Expressing and processing complex preferences in route planning queries: Towards a fuzzy-set-based approach

A. Hadjali, A. Mokhtari, O. Pivert*

IRISA/ENSSAT, University of Rennes 1, 6, Rue de Kerampont, BP 80518, 22305 Lannion Cedex, France

Abstract

In this paper, we propose a contribution for a new generation of route planners able to deal with complex and sophisticated preferences. Fuzzy set theory is advocated as a framework for modelling preferences. First, a typology of user preferences that make sense in the context of unimodal route planning is investigated. The bipolar nature of such preferences is discussed as well. The foundations of both a formal language and an SQL-like language dedicated to bipolar route planning queries are then presented and illustrated with different examples. The basic components of the architecture of the system proposed are described and deep details about query evaluation are provided. Finally, the approach is evaluated by means of a set of experiments.

© 2012 Elsevier B.V. All rights reserved.

Keywords: Route planning; Preferences; Bipolarity; Fuzzy sets

1. Introduction

Route planners are systems which aim at computing the “best” route, according to the needs of a user, between two locations A and B, using different types of information about a given road network: structural information, of course, but also, in some cases, real-time data about traffic jams, road conditions, etc.

Even though there now exists a relatively large set of commercially available route planners, these systems have very limited capabilities—if any—when it comes to taking into account sophisticated user preferences. For instance, let us consider a person who wants to go from Paris to Munich, preferably by a route which is inexpensive, short and passes about halfway near a restaurant accessible to disabled persons. To the best of our knowledge, there does not exist any system able to process such a query. Currently, in the most “intelligent” commercial systems and research prototypes: (i) only a small set of predefined preferences (e.g., avoid turnpikes, prefer freeways) are made available to the user who can sometimes attach weights to them and (ii) spatial preferences (e.g., prefer routes passing near a restaurant accessible to disabled persons) cannot be captured.

Roger and Langley [1–3] propose a route planning system based on learning preferences from user feedback. Letchner et al. [4] develop a route planner, named TRIP, which leverages users experiences within a region. In [5,6], a case-based route planning approach is proposed where the preference model is not fixed. Let us also mention the work done by Balke et al. [7,8] where user preferences are explicitly represented. They propose a route planning system integrating preferences over four characteristics of a route: length, traffic jams, road works and weather conditions. To aggregate the scores

* Corresponding author.
E-mail addresses: hadjali@enssat.fr (A. Hadjali), mokhtari@enssat.fr (A. Mokhtari), pivert@enssat.fr, olivier.pivert@enssat.fr (O. Pivert).

0165-0114/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2012.01.006

related to these preferences into an overall degree, a weighted function is used. In response to a route planning query, the top $k$ results are delivered to the user.

Building a personalized route planner is not an easy task, for several reasons. The large size and complexity of modern road networks, and the wide variety of possible user preferences are important difficulties to overcome. In this paper, we describe a fuzzy-set-based approach to the modelling and handling of sophisticated route planning queries involving complex user preferences. Fuzzy set theory has already been used to deal with some transportation issues; in particular route choice, where several models based on fuzzy sets have been proposed in order to take into account uncertainty and the subjectivity of the user [9,10].

The choice of this theoretical framework is motivated by the fact that it is important to develop a user-friendly query language. Fuzzy set theory is very well suited to the interpretation of linguistic terms, which constitute a convenient way for a user to express her/his preferences. Moreover, fuzzy set theory provides us with a very rich range of connectives that can capture the different user’s attitudes concerning the way the different (flexible) criteria present in her/his query compensate or not. Also, as it will be discussed further, preferences in the unimodal route planning context are inherently bipolar and fuzzy sets are suitable for taking into account this aspect. Three main contributions are brought throughout this work: (i) a typology of user preferences that make sense in the context of unimodal route planning is proposed, (ii) the basis of both a formal and an SQL-like language dedicated to route planning queries with spatial and global bipolar preferences are investigated, and (iii) an efficient and rational technique for evaluating bipolar route planning queries is proposed.

The remainder of the paper is organized as follows. Section 2 deals with some representational issues and provides a comprehensive survey on bipolarity. In Section 3, we present a typology of preferences in a route planning context. Section 4 describes the foundations of the query languages proposed. Section 5 provides an overview of the architecture of the route planner we propose and discusses an efficient query evaluation strategy based on an approximation technique. Section 6 gives an illustrated example. In Section 7, we present some experimental results. In Section 8, we summarize related work and highlight the main aspects that distinguish our proposal from these works. Finally, Section 9 recalls the main contributions and discusses some directions for future work.

2. Preliminary notions

2.1. Representation issues

Modelling user preferences requires a clear representation of concepts, relations and geographic entities which can be involved in a route planning query. Different spatial data models have been proposed in the literature, and road networks can be represented in several ways [11]. This section provides some insights about these notions.

2.1.1. Road network modelling

A road network is generally modelled as a (directed) graph. Several granularity levels can be considered, according to what the components of the graph (vertices and edges) represent. In the case of an interurban road network, a vertex represents a city, and an edge describes an interurban link (freeway, national roads, etc.). However, in an urban road network, a vertex represents an intersection between two or more routes, and an edge corresponds to a street.

In our work, we consider the finest granularity: a vertex represents any intersection/bifurcation between routes or dead ends, and an edge corresponds to a road of any kind.

2.1.2. Geographic data files (GDF)

We assume a priori that any spatial feature may be concerned by a user preference. Thus, interpreting and processing queries involving such a wide range of potential preferences implies to have available a model suited to the representation of these features. Geographic data files (GDF) [11] is an international standard that specifies the conceptual and logical data model for geographic databases for intelligent transportation systems (ITS) applications. It includes a specification of potential contents of such databases (features, attributes and relationships), a specification of how these contents shall be represented, and of how relevant information about the database itself can be specified (meta data).

---

1 For a given origin–destination pair and a given transport mode, the route choice problem (also called traffic assignment) deals with identifying which route a given traveller would probably take.
2.2. About bipolarity

Generally speaking, bipolarity refers to the existence of positive and negative information. Negative information discards situations as being impossible, forbidden, absolutely ruled out. Positive information supplies examples, legal cases, preferred choices. During the last few years, bipolarity has been studied in reasoning [12,13], decision making [14,15] and in databases and information systems [16–19]. In database flexible querying, bipolarity corresponds to dealing with two kinds of preferences: positive and negative ones.

2.2.1. Constraints vs. wishes

Fuzzy constraints correspond to negative preferences of the user, in the sense that their complements define fuzzy sets of values that are rejected as being non-acceptable. These constraints should be combined conjunctively, thus acknowledging the fact that they are constraints. The second type of preference is qualified as being positive in the sense that it does not express a constraint, but only a desire, a wish, a recommendation that is more or less strongly suggested. The satisfaction of some of the wishes should give some bonus to the corresponding solutions (provided that they also satisfy the constraints, if any). Wishes are not compulsory and can be combined disjunctively.

Now, let $C_i$ (respectively $W_i$) be a (flexible) constraint (respectively wish) on the attribute $A_i$ (with $U_i$ as a domain of values). It has been shown in [17] that the pair $(C_i, W_i)$ must obey a minimal consistency requirement, i.e.,

$$\forall u_i \in U_i, \quad \mu_{C_i}(u_i) \leq \mu_{W_i}(u_i),$$

which expresses that the wish must be included in the constraint (in the sense of Zadeh). Then, the wish can be used to discriminate between the tuples that somewhat satisfy the constraint. It may happen that this consistency condition does not hold (it is the case when $C_i$ and $W_i$ concern different attributes). In this case, one can restore it by considering $(C_i, C_i \land W_i)^3$ instead of $(C_i, W_i)$.

