Temporal Route Interactive Planning (TRIP)

Temporal Route Interactive Planning (TRIP)

Roy Levin and Yaron Kanza

February 24, 2010

C.1 Introduction

Route-search is the task of searching for a list of objects, which represent real world entities that are to be visited by a route, based on a set requirements specified in the form of a query issued by a user. In this preliminary work we define the problem of route-search in the presence of temporal and order constraints and formulate it as a mixed integer programming (MIP) problem. A MIP formulation is a linear programming (LP) formulation with an additional constraints requiring that some of the variables will be assigned integer values in our case zero or one. This formulation has two main goals:

Finding an optimal route for queries issued over small scale datasets.

The straightforward use of such a formulation is for being able to find an optimal route for small scale datasets using a MIP solver. This is not expected to work for larger datasets since the MIP problem is NP-Hard.

Finding a satisfactory route for queries issues over large scale datasets.

By removing the integer constraints the problem can be solved in polynomial time using a LP solver. The solution to the LP problem can serve as an heuristic for finding satisfactory routes. Possible heuristics will be discussed in the final section of this appendix.

Once we have a component that can answer route-search queries, we may use it either in an interactive or non-interactive environment. In a non-interactive environment, a query is issued and a route that satisfies the query is found using one of the methods suggested above. The user will follow that route from start to end. In an interactive environment, instead of returning a route as an answer to the query, only the next object to be visited is returned. The route is then continuously recalculated based on user feedback. User feedback is used to specify whether the visited object satisfies the user. Using one of the methods above, a route can be recalculated after every negative feedback from the user.

C.2 Route Search with Temporal and Order Constraints

In this section, we show how to formulate a route search in the presence of temporal and order constraints as MIP and LP problems. Consider a dataset containing spatial objects that represent real-world entities and a set of roadways connecting these spatial objects. Suppose the dataset includes an estimation of the time it should take to travel from one point to another. Units of time throughout this work are normalized to the interval $[0, 1]$, where 0 represents the time in which the query is issued and 1 represents a higher bound on the arrival time at any object, i.e. this represents a time window within which all the selected objects of a route are to be visited. A user issues a query indicating his requirements and the goal is to find the route whose overall travel time is minimal and that satisfies the query. The user's query contains a start and end location, a set of subqueries and a list of order constraints. Each subquery contains a search term, a minimal satisfaction probability threshold, a duration

and optionally a time window specifying when to arrive, depart and how much time to spend at spatial entities corresponding to the spatial objects matching the search term. The minimal satisfaction probability threshold of τ indicates that for a route to satisfy the user it must go via a set of objects having a total probability of no less than τ to satisfy the user. The list of order constraints is specified in the form of a list of pairs of references to subqueries. Each pair indicates that objects matching the search term of the subquery on its left hand side must precede those matching the search terms on its right hand side. The following notations define the constraints imposed by the user's query and the properties of the spatial objects in the map. This constitutes the input to the MIP or LP solver.

1. s and t which are the start and an end location derived from the query.
2. A set of all the spatial objects in the dataset indexed $\overline{0, n+1}$ where s is represented by 0 and t is represented by $n+1$.
3. A set of search subqueries Q_1, \dots, Q_m , present as part of the user's query, which yield a set of disjoint object index sets $C = \{c_1, \dots, c_m\}$, representing different categories.
4. The following holds for each category; $\forall i \in \overline{1, m} : c_i = \{k_{i,1}, k_{i,2}, \dots, k_{i,n_i}\}$. Each $k_{i,j}$ is the index of an object in the dataset, e.g. $\forall i \in \overline{1, m}, \forall j \in \overline{1, n_i} : k_{i,j} \in \overline{1, n}$.
5. The probability of each spatial object to satisfy the user with respect to his query, $\forall j \in \overline{1, n} : p_j \in (0, 1)$. This information can be obtained by evaluating the match between each spatial object in the dataset and the subqueries in the user's query.
6. A minimal probability threshold that must be achieved for each category; $\forall c_i \in C : \tau_i \in (0, 1)$.
7. A set of order constraints $\Omega = \{(c_i, c_j) | c_i, c_j \in C\}$ such that for each $(c_i, c_j) \in \Omega$, c_i must be visited before c_j .
8. Time windows for each spatial object defining the time frame in which the spatial object is available. Hence $\forall i \in \overline{0, n+1} : e_i$ and l_i respectively define the earliest and latest times the object index by i is available. These may be derived from the user's query or the spatial object's properties, e.g. opening closing hours.
9. The minimal duration at each spatial object is given by $d_i, \forall i \in \overline{1, n}$.
10. $\forall i \in \overline{0, n}$ and $\forall j \in \overline{1, n+1}$ the time required to travel from the node indexed by i to the node indexed by j is given by t_{ij} . This information is assumed to be present in the dataset.