2.2.2. Ranking bipolar query results

Let $Q$ be a bipolar query over a database containing $n$ attributes $A_1, \ldots, A_n$. $Q$ involves then two components of preferences, one pertaining to simple wishes (denoted by $P_W$), and the other expressing constraints (denoted by $P_C$). As pointed out in [17], there exist different methods for rank-ordering the answer tuples when preferences are handled in a bipolar way. Hereafter, we review some of them. To this end, and for the sake of clarity and brevity, we assume that the set of preferences involved in $Q$ is simply represented by a set of pairs

$$\{(C_i, W_i), i = 1, \ldots, n\},$$

This means that $P_C = \{C_i, i = 1, \ldots, n\}$ and $P_W = \{W_i, i = 1, \ldots, n\}$. Note that it may happen that for some attribute $A_i$ there is no constraint. In such a case $C_i = \text{True}$, i.e., $\forall u_i \in U_i, \mu_{C_i}(u_i) = 1$. If no value is particularly wished in $U_i$ then $W_i = \text{False}$, i.e., $\forall u_i \in U_i, \mu_{W_i}(u_i) = 0$.

In this study, we use the lexicographic-order-based ranking. It can be applied using different kinds of preference aggregations as explained below.

- **Min–max-based aggregation**: For a given tuple $u = (u_1, \ldots, u_n)$, we compute a pair of satisfaction degrees in the following way:

$$\left(\mu_{P_C}(u), \mu_{P_W}(u)\right) = (\otimes_i \mu_{C_i}(u_i), \oplus_i \mu_{W_i}(u_i)),$$

where $\otimes$ (respectively $\oplus$) is a t-norm (respectively a t-conorm) modelling a conjunctive (respectively disjunctive) combination. Using min and max respectively, it reflects the extent to which $u$ satisfies all the constraints and at least one wish. To rank-order the query results, a solution is to apply the lexicographic order, by using $\mu_{P_C}(u)$ as a primary criterion and $\mu_{P_W}(u)$ as a secondary one for breaking ties.

---

2 We adopt here the approach to bipolarity proposed by Dubois and Prade [17].

3 The symbol $\land$ stands for a conjunction operation which can be modelled by a t-norm operator (e.g., the min or the product operator).
Leximin–leximax-based aggregation: The above procedure can be improved by considering that satisfying several wishes is certainly better than satisfying just one. To this end, we use a double lexicographic ranking procedure by applying

- the leximin order on the $C_i$’s and,
- the leximax order on the $W_i$’s.

The principle of the leximin ordering is as follows. Let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ be two tuples and let $e(u) = (x_1, \ldots, x_n)$ and $e(v) = (y_1, \ldots, y_n)$ respectively be their evaluations. First, $e(u)$ (resp. $e(v)$) is reordered in a non-decreasing way: $e^*(u) = (x_1^*, \ldots, x_n^*)$ with $x_1^* \leq \cdots \leq x_n^*$ (respectively $e^*(v) = (y_1^*, \ldots, y_n^*)$ with $y_1^* \leq \cdots \leq y_n^*$). Leximin is then defined such as: $u \succ_{\text{leximin}} v$ iff $\exists k \leq n, \forall i < k, x_i^* = y_i^*$ and $x_k^* > y_k^*$.

Leximax can be defined in the same way but once the two vectors $e(u)$ and $e(v)$ have been reordered in a non-increasing order. In this way, we refine the min-based ordering on the constraints and the max-based ordering on the wishes.

3. Typology of user preferences

From a formal point of view and from common sense, in the context of unimodal point-to-point route planning queries, three families of user preferences may be distinguished [20,21]:

- Spatial preferences, which aim at capturing the wish of a user to pass or not by a particular road(s), place(s) or some part(s) of road network. For example, “avoid secondary roads”.
- Global preferences, which concern some properties of a route seen as a whole, such as comfort, length, duration or safety. For example, “prefer a fast route”.
- Spatio-temporal preferences, which are spatial (resp. global) preferences involving a time component whose purpose is to express the moment or period when the spatial (respectively global) preference is relevant. An example is “avoid the city centre around noon”.

Hereafter, we present each family of preferences in more details.

3.1. Spatial preferences

Spatial preferences represent the largest class of preferences in terms of diversity. Every entity of the road network or in its environment may be concerned by such a preference.

Definition (Spatial preferences): a preference is said to be spatial if it concerns a geographic entity, i.e., an entity which has spatial coordinates.

Two approaches exist for designating road network elements concerned by a spatial preference. The explicit approach consists in designating the different entities concerned by the preference by means of their inherent, individual, properties. For example, avoiding secondary roads or turnpikes.

On the other hand, the implicit approach uses references to other spatial entities, by means of spatial relations (i.e., relations between two or more spatial objects) so as to identify the elements of a road network to favour (respectively disfavour). An example is, prefer routes which have gas stations along them. Several classes of spatial relationships can be involved in such preferences: Topological (such as along, adjacent, inside, disjoint), Spatial (strict) order (such as behind, in front of, left to, right to), Metric (such as near, far) or directional relationships (such as north, northeast, south) [22,23]. Some of them are intrinsically fuzzy like near or far [24]. Spatial entities used as references can be either an object in the neighbourhood of the network or entities representing an induced part of the road network, that we call a zone in the following. A zone is a geographic area, for instance an administrative area. Preferences on a zone are about the environment that surrounds a route. An example is, prefer a route which passes across Luxembourg.

3.2. Global preferences

A route can be seen as a particular spatial feature within the framework we propose. Contrary to spatial and spatio-temporal preferences which can be called local inasmuch as they concern a part of a route, global preferences characterize the route as a whole. They involve qualitative criteria such as expensive, fast, safe, and so on. Among the most intuitive
properties of a route, let us mention: rapidity, length, safety, cost and comfort. Other properties may be of interest in some specific contexts, for example: robustness (for military applications or rescue services) or pollution produced.

3.3. Spatio-temporal preferences

Time has an important place in route planning since the duration of a trip is in general a major factor as to the satisfaction of the user. Thus, it is essential to enable the user to express his/her preferences regarding this aspect. A spatio-temporal preference involves both spatial and temporal entities. Thus, a spatio-temporal preference \( p_{st} \) can be seen as a complex preference, formed of a spatial preference \( p_s \) (respectively global preference \( p_g \)) and a temporal preference \( p_t \). In the context of a spatio-temporal preference, time is used to express the validity period of a spatial preference (respectively global preference).

Let us note that two types of temporal preferences can be involved in a route planning query: quantitative and qualitative. Quantitative preferences express absolute bounds or restrict the temporal distance between two instants. In other terms, they express preferences on the duration of events or their timing. A quantitative preference can be unary or binary:

- **Unary form**, expresses a constraint on an instant \( i \) by means of a (set of) interval(s), and are expressed as: \( (i \in I) \).
- **Binary form**, concerns two instants \( i_1 \) and \( i_2 \) and constrains the distance \( i_2 - i_1 \); \( (i_2 - i_1) \in I \). For instance, find a route from Paris to Rennes such that the duration of the trip is less than **three hours**.

Qualitative preferences provide a means to specify the relative position of a pair of temporal entities (instants or intervals) \( i_1 \) and \( i_2 \) [21]. For instance, find a route from Paris to Berlin, with the preference to **arrive before night**. Processing such a query requires comparing the arrival time with the linguistic term **night**.

In this paper, due to space limitation, we restrict the scope to time-independent queries. So, in the sequel, only the first two families of preferences are considered.

**Remark 1.** Besides, we advocate that preferences in route planning are inherently bipolar (in the sense introduced in Section 2.2). Then, we distinguish between preferences expressing (flexible) constraints (describing what is acceptable), \( P_C \), and the ones representing wishes (i.e., what is really satisfactory), \( P_W \). For instance, in the query **look for a route between the city A and the city B which is not too expensive and which if possible passes near a gas station**, a route that does not pass near a gas station must not be systematically rejected since it could interest the user provided that the constraint pertaining to the cost is satisfied. So, some route query criteria are **not mandatory** but **only desirable**.

4. Route planning query language with preferences

Taking into account complex user preferences in the context of a route planning task requires: (i) the definition of a language allowing the user to express his/her query, on the one hand; (ii) the development of a route planning system able to parse, understand and efficiently process such queries on the other hand. This section discusses in depth the language allowing the user to express his/her query, on the one hand; (ii) the development of a route planning system based on tuple relational calculus (TRC)\(^4\) [26].

4.1. Basic notations

A route \( r \) is a finite sequence \( (\omega_1, \omega_2, \ldots, \omega_n) \) of road segments where \( \omega_1 = (\delta_d, \delta_1) \), \( \omega_n = (\delta_{n-1}, \delta_a) \) and \( \omega_i = (\delta_{i-1}, \delta_i) \), \( 1 < i < n \). \( \delta_d \) and \( \delta_a \) represent respectively the departure and the arrival points.

Let \( S\) (Route, Segment, \( R_1, R_2, \ldots, R_n \)) be a spatial database schema where the \( R_i \)'s are spatial relations which represent a road network as well as spatial objects in its neighbourhood such as gas stations, restaurants, etc. (see for instance their specifications in [111]). Route and Segment are relations which represent respectively the routes

\(^4\) Recall that any variable in TRC represents a tuple of a relation.
some of the routes, as explained in Section 5) linking two given locations \( \delta_d \) and \( \delta_u \), and the road segments which compose them. In the following, the schemas of relations \textit{Route} and \textit{Segment} are assumed to be respectively:

\[
(idRoute, \delta_d, \delta_u, \text{length}, \text{rapidity}, \text{safty}, \text{duration}, \text{cost})
\]

\[ (idSeg, idRoute, idRoad, \text{length}) \]

These relations are initialized by a preprocessing step which will be described further. Several other attributes could be added to \textit{Route} and \textit{Segment}, see, e.g., [11] for more details. Let \( \Phi \) denote the set of spatial relationships (topological, metric, directional, etc. [23,22]) supported by the spatial database considered. For instance, \( \Phi \) can contain relations such as close to, along, etc.

### 4.2. Route query language

In the context considered here, the general form of a top-\( k \) fuzzy spatial query expressed in TRC can write [27]:

\[
Q = [x/\text{COND}, k],
\]

where \( x \) is a tuple variable and \text{COND} is a formula having only \( x \) as a free variable. The result of such a query is a (fuzzy) set \( S \) of tuples \( x_i \) which (totally or partially) satisfy \text{COND}. Each tuple \( x_i \) is associated with a score \( \mu_i / x_i \), or a pair of scores \((\mu_i, \nu_i)/x_i\), expressing the extent to which it satisfies \text{COND}. As to \( k \), it denotes the desired number of answers in the result (the best ones, in accordance with the top-\( k \) query paradigm).

Syntax of \text{COND}. A term \( \tau \) (in the sense of TRC) involved in a formula is either a variable \( v \), a constant \( c \), or a function \( f(\tau_1, \tau_2, \ldots, \tau_m) \) where each \( \tau_j \) represents a term.

A formula is built from predicate calculus atoms, which can be of the following types:

- \( R(u) \), where \( R \) is a relation name and \( u \) is a tuple variable. This type of atom specifies the range of the tuple variable \( u \) as the relation whose name is \( R \). For instance: \text{gasStation}(u);
- \( u.A \theta \tau \) or \( \tau u.A \), where \( \theta \) is a crisp or fuzzy comparator (such as \textit{approximately equal}, \textit{much greater than}, \textit{much less than}, \textit{slightly greater than}, etc.), \( u \) is a tuple variable, \( A \) is an attribute from the relation on which \( u \) is defined, and \( \tau \) is a term of the same type as \( A \). For instance: \text{u.company} = “Esso”, where \( u \) is a tuple variable representing a gas station;
- \( u.A \pi \) where \( \pi \) is a (fuzzy or crisp) user-defined predicate. For instance: \text{u.cost} is \text{cheap};
- \( u_1.A_1 \phi u_2.A_j \), where \( \phi \in \Phi \) is a spatial relationship, \( u_1 \) and \( u_2 \) are two tuple variables defined on spatial relations, and \( A_1 \) and \( A_j \) are spatial attributes over which \( \phi \) is defined. For instance: \text{u_1.coordinates} close to \text{u_2.coordinates}.

A formula is inductively defined from the atoms according to the following rules:

- every atom is a formula;
- if \( p \) is a formula, then \( \neg p \) is also a formula where \( \neg \) stands for fuzzy negation;
- if \( p_1 \) and \( p_2 \) are formulas, then \( p_1 \land p_2, p_1 \lor p_2 \) and \( p_1 \Rightarrow p_2 \) are also formulas, where \( \land \) and \( \lor \) denote a fuzzy conjunction and disjunction respectively. \( \Rightarrow \) is a fuzzy implication extending the usual (material) one (see [28] about the different families of such implications);
- if \( p \) is a formula, then so are:
  - \( (\forall u)[p(u)] \) where \( \forall \) is a universal quantifier \((\forall, \exists) \) and \( u \) is a tuple variable which is free in \( p \). See Example 1.
  - \( (\exists u)[p_1(u)] \) are \( p_2(u) \) where \( A \) is a fuzzy quantifier [29], e.g., \textit{most}, \textit{around_n}, \textit{at_least[n]}, \textit{at_most[n]}, and \( u \) is a tuple variable which is free in \( p_1 \) and \( p_2 \). For instance: \text{most u}[R(u) \land p_1(u)] \text{ are p_2(u)} \) reads “most of the tuples from relation \( R \) which satisfy \( p_1 \) also satisfy \( p_2 \)”. See Example 2.

As previously mentioned, route planning queries can be viewed as a particular class of spatial queries whose aim is to retrieve the best route(s) from one location to another taking into account a set of user-defined preferences. Let \( Q \) be a spatial query. \( Q \) is said to be an RPQ iff

- \( Q \) is a query over an instance of relation \textit{Route}. In other words, the result is made of tuples from \textit{Route};
- \text{COND} includes a condition on the departure place: \( r.\delta_d = \text{constant}_1 \);
- \text{COND} involves a condition on the arrival place: \( r.\delta_u = \text{constant}_2 \);

Expression of an RPQ. Then, a unimodal point-to-point route planning query involving a set of preferences \( P \), a departure place \( A \) and an arrival place \( B \), denoted by \( Q^A_B(P) \), can be expressed as follows:

\[
Q = \left[ r / \text{Route}(r) \land (r.\delta_d = A) \land (r.\delta_a = B) \land (P_C(r), P_W(r), k) \right]
\]

where \( P_c \) (respectively \( P_W \)) represents the set of conditions that the user views as (possibly flexible) constraints (respectively wishes), as pointed out in Remark 1 of Section 3.

Example 1. Let us consider a user who wants to find a route from \( A \) to \( B \) with the constraint that the route must be inexpensive, and the wish that it does not pass through Brittany. The corresponding predicates \( P_C \) and \( P_W \) are then:

\[
P_C(r) = \{ p_1 \} \text{ where } p_1 = (r.\text{cost is inexpensive})
\]

\[
P_W(r) = \{ p_2 \} \text{ where } p_2 = (\forall o, \exists ((\text{Segment}(o) \land \text{Area}(x)) \land (o.\text{idRoute} = r.\text{idRoute}) \land (x.\text{name} = \text{"Brittany"})) \Rightarrow (o.\text{coord outside x.perimeter})).
\]

Example 2. Let us now consider a user who wants to find a route from \( A \) to \( B \) with the constraint that the route must be as short as possible and the preference that most of the roads used must be coastal (this latter term being viewed as fuzzy, depending on the actual distance to the sea). The corresponding predicates \( P_C \) and \( P_W \) are then:

\[
P_C(r) = \{ p_1' \} \text{ where } p_1' = (r.\text{length is short})
\]

\[
P_W(r) = \{ p_2' \} \text{ where } p_2' = (\exists o ((\text{Segment}(o) \land (o.\text{idRoute} = r.\text{idRoute})) \text{ are (o.type is coastal)}).
\]

About this latter example, let us mention that several interpretations of fuzzy quantified statements can be found in the literature, see in particular [30,31].