We propose two MIP formulations, The second is superior when the number of roadways is much larger than the number of spatial objects. We discuss the circumstances in which this occurs when we present the second approach.

C.2.1 The first formulation

Next, we present our first formulation of the problem as a MIP. The following tables list all the notations we used above and the variables the are to be assigned values by the MIP solver.

Notations	
$C = \{c_1, c_2, \dots, c_m\}$	categories to be visited
$\forall c_i \in C : c_i = \{k_{i,1}, k_{i,2}, \dots, k_{i,n_i}\}$	node indexes per category
$\forall c_i, c_j \in C, c_i \cap c_j = \phi$	categories are disjoint
$N = \overline{0, n+1}$	indexes to spatial objects, with $0=s, n+1=t$
$\forall i \in \overline{1, n} : p_i \in (0, 1)$	matching probabilities
$\forall c_i \in C : \tau_i \in (0, 1)$	minimal probability threshold for category i
$\Omega = \{(c_i, c_j) c_i, c_j \in C\}$	order constraints
$\forall i, j \in N : t_{ij}$	travel time from i to j
$\forall i \in N : d_i$	delay time at node i
$\forall i \in N : e_i$	earliest arrival time at node i
$\forall i \in N : l_i$	latest departure time from node i

Decision variables	
$\forall i, j \in N : x_{ij} \in \{0, 1\}$	1 iff path goes from node i to j
$\forall i \in N : \alpha_i \in [0, \infty)$	arrival time at node i

The MIP solver finds an assignment to the variables above, which satisfies the objective and the constraints that follow next. Constructing a route from the variable assignments is simple. For every x_{ij} that is assigned a value of 1 the road leading from the spatial object indexed i to the one indexed j should be included in the route. Selecting roads in such a way will yield a valid route that satisfies all the constraints of the user query and minimizes the overall travel time. The $\alpha_i^{(k)}$ represent the arrival time at the object indexed i . If the object is not visited this value will be zero.

Objective: Minimize total travel time

$$\min(\alpha_{n+1} - \alpha_0)$$

Linear constraints:

(1) Must leave any node we enter (except s and t)

$$\forall j \in \overline{1, n} : \sum_{i \in \overline{1, n}} x_{ij} - \sum_{k \in \overline{1, n}} x_{jk} = 0$$

(2) Source/target do not have incoming/outgoing edges

$$\sum_{i \in N} x_{i,0} = 0, \sum_{j \in N} x_{n+1,j} = 0$$

(3) Do not allow visiting a node more than once

$$\forall j \in N : \sum_{i \in N} x_{ij} \leq 1$$

(4) Make sure all categories are visited

$$\forall c \in C : \sum_{i \in c} \sum_{j \in N/c} x_{ij} \geq 1$$

(5) Make sure minimal probability threshold is enforced for each category. Here we assume that $p_i \in (0, 1)$ holds for each $p_i \in P$. This is a reasonable assumption since we can filter out any node with a probability of 0, and we assume the probability for any node can be close to but never actually 1 (this can be done by reducing a small positive ϵ).

$$\forall c_k \in C : 1 - \prod_{i \in c} (1 - p_i)^{\sum_{j \in N} x_{ij}} \geq \tau_k$$

To make constraint 4 linear we proceed as follows:

$$\forall c_k \in C : \prod_{i \in c} (1 - p_i)^{\sum_{j \in N} x_{ij}} \leq 1 - \tau_k$$

$$\lg(1 - p_{i_1})^{\sum_{j \in N} x_{i_1 j}} + \dots + \lg(1 - p_{i_{|c|}})^{\sum_{j \in N} x_{i_{|c|} j}} \leq \log(1 - \tau_k)$$

$$\lg(1 - p_{i_1}) \sum_{j \in N} x_{i_1 j} + \dots + \lg(1 - p_{i_{|c|}}) \sum_{j \in N} x_{i_{|c|} j} \leq \log(1 - \tau_k)$$

$$\forall c \in C : \sum_{i \in c} \sum_{j \in N} \lg(1 - p_i) x_{ij} \leq \log(1 - \tau_k)$$

(6) Valid arrival times at each node

$$\forall i \in N : \alpha_i \geq x_i \cdot e_i, \alpha_i \leq (l_i - d_i) \cdot x_i$$

(7) Coherent visiting time at each node

$$\forall j \in \overline{1, n+1} : \alpha_j \geq \sum_{i \in N} (\alpha_i + d_i + t_{ij}) x_{ij}$$