4.3. RPQL: an SQL-like language

In the previous section, we have defined theoretical foundations of the route planning query language. However, using this language in its current form remains difficult. Now, in order to allow for an end-user to easily formulate her/his RPQ we propose an SQL-like language, named RPQL, which is less expressive than the formal version but more user friendly. The challenge here is to define a syntax that strikes a balance between expressiveness, computational complexity and ease of use.

As SQL, RPQL consists of a series of keywords, statements and clauses. The general form of an RPQ with preferences, using RPQL is as follows:

\[
\text{FIND } R \text{ ROUTES FROM } \text{departure place} \text{ TO } \text{arrival place} \text{ WHERE } \{ \text{constraints} \} \text{ (PREFERRING) wishes}\}
\]

where the clauses surrounded with braces represent optional blocks. Only departure place and arrival place blocks are required and represent simple addresses in the form of a string.

Example 3. Find a route from Paris town hall, whose address is: 71 av Henri Martin 75016 PARIS, to “La Tour Eiffel” at Champ de Mars 75007 PARIS.

\[
\text{FIND } R \text{ ROUTES FROM } '71 av Henri Martin 75016 PARIS' \text{ TO } 'Champ de Mars 75007 PARIS'
\]

The WHERE (respectively PREFERRING) clause represents a set of constraints (respectively wishes) built from a set of atomic preferences \( p_i \) of the following form:

\[
R.A \theta \tau \quad (1)
\]

\[
R.A \text{ is } \pi \quad (2)
\]

where $A$ represents an attribute of relation $R$. $\theta$ is a crisp or fuzzy comparator, $\tau$ a given constant (respectively variable) and $\pi$ a crisp or fuzzy predicate.

The syntax of a global preference represents a particular case of (1) and (2), where $R$ is the relation $Route$ and $A$ one of its attributes.

**Example 4.** Find a route with a cost less than 35 Euros, preferably a fast route from $X$ to $Y$.

\[
\text{FIND $k$ ROUTES FROM } X \text{ TO } Y \text{ WHERE } \text{route.cost} < 35 \text{ PREFERENCES route.duration is fast}
\]

As to spatial preferences, we distinguish between two different syntaxes according to their type (i.e., explicit or implicit). An explicit spatial preference writes:

\[
[\text{all | exists | most}] \text{ segment are } p_1 \odot p_2 \odot \cdots \odot p_n
\]

where $p_i, i = 1, n$ stand for an atomic preference on the $Segment$ relation, and $\odot$ denotes a (fuzzy) conjunction/disjunction operator. On the other hand, the syntax of an implicit spatial preference is:

\[
[\text{all | exists | most}] \text{ R such that } p_1 \odot p_2 \odot \cdots \odot p_m \text{ and route passes [near | inside | outside] R}
\]

Contrary to explicit spatial preferences, implicit spatial preferences concern any spatial relation and rely on spatial relationships. Moreover, atomic preferences, $p_i, i = 1, m$ involved in this kind of preference concern the $Segment$ relation as well as the spatial relation $R$.

**Example 5.** Find a route from $X$ to $Y$, avoiding freeways (constraint) and preferably passing near a French restaurant (wish).

\[
\text{FIND $k$ ROUTES FROM } X \text{ TO } Y \text{ WHERE all segment are segment.category /p1 ‘freeways’ exists restaurant such that (restaurant.type = ‘french’ and route passes near restaurant).}
\]

4.4. From RPQL to SQLf

For practical implementation reasons, it is extremely desirable to be able to translate an RPQL query into a query expressed in a bipolar variant of SQLf (a fuzzy version of SQL, see [32] for more details). To this aim, we provide hereafter a set of mapping rules that make this translation possible. The idea is to build an SQLf version of an RPQL query by mapping each expression, preference and clause of the latter into a fragment of the former thanks to a set of mapping rules.

Fig. 1 provides the general form of a bipolar SQLf query obtained by translating an RPQL query. One can observe that trip parameters (i.e., departure and arrival place) are translated into constraints.

In Tables 1 and 2, we summarize the different mapping rules for each clause and preference type present in an RPQL query. Departure and arrival place clauses are naturally translated into preferences which can be viewed as global preferences. As shown in Table 2, different mapping rules are needed to translate spatial preferences according to their type (implicit/explicit) and to the kind (crisp/fuzzy) of the quantifiers they involve.

**Remark 2.** RPQL can be seen as a fragment of (a bipolar variant of) SQLf. To formally prove that there is an inclusion between RPQL and SQLf, one can rely on the well-known equivalence between SQL and TRC in terms of expressivity.
Table 1
RPQL clauses and their translations in SQLf.

<table>
<thead>
<tr>
<th>RPQL clause</th>
<th>SQLf expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departure place route</td>
<td>route.( d = \text{departure place} )</td>
</tr>
<tr>
<td>Arrival place route</td>
<td>route.( a = \text{arrival place} )</td>
</tr>
</tbody>
</table>

Fig. 1. A bipolar SQLf query resulting from the translation of an RPQL query.

Table 2
Preference mapping rules.

<table>
<thead>
<tr>
<th>Preference type</th>
<th>RPQL version</th>
<th>SQLf version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>route.( A \theta \tau ) \text{ route.}A \text{ is } \pi</td>
<td>route.( A \theta \tau ) \text{ route.}A \text{ is } \pi</td>
</tr>
<tr>
<td>Explicit spatial</td>
<td>\textit{exists} segments are ( p_1 \odot \ldots \odot p_n )</td>
<td>\textit{exists} (\textit{SELECT} * \text{ FROM} \text{ segment inner join route} \text{ ON} \text{ segment.idRoute = route.idRoute} \text{ WHERE} \text{ ( p_1 \odot \ldots \odot p_n ) })</td>
</tr>
<tr>
<td>Implicit spatial</td>
<td>\textit{exists} ( R ) such that ( (p_1 \odot \ldots \odot p_m) \text{ and route passes [near</td>
<td>inside</td>
</tr>
</tbody>
</table>

This equivalence also holds between SQLf and TRCf (i.e., the fuzzy version of the TRC language as described in Section 4.2). Now, since RPQL is embedded in TRCf w.r.t. expressivity, the inclusion between RPQL and SQLf can be established. However, it is much more simple to express a route planning query in RPQL than it is in SQLf (cf. Example 6 below), since many joins and attributes from the geographic database are hidden in RPQL (and can thus be ignored by the user). As mentioned above, translating an RPQL query into an SQLf query is based on simple rules (cf. Table 2) and this translation step is totally negligible w.r.t. the overall query evaluation cost (as shown in the experimental part, cf. Section 7, the significant costs are related to the loading of the road network and the computation of the top-k answers).
Example 6. We give hereafter the SQLf version of the RPQL query of Example 5:

```
SELECT k, *
FROM route
WHERE route.δ_f = X and route.δ_a = Y
and all (SELECT *
FROM segment inner join route
ON segment.idRoute = route.idRoute)
are segment.category ≠ 'freeways'
PREFERENCE exists (SELECT *
FROM segment inner join route
inner join restaurant
ON segment.idRoute = route.idRoute and
segment.coord near restaurant.coord
WHERE restaurant.type = 'French')
```

5. System architecture and query processing

In the classical architecture of route planning systems, the database and the route planner (module) are considered to be two distinct components: the first stores and makes available a set of data, whereas the second computes the best routes.

As mentioned in Section 4, processing an RPQ involving preferences requires that a system be able to parse, understand and evaluate such queries. In this section we outline the architecture of a route planning system where a spatial database is the core of a route planning engine. Before we present the architecture of our system, two central points need to be discussed:

- how to define the semantics (in terms of trapezoidal membership functions, t.m.f.\(^5\)) of the fuzzy predicates which model preferences;
- how to proceed for finding the best answers.

5.1. Semantic issues

Regarding the semantics of the fuzzy predicates used, two kinds of predicates are considered:

**User-defined predicates.** The semantics of such predicates is explicitly given by the user. It is assumed that a route planning system is endowed with an interface that allows the user to define the t.m.f. of such fuzzy terms in a convenient and user-friendly way. An example of a condition involving such a predicate is: `route.cost` is high. Now, to define the t.m.f. of the predicate `high`, one can ask the user to specify its ideal values (that constitute the core of the t.m.f.) and acceptable values (that constitute the support of the t.m.f.) via a suitable interface. Denoting by \(I\)-values (respectively \(A\)-values) the ideal values (respectively acceptable values) for the attribute `Price` whose domain `\(\text{Dom(Price)} = [k_1, k_2]\)` three cases can be distinguished:

- **\(I\)-values** = \(\{v \in \text{Dom(Price)}, v \leq \alpha\}\) and
  \(A\)-values = \(\{v \in \text{Dom(Price)}, v \leq \alpha + \beta\}\).
  This corresponds to the t.m.f. defined as \((\alpha, 0, \alpha, \beta)\).
- **\(I\)-values** = \(\{v \in \text{Dom(Price)}, v \geq \alpha\}\) and
  \(A\)-values = \(\{v \in \text{Dom(Price)}, \alpha - \beta \leq v\}\).
  This corresponds to the t.m.f. defined as \((\alpha, k_2, 0, \beta)\).
- **\(I\)-values** = \(\{v \in \text{Dom(Price)}, \alpha \leq v \leq \beta\}\) and
  \(A\)-values = \(\{v \in \text{Dom(Price)}, \alpha - \lambda_1 \leq v \leq \beta + \lambda_2\}\).
  This corresponds to the t.m.f. defined as \((\alpha, \beta, \lambda_1, \lambda_2)\).

For some attributes, one may also assume available default predicates coming from common sense fuzzy partitions defined by an expert over the associated domains.

---

\(^5\) A t.m.f. is represented by a quadruplet \((A, B, a, b)\) where \([A, B]\) and \([A - a, B + b]\) respectively stand for the core and the support.
System-defined predicates. Examples of such predicates are: comparatively fast (denoted by $\#\text{fast}$), comparatively short (denoted by $\#\text{short}$). The definition of such predicates (i.e., their t.m.f.) is automatically computed by the system on the basis of a set of values (that we call the context [33]) returned by another query related to the initial one. For example, the context of the predicates present in a user query $Q^\delta_a(P, k)$ may be defined as the result of the query which returns the top-$k'$ shortest routes between $\delta_d$ and $\delta_a$ without taking into account the preferences. The idea is to define the t.m.f. of a predicate using the minimum and maximum values of the context [33], as illustrated in the example given further.

5.2. Overview of query evaluation

Let us first stress that finding the best path in a graph w.r.t. a set of preferences is known as a multi-objective shortest path problem [34,35], where preferences represent the set of objectives to optimize. Depending on the preference model used, the goal can then be either to compute the set of non-dominated routes (preference model based on Pareto order where the preferences are not commensurable [36]) or the best $k$ answers (as in the fuzzy-set-based preference model used here). In any case, evaluating an RPQ is an NP-hard problem [37,38]. In the following, for efficiency reasons, we define a processing strategy which returns a set of answers $S$ where each answer is associated with a score (which can be either a single degree or a pair of degrees). This set constitutes a reasonable approximation of the actual best answers. Let us emphasize that our processing strategy is somewhat similar to the solution advocated in the work done in [39]. Indeed, in this work the authors also make use of an approximation to provide the best $k$ routes (see Section 8).

In the system proposed, query evaluation is divided into two main phases:

Preprocessing phase: It aims at computing the best $k'$ routes from $\delta_d$ to $\delta_a$ w.r.t. some criterion of interest (for instance, the shortest routes may generally be used as a criterion), see further a discussion about this issue in Remark 3 of Section 5.3. This task is of the responsibility of the module named Path generator of our system as explained in Section 5.3.

Evaluation phase: It makes use of the set of routes provided by the first phase and, on the one hand, evaluates each route w.r.t. to each atomic user’s preference involved in the query at hand and, on the other hand, combines the preference degrees and returns the best $k$ results. This phase is performed by the module called evaluator of our system, see Fig. 2.

5.3. System architecture: the basic components

This section gives the basic architecture of a DBMS supporting RPQ’s, as well as the detail about the query processing strategy advocated. Let us consider a route planning query $Q^\delta_a(P, k)$ and assume that the user is interested in the $k$ best route plans (in the sense of $P$).

The system includes four main modules (see Fig. 2): a parser, an optimizer, a path generator and an evaluator whose roles are described hereafter.

Parser: The main task of the parser is to check whether the query is correctly specified and to resolve names and references. It takes as an input the query $Q^\delta_a(P, k)$ in its textual form and, if $Q^\delta_a(P, k)$ is valid, produces a parse tree that is an internal representation of the query.

Fig. 2. Components of the system.

Optimizer: It generates an efficient query plan for evaluating the query as well as a route generation plan. The latter aims at describing the optimal way to produce a relevant set of route candidates. Indeed, according to the user query different ways are possible to build this set.

Several pieces of information conveyed in the user route query can be leveraged so as to build an efficient path plan. Some of them make it possible to reduce the search space. Thus, a spatial constraint, like avoid some spatial entities, can be used to reduce the size of the road network from which the set of paths is generated, by removing all elements from the road network that the user wants to avoid (for example, avoid turnpikes). One can also observe that the type of the trip (i.e., urban or inter-urban) can also be used as a pruning criterion to guide the path generation and consider only the region (e.g., city) wherein the trip is done.

Path generator: Its purpose is to compute the set \( S' \) of the \( k' (> k) \) shortest routes from \( \delta_d \) to \( \delta_a \) in the graph representing the road network. It also extracts the value of each route and segment attribute on which there exists at least one condition in the preference part \( P \) of the query. Computing the \( k' \) shortest paths can be done using one of the classical algorithms from the literature. Here, we have used the one proposed in [40]. Of course, for this strategy to be acceptable and relevant, \( k' \) must be high enough to cover all of the realistic routes from \( \delta_d \) to \( \delta_a \).

Let \( Rte \) and \( Seg \) denote temporary instances of Route and Segment (introduced in Subsection 4.1) where the set of resulting routes and their segments are stored respectively.

Remark 3. We are aware that the criterion of shortest path is not always the most appropriate for the preprocessing phase and for really serving users’s expectations. For instance, for a scenic trip one could choose the most picturesque path. We believe that the shortest path criterion is the most interesting one in general, but any other additive criterion could be used instead.

Evaluator: It constitutes the core of the system proposed. It processes the user query \( Q_{\delta_d}^{\delta_a}(P, k) \) in three steps:

- Step 1: Setting. It consists in building the t.m.f. of the system-defined predicates involved in \( Q_{\delta_d}^{\delta_a}(P, k) \). The relations \( Rte \) and \( Seg \) generated by the path generator are used as the context of such predicates.
- Step 2: Evaluating. It evaluates each route \( r \in Rte \) w.r.t. each atomic user preference present in \( P \).
- Step 3: Aggregating and ranking. It consists in aggregating the preference degrees and ranking the answer tuples using one of the methods described in Section 2.2.2.

Remark 4. An alternative solution to the use of a dedicated query language would be to consider an architecture based on an API. We have chosen a language instead of an API because it offers, in our opinion, a good trade-off between expressivity (some spatial preferences would be difficult to express otherwise) and user-friendliness (RPQL is still relatively close to the natural language). In terms of results/performance, this would make no difference, however, since the internal machinery of the system would have to be the same (when it comes to preference processing and access to the database, in particular).