To make constraint 6 linear we define a large constant value M and rewrite it as follows:

$$\forall i \in N, \forall j \in \overline{1, n+1} : \alpha_j \geq \alpha_i + d_i + t_{ij} - M \cdot (1 - x_{ij})$$

(8) Presence of order constraints

$$\forall (c_l, c_r) \in \Omega : \forall i \in c_l, \forall j \in c_r : \alpha_i \leq \alpha_j$$

C.2.2 A better formulation

Note that the spatial objects present in our dataset are not the same as nodes that can be found in a standard digital map. A node in a standard digital map represents an intersection of multiple roadways, e.g. it has no meaning in terms of matching search terms in user subqueries. Since our dataset contains only nodes that represent entities that can match search terms of a query (i.e. not intersections), a roadway between them does not exist. As a result, there is no

direct information about the amount of time t_{ij} required for traveling from the object indexed i to the object indexed j . The best way to determine t_{ij} for each pair of spatial entities is to compute them offline, in advance. As a result, the number of t_{ij} 's pairs will be in the order of n^2 , as there are n objects in the dataset. The problem with our first approach is that the number of decisions and constraints is, thus, proportional to n^2 . This significantly impacts the running time required for a MIP or LP solver to find a solution. To improve the efficiency of our approach we introduce a new scalar K which represents the expected number of objects in the route, and reformulate the model so that it will include less variables and constraints.

New decision variables

$\forall i \in \overline{0, n+1}, \forall k \in \overline{1, K} : x_i^{(k)} \in \{0, 1\}$	1 iff path goes via node i as its k 'th node
$\forall i \in \overline{0, n+1}, \forall k \in \overline{1, K} : \alpha_i^{(k)} \in [0, \infty)$	arrival time at node i as the k 'th node

Here, we also use a MIP solver to find an assignment to the variables above in order to satisfy the objective and the constraints that we present next. To construct a route from the value assignments to the variables above, for every $x_i^{(k)}$ that is assigned a value of 1, the spatial object indexed i is selected as the k th waypoint of the route. The $\alpha_i^{(k)}$ represents the arrival time at the object that is indexed i and is visited as waypoint k . If the object is not visited as waypoint k , this value will be zero.

Objective: Minimize total travel time

$$\min \left(\sum_{k=2}^K \alpha_{n+1}^{(k)} - \alpha_0^{(1)} \right)$$

Linear constraints:

(1) Path starts at the source s

$$x_0^{(1)} = 1$$

(2) No node exists directly after the target t

$$\forall i \in \overline{1, n} : x_{n+1}^{(k)} \cdot x_i^{(k+1)} = 0$$

To make this equation linear we rewrite it as follows

$$x_{n+1}^{(k)} + x_i^{(k+1)} \leq 1$$

(3) The number of nodes visited at step k equals to the number of nodes visited at step $k+1$ (excluding the target t)

$$\forall k \in \overline{1, K} : \sum_{i \in \overline{0, n}} x_i^{(k)} = \sum_{j \in N} x_j^{(k+1)}$$

(4) Do not visit more than a single node at each step

$$\forall k \in \overline{1, K} : \sum_{i \in N} x_i^{(k)} \leq 1$$

(5) Visit every category at least once

$$\forall c \in C : \sum_{k \in \overline{1, K}} \sum_{i \in c} x_i^{(k)} \geq 1$$

(6) Enforce a minimal probability threshold. Similar to step 5 in the previous formulation, after linearization we get:

$$\forall c_l \in C : \sum_{k \in \overline{1, K}} \sum_{i \in c} \lg(1 - p_i) x_i^{(k)} \leq \lg(1 - \tau_l)$$

(7) Valid arrival times at each node

$$\forall i \in N, \forall k \in \overline{1, K} : \alpha_i^{(k)} \geq x_i^{(k)} \cdot e_i$$

$$\forall i \in N, \forall k \in \overline{1, K} : \alpha_i^{(k)} \leq x_i^{(k)} \cdot (l_i - d_i)$$

(8) Coherent arrival times

$$\forall k \in \overline{1, K}, \forall j \in \overline{1, n+1} :$$

$$\alpha_j^{(k+1)} \geq \sum_{i \in N} (\alpha_i^{(k)} + d_i + t_{ij}) \cdot x_i^{(k)} \cdot x_j^{(k+1)}$$