6. An illustrative example

Let us go back to the example given in introduction about a person who wants to go from Paris to Munich by a route that is comparatively short (flexible constraint) and comparatively not too expensive (flexible constraint), preferably with an affordable cost (flexible wish), and if possible passes about halfway near a restaurant accessible to disabled persons (flexible wish). Let us denote by \( Q \) this query (where \( P_W \) represents the wish part and \( P_C \) the constraint part). Query \( Q \) can be formulated as follows:

\[
Q = \{ r / \text{Route}(r) \wedge (r.\delta_d = 'Paris') \wedge (r.\delta_a = 'Munich') \wedge (P_C(r), P_W(r)), k \}
\]

where

\[
P_C(r) = \{ p_0, p_1 \} \text{ with } \begin{align*}
p_0 &= (r.\text{length is #short}) \quad \text{and} \\
p_1 &= (r.\text{cost is #inexpensive})
\end{align*}
\]
and

\[ P_W(r) = \{p_2, p_3\} \]

with

\[ p_2 = r.\text{cost is affordable}, \text{ and} \]

\[ p_3 = \exists o, \exists r (\text{Segment}(o) \land (o.\text{idRoute} = r.\text{idRoute}) \land \text{Restaurant}(rest) \land \text{near}(\text{rest.coord}, o.\text{coord}) \land r.\text{cost is affordable} \land \text{affordable} \land \left( \frac{\text{distance}(\text{rest.coord}, \text{getCoord('Paris')})}{r.\text{length}} \right) \text{ is halfway} \)

where \( p_0, p_1 \) and \( p_2 \) represent global preferences whereas \( p_3 \) is a compound spatial one. Three sets of spatial features are involved in the query as expressed in natural language: two in an explicit way (Route and Restaurant) and one in an implicit way (Segment): the preference \( \text{route passes near a restaurant} \) implies the existence of a restaurant near at least one segment of the route. In the above expression of \( p_3, \text{getCoord} \) is a function that returns coordinates of a given place provided as input and \( \text{distance} \) is a function that computes the Euclidean distance between two spatial objects.

Let us take a look at the RPQL and SQLf versions of the above query \( Q \). Fig. 3 (respectively Fig. 4) depicts the former (respectively the latter):

Now, the fuzzy predicates involved in the query \( Q \) are:

- \#short: It is a system-defined predicate. Its t.m.f. is \((0, l_{\text{min}}, 0, l_{\text{max}} - l_{\text{min}})\) where \( l_{\text{max}} \) and \( l_{\text{min}} \) represent respectively the maximum and the minimum length computed from the tuples in \( \text{Rte} \).
- \#inexpensive: It is also a system-defined predicate, its t.m.f is \((0, c_{\text{min}}, 0, c_{\text{max}} - c_{\text{min}})\) where \( c_{\text{max}} \) and \( c_{\text{min}} \) represent the maximum and the minimum cost found in relation \( \text{Rte} \).
- affordable: It is a user-defined predicate. Its t.m.f. writes: \((0, 15, 0, 15)\).

Table 3
Set of routes computed by the path generator.

<table>
<thead>
<tr>
<th>idRoute</th>
<th>δ_a</th>
<th>δ_d</th>
<th>Length</th>
<th>Rapidity</th>
<th>Duration</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_1</td>
<td>Paris</td>
<td>Munich</td>
<td>820</td>
<td>Null</td>
<td>Null</td>
<td>35</td>
</tr>
<tr>
<td>r_2</td>
<td>Paris</td>
<td>Munich</td>
<td>839.7</td>
<td>Null</td>
<td>Null</td>
<td>30</td>
</tr>
<tr>
<td>r_3</td>
<td>Paris</td>
<td>Munich</td>
<td>845.3</td>
<td>Null</td>
<td>Null</td>
<td>25</td>
</tr>
<tr>
<td>r_4</td>
<td>Paris</td>
<td>Munich</td>
<td>830.1</td>
<td>Null</td>
<td>Null</td>
<td>25</td>
</tr>
<tr>
<td>r_5</td>
<td>Paris</td>
<td>Munich</td>
<td>887</td>
<td>Null</td>
<td>Null</td>
<td>20</td>
</tr>
<tr>
<td>r_6</td>
<td>Paris</td>
<td>Munich</td>
<td>899</td>
<td>Null</td>
<td>Null</td>
<td>15</td>
</tr>
</tbody>
</table>

Fig. 5. The t.m.f. of affordable, #short, #inexpensive, near and halfway.

- near: It is also a user-defined predicate. Its semantics is assumed to be given by the following t.m.f.: (0, 5 km, 0, 5 km).
- halfway: It is also a user-defined predicate. Its t.m.f. is given by: (0.5, 0.5, 0.05, 0.05).

Now, assume that the path generator returns the set of routes Rte (with k' = 6) given in Table 3.

The t.m.f. associated with the predicates short and inexpensive, computed from Rte in the first step of the evaluation procedure are respectively: (0, 820, 0, 79), (0, 15, 0, 20). See Fig. 5.

Let us now say a few words about the evaluation of condition p_3 for a given route r from Rte, i.e., about the way the system could deal with it. As can be seen in the SQLf version of Q (see Fig. 4), several steps are needed:

- retrieve by means of a join query the restaurants that are somewhat near the route r (i.e., near a segment of r). Let μ_{near}(rest) be the degree associated with each tuple resulting from the Cartesian product segment × route × restaurant.
- among restaurants alongside the route r, retrieve those which are accessible for disabled person and whose location is in the support of the fuzzy predicate halfway. Let μ_{halfway}(rest) denote the satisfaction degree associated with each tuple rest. Let restaurant' be the fuzzy set obtained; one has:
  \[ μ_{restaurant'}(rest) = \min(μ_{halfway}(rest), μ_{near}(rest)) ; \]
- finally, \( μ_{p_3}(rest) = \max_{rest ∈ restaurant'} μ_{restaurant'}(rest) \), since max is the usual way to interpret \( ∃ \) in fuzzy logic [28].

By representing Q as a set of pairs \{(C_i, W_i), i = 1, ..., 3\}, where

- attribute A_1 = Length, C_1 = p_0 and W_1 = ∅,
- attribute A_2 = Cost, C_2 = p_1 and W_2 = p_2,
- attribute A_3 = Points of Interest, C_3 = True and W_3 = p_3.

we obtain the satisfaction degree of each \( r ∈ Rte \) w.r.t. each constraint and each wish involved in Q as shown in Table 4.

Table 4  
Results of steps 2 and 3.

<table>
<thead>
<tr>
<th>idRoute</th>
<th>$\mu_{C_1}(r)$</th>
<th>$\mu_{W_1}(r)$</th>
<th>$\mu_{C_2}(r)$</th>
<th>$\mu_{W_2}(r)$</th>
<th>$\mu_{C_3}(r)$</th>
<th>$\mu_{W_3}(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>$r_2$</td>
<td>0.75</td>
<td>0</td>
<td>0.25</td>
<td>0</td>
<td>1</td>
<td>0.51</td>
</tr>
<tr>
<td>$r_3$</td>
<td>0.68</td>
<td>0</td>
<td>0.5</td>
<td>0.33</td>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>$r_4$</td>
<td>0.87</td>
<td>0</td>
<td>0.5</td>
<td>0.33</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>$r_5$</td>
<td>0.15</td>
<td>0</td>
<td>0.75</td>
<td>0.66</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$r_6$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 5  
The pairs of scores (where the product operator is used as a t-norm to restore the consistency requirement).

<table>
<thead>
<tr>
<th>idRoute</th>
<th>$\mu_{PC}(r)$</th>
<th>$\mu_{PW}(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$r_2$</td>
<td>0.25</td>
<td>0.12</td>
</tr>
<tr>
<td>$r_3$</td>
<td>0.5</td>
<td>0.45</td>
</tr>
<tr>
<td>$r_4$</td>
<td>0.5</td>
<td>0.35</td>
</tr>
<tr>
<td>$r_5$</td>
<td>0.15</td>
<td>0.1</td>
</tr>
<tr>
<td>$r_6$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6  
evaluations w.r.t. constraints and wishes.