To linearize this equation we proceed as follows:

$$\begin{aligned} \alpha_j^{(k+1)} &\geq x_j^{(k+1)} \sum_{i \in N} (\alpha_i^{(k)} + d_i + t_{ij}) \cdot x_i^{(k)} = \\ &= x_j^{(k+1)} \left[\sum_{i \in N} \alpha_i^{(k)} + \sum_{i \in N} (d_i + t_{ij}) x_i^{(k)} \right] \rightarrow \\ \alpha_j^{(k+1)} &\geq \sum_{i \in N} \alpha_i^{(k)} + \sum_{i \in N} (d_i + t_{ij}) x_i^{(k)} - M \cdot (1 - x_j^{(k+1)}) \end{aligned}$$

We can set the value of M for each constraint as

$$M = 1 + \max_{i \in N} (d_i + t_{ij}) + \sup(\alpha_i^{(k)})$$

we can estimate $\sup(\alpha_i^{(k)})$ by summing up the $k - 1$ largest edges where only one of them has the source s as its left hand side and none of the edges has the target t in either of its sides.

(9) Presence of order constraints

$$\begin{aligned} \forall (c_l, c_r) \in \Omega : \forall i \in c_l, \forall j \in c_r : \\ \sum_{k \in \overline{1, K}} \alpha_i^{(k)} \leq \sum_{k \in \overline{1, K}} \alpha_j^{(k)} \end{aligned}$$

The problem with our final equation introduces $|c_l| \cdot |c_r|$ new constraints for each ordering. This number can be rather large. The following observations allow us to rewrite this equation using less constraints:

$$\forall (c_l, c_r) \in \Omega, \forall k' \in \overline{1, K} :$$

$$\sum_{i \in c_r} x_i^{(k')} = 1 \Rightarrow 1 - \prod_{k=1}^{k'} \prod_{j \in c_l} (1 - p_j)^{x_j^{(k)}} \geq q_l$$

After linearization we get (by extracting the logarithm using similar steps as shown before):

$$\sum_{i \in c_r} x_i^{(k')} = 1 \rightarrow \sum_{k=1}^{k'} \sum_{j \in c_l} (x_j \cdot \lg(1 - p_j)) - \lg(1 - q_l) \leq 0$$

We can now express this as a simple linear equation

$$\forall (c_l, c_r) \in \Omega, \forall k' \in \overline{1, K'} :$$

$$\sum_{k=1}^{k'} \sum_{j \in c_l} (x_j \cdot \lg(1 - p_j)) - \lg(1 - q_l) - M \cdot \left(1 - \sum_{i \in c_r} x_i^{(k')}\right) \leq 0$$

We now set $m = -2\lg(1 - q_l)$ and get the following constraint:

$$\begin{aligned} \sum_{k=1}^{k'} \sum_{j \in c_l} (x_j \cdot \lg(1 - p_j)) - \lg(1 - q_l) + 2\lg(1 - q_l) \cdot \left(1 - \sum_{i \in c_r} x_i^{(k')}\right) &\leq 0 \\ \sum_{k=1}^{k'} \sum_{j \in c_l} (x_j \cdot \lg(1 - p_j)) + \lg(1 - q_l) - 2\lg(1 - q_l) \cdot \sum_{i \in c_r} x_i^{(k')} &\leq 0 \end{aligned}$$

C.2.3 Validity

To confirm the validity of this approach we implemented it using the Microsoft Solver Foundation Express Edition Framework¹. We generated a few datasets containing less than 10 spatial objects with time windows specified for each of them and added roadways with time values to connect every pair of objects. We then issued multiple queries with different start and end locations and a set of categories to visit. We ran an algorithm that formulated the problem as proposed in our second approach and ran a MIP solver provided by the framework. We confirmed that in every case the algorithm found the fastest route, i.e. the route with the shortest travel time, that satisfies all the constraints.

C.2.4 Changing traffic conditions

We intend to further develop our formulation in order to deal with changes in traffic. In the original setting, t_{ij} is constant. The MIP can easily deal with t_{ij} s that are themselves linear functions. Let $S_{ij}(t)$ denote a set of samples that correspond to the time it takes to travel from the object indexed by i to the object indexed by j , given that the arrival time at the object indexed i was t . Let $T_{ij}(t)$ be some function, of all the samples, which yields an estimation of the time required to travel from the object index i to that indexed j , where t

¹<http://code.msdn.microsoft.com/solverfoundation>

is the arrival time at object i . Given a route that satisfies the user's query and contains an object indexed i , let $BL(i)$ and $BH(i)$ denote the earliest and latest times, respectively, by which a user can arrive at or depart from the object indexed i . An approach we intend to examine is of approximating $T_{ij}(t)$ within the time window $[BL(i), BH(i)]$ by the linear function $T'_{ij}(t)$.