<table>
<thead>
<tr>
<th>Route $i$</th>
<th>Constraints $C_i$</th>
<th>Wishes $W_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>(1, 0, 1)</td>
<td>(0, 0, 2)</td>
</tr>
<tr>
<td>$r_2$</td>
<td>(0.75, 0.25, 1)</td>
<td>(0, 0.51)</td>
</tr>
<tr>
<td>$r_3$</td>
<td>(0.68, 0.5, 1)</td>
<td>(0.33, 0.9)</td>
</tr>
<tr>
<td>$r_4$</td>
<td>(0.87, 0.5, 1)</td>
<td>(0.33, 0.7)</td>
</tr>
<tr>
<td>$r_5$</td>
<td>(0.15, 0.75, 1)</td>
<td>(0.66, 0)</td>
</tr>
<tr>
<td>$r_6$</td>
<td>(0, 1, 1)</td>
<td>(0, 1, 0.6)</td>
</tr>
</tbody>
</table>

Query results ranking. As pointed out in Subsection 2.2.2, in case of the lexicographic order, the preference aggregation method used may influence the final ordering:

- **Min–max-based aggregation**: The lexicographic order is used with a priority given to the constraints, which means that the wishes are only used to break ties. The way the consistency requirement is enforced is illustrated in Table 5. The result obtained in this case is:

  $\{ (0.5, 0.45)/r_3, (0.5, 0.35)/r_4, (0.25, 0.12)/r_2 \}$.

- **Leximin–leximax-based aggregation**: The scores obtained by the six routes $r_i \ (i = 1 \ldots 6)$ are given in Table 6. Applying the leximin ranking on the $C_i$'s, one gets $r_4 \succ r_3 \succ r_2 \succ r_5 \succ r_1 \approx r_6$ (where $\succ$ stands for strictly preferred, and $\approx$ for equivalent). Then the leximax ranking on the $W_i$'s yields $r_4 \succ r_3 \succ r_2 \succ r_5 \succ r_6 \succ r_1$. The best $k = 3$ routes to be returned to the user are $\{r_4, r_3, r_2\}$. One can observe that in this ordering, $r_4$ is preferred to $r_3$ whereas $r_3$ is preferred to $r_4$ with the previous strategy. This is because $r_4$ better satisfies the actual constraints than $r_3$.

One may think that in general the value $k$ specified by the user will be $\leq 5$. As to $k'$, some experiments should be made in order to determine the best value of $k'$ (expressed as a multiple of $k$). It should be high enough to include all of the relevant routes, and small enough to permit a reasonable computational cost.
Table 7
Overall and detailed cost of an RPQ on a small (respectively medium) road network for \( k' = 10 \).

<table>
<thead>
<tr>
<th>Size of the network</th>
<th>Overall cost (ms)</th>
<th>Loading step (ms)</th>
<th>Computing step (ms)</th>
<th>Evaluation step (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(%)</td>
<td>(%)</td>
<td>(%)</td>
</tr>
<tr>
<td>10000</td>
<td>842.928</td>
<td>606.013</td>
<td>71.894</td>
<td>173.704</td>
</tr>
</tbody>
</table>

Table 8
Overall and detailed cost of an RPQ on a large road network for \( k' = 10 \).

<table>
<thead>
<tr>
<th>Size of the network</th>
<th>Overall cost (s)</th>
<th>Loading step (s)</th>
<th>Computing step (s)</th>
<th>Evaluation step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(%)</td>
<td>(%)</td>
<td>(%)</td>
</tr>
<tr>
<td>50000</td>
<td>58.147</td>
<td>55.501</td>
<td>95.449</td>
<td>2.037</td>
</tr>
<tr>
<td>100000</td>
<td>382.665</td>
<td>374.834</td>
<td>97.954</td>
<td>6.053</td>
</tr>
<tr>
<td>150000</td>
<td>837.244</td>
<td>821.073</td>
<td>98.069</td>
<td>12.618</td>
</tr>
<tr>
<td>200000</td>
<td>1620.175</td>
<td>1576.472</td>
<td>97.303</td>
<td>24.121</td>
</tr>
</tbody>
</table>

7. Experimental results

In this section, we present some experiments aimed at assessing the efficiency of the proposed approach. We have conducted these experiments using the PostgreSQL RDBMS running on a PC with Intel Core™ Duo CPU T7700 @ 2.40 GHz, 2024 MB of RAM, a processor cache of 4096 KB, and a hard disk Hitachi TravelstarTM 7K200 with 16 MB of cache. These experiments aim at evaluating:

- the overall cost of an RPQ as well as that of each main step of the processing,
- the impact of the parameter \( k' \) (i.e., the number of route candidates to generate) on that overall cost,
- the cost resulting from handling an explicit (respectively implicit) spatial preference.

Given the way an RPQ is evaluated, one can observe that the evaluation of a global preference can be done by a simple selection operation on the set of route candidates. As a consequence, the cost of this kind of preferences is negligible (< 1 ms).

7.1. Overall cost of an RPQ

In order to assess the cost of an RPQ, we have considered eight different road networks having different sizes that we have grouped into two families of four networks. The first one (see Table 7) corresponds to the road networks of a small and a medium city, having 500, 1000, 5000 or 10,000 road segments. The second family (see Table 8) gathers four road networks having respectively 50,000, 100,000, 150,000 and 200,000 road segments. They correspond to networks of large cities, e.g., Paris, or to the French national road network.

Remark 5. To minimize disk access, a bidirectional road segment is represented in our experimentation as one road segment having a Boolean attribute \( is\_bidirectional \) set to \( true \). By doing so, road networks considered can contain up to twice as many road segments as the numbers indicated above.

Table 7 (respectively Table 8) shows in the first column the total execution time (TET) of processing an RPQ in the context of four road networks with different sizes. As shown in Fig. 6(a), the overall cost of an RPQ on a small (respectively medium) road network is lower than one second, which is satisfactory. The form of the curve shows that this cost increases in terms of the size of the road network in an exponential way. This behaviour applies also in the
Fig. 6. Cost of an RPQ on a small (respectively medium) road network for $k' = 10$. (a) Overall cost according to the size of the road network. (b) Cost of each step according to the size of the road network.

Fig. 7. Cost of an RPQ on a large road network for $k' = 10$. (a) Cost of a route planning query on a large road network. (b) Cost of each step on a large road network.

case of a large network (see Fig. 7(a)). In particular, one can observe that for a network having 200,000 road segments, the cost reaches 27 min.

To analyse the increase of the cost of an RPQ in large road networks, a second series of measures has been performed. Such measures allow us to identify which step (respectively steps) is (respectively are) at the origin of this kind of behaviour. In Table 7 (respectively Table 8), the last three columns contain respectively the execution times and their percentages (w.r.t. TET) of the following steps: (i) Loading step which corresponds to loading and building the road network in main memory; (ii) computation step which consists in calculating the $k'$ route candidates; (iii) evaluation step which evaluates the preferences involved in the RPQ considered against the set of route candidates. As shown in Figs. 6(b) and 7(b), loading and building the road network is time consuming and the corresponding execution time represents more than 95% of TET of an RPQ.

To optimize the cost of this step, one way consists in representing the road network as a BLOB\(^6\) in the database. Combined with the cache system, this approach can considerably reduce the cost of this step. A second possible approach makes use of graph databases in order to avoid building the road network after its loading [41].

\(^6\) BLOB, Binary Large OBject, is a collection of binary data stored as a single entity in DBMS.
7.2. Impact of the parameter \( k' \)

To investigate the effect of the parameter \( k' \) on the execution time of an RPQ, we have conducted a series of measures to assess the time needed to generate the set of route candidates (i.e., to compute the \( k' \) shortest paths) with \( k' = \{10, 20, 30, 40, 50\} \) for a same RPQ. To this end, the road network having 100,000 road segments has been used.

The experimental results summarized in Table 9 show that the parameter \( k' \) has only a small impact on the TET of an RPQ. Indeed, generating \( k' + 10 \) paths represents only an extra cost of about 34 ms (+0.49%) w.r.t. the generation of \( k' \) paths. The quasi-linear form of the curve (see Fig. 8) representing the variation of the cost of the route generation step with respect to \( k' \), demonstrates the robustness of our approach w.r.t. the parameter \( k' \). Even if the difference between the sizes of the sets of routes generated is the same (i.e., 10), a significant variation in terms of additional cost on the execution time can be observed (see the last value of the fourth column of Table 9). This is mainly due to the complexity level of building an alternative path, which may be more important.

7.3. Evaluation of an explicit (respectively implicit) spatial preference

To study the cost of explicit and implicit spatial preferences, we have performed a set of experiments aimed at assessing the execution time needed to evaluate different numbers of spatial preferences. We have run such experiments on a road network having 100,000 road segments with \( k' = 50 \). Regarding implicit spatial preferences, we have taken care to formulate conditions on different spatial entity types (i.e., relations) in order to take into account the worst case.

The results depicted in Table 10 show that in general the evaluation of an implicit spatial preference is more expensive than that of an explicit preference, which is natural given the number of join operations involved in each type of spatial preferences. Indeed, only one join operation (between the relations Route and Segment) is required when dealing with an

---

Table 9

<table>
<thead>
<tr>
<th>( k' )</th>
<th>Time needed to compute ( k' ) shortest paths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(ms)</td>
</tr>
<tr>
<td>10</td>
<td>6153.487</td>
</tr>
<tr>
<td>20</td>
<td>6176.129</td>
</tr>
<tr>
<td>30</td>
<td>6209.969</td>
</tr>
<tr>
<td>40</td>
<td>6261.162</td>
</tr>
<tr>
<td>50</td>
<td>6274.830</td>
</tr>
<tr>
<td></td>
<td>(s)</td>
</tr>
<tr>
<td>10</td>
<td>6.153</td>
</tr>
<tr>
<td>20</td>
<td>6.176</td>
</tr>
<tr>
<td>30</td>
<td>6.210</td>
</tr>
<tr>
<td>40</td>
<td>6.261</td>
</tr>
<tr>
<td>50</td>
<td>6.275</td>
</tr>
<tr>
<td></td>
<td>Additional cost in %</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
</tr>
<tr>
<td>20</td>
<td>+0.37</td>
</tr>
<tr>
<td>30</td>
<td>+0.54</td>
</tr>
<tr>
<td>40</td>
<td>+0.82</td>
</tr>
<tr>
<td>50</td>
<td>+0.22</td>
</tr>
</tbody>
</table>

Fig. 8. Execution time of the generation of route candidate.
Table 10
Cost of an explicit (respectively implicit) spatial preference.

<table>
<thead>
<tr>
<th>Number of preference</th>
<th>Explicit preference</th>
<th>Implicit preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (ms)</td>
<td>Additional cost (%)</td>
</tr>
<tr>
<td>1</td>
<td>23.324</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>59.895</td>
<td>+61.06</td>
</tr>
<tr>
<td>3</td>
<td>79.096</td>
<td>+24.28</td>
</tr>
<tr>
<td>4</td>
<td>87.356</td>
<td>+9.46</td>
</tr>
<tr>
<td>5</td>
<td>93.919</td>
<td>+6.99</td>
</tr>
</tbody>
</table>

Fig. 9. Cost of an explicit (respectively implicit) spatial preference.

explicit spatial preference whereas at least two join operations (between Route, Segment and the spatial entity relations) are needed in the case of an implicit spatial preference.

One can observe that the execution time needed to evaluate a set of spatial preferences follows somewhat a logarithmic curve rather than a linear curve as one could have expected, see Fig. 9 and the additional cost column in Table 10. This behaviour is due to the database cache system that allows to reuse the results of some join operations. In particular, the join operation between the relations Route and Segment that is common to every spatial preference is a typical example.

8. Related work

In this section, we give a critical review of the main existing approaches to personalized route planning systems. On the other hand, we point out some works that are the most related to our proposal in the transportation domain where fuzzy sets are used as a modelling framework for the purposes of flexibility and uncertainty.

8.1. Personalizing route planners

In the last decade, a few propositions have been made to personalize route planners. Most of them try to take into account user preferences without necessarily representing them explicitly. This is due, mainly, to the difficulty of their modelling and elicitation [4].

Liu [42] proposes a route planning system which combines knowledge about the road network with case-based reasoning and brute-force search. He describes how geographical knowledge isolates the search for useful route segments to a local map region. The approach makes the strong assumption that users prefer routes that follow major roads, and the planning algorithm explicitly seeks out major roads to form the plan of a target route.

Rogers and Langley [1–3,43] propose a route planning system which can learn preferences from user feedback. During each interactive session, the user is asked to express his/her preferences among recommended routes. The feedback resulting from this interaction is used as the training data for a perceptron-style training algorithm. The authors assume
a fixed user preference model which only concern route length, driving time and turn angles. A numeric weight can be associated with the corresponding preferences.

Let us also mention the work by McGinty and Smyth [5,44,6] about a case-based route planning approach. The system they describe generates routes which reflect implicit preferences of individual users. The main aspect that distinguishes this system from that proposed by Rogers and Langley [1] is the fact that it does not assume a fixed preference model. Every user preferences are represented as a collection of previous route cases that the user considered satisfactory. Thus, new routes are generated by reusing and combining relevant sections of multiple cases.

Letchner et al. [4] developed a route planner, named TRIP, which produces route plans that more closely match the routes chosen by people who have extensive experience travelling within a region. To do so, it incorporates time-variant road speeds learned from large amounts of driver-collected GPS data. It also exploits a driver’s past GPS logs when responding to future route queries in order to provide routes which are more suited to the driver’s individual driving preferences.

Balke et al. [7,39,8] are the first to provide an approach where user preferences are explicitly represented. They propose a route planning system integrating user preferences over four characteristics of a route: length, traffic jams, road works and weather conditions. To aggregate the scores related to these preferences into an overall degree, a weighted function \( F \) is used. The weights are user-defined and express the importance of each of the route characteristics. The user may use a five-level linguistic scale for asserting the importance of a criterion, and the levels are automatically mapped onto numerical weights \( w_i \). In response to a route planning query, the top \( k \) results are delivered to the user. It is worth noticing that the set of routes returned to the user represents an approximation of the optimal routes (w.r.t. to the four criteria mentioned above). Indeed, only a set of routes where no route is longer than 1.5 times the shortest route (instead of all possible routes) is considered in the query evaluation step. Let us also mention a somewhat similar approach by Silva et al. [45]. This approach discusses some main routing algorithms and, in particular, presents the Coolest path algorithm which enables multi-criteria personalization based on travel distance, travel time, points of interest, and path simplicity (i.e., with less turns). Users may also set a level of importance for each of these criteria in a given scale of five levels.

More recently, Niaraki et al. [46] proposed an ontology-based personalized route planning system. A road segment ontology associated with (i.e., combined with) a user model and a context model are used to identify the most appropriate criteria for a given user. The user model (respectively context model) is defined as a set of parameters of the form of (key:value), organized in a hierarchical way. A multi-criteria decision making technique (i.e., analytic hierarchical process) is used to assign a weight to each criterion selected, combine them and build a cost function for the route planning algorithm.

The main aspect which distinguishes our approach from the works presented in [42,1,5,4] is that we choose—like Balke et al. [7,39,8]—to explicitly model user preferences. On the other hand, unlike Balke et al., we aim at taking into account a wide range of atomic user preferences in a bipolar way, and fuzzy set theory provides us with a rich set of connectives for combining such preferences, leading to a highly expressive query language.

### 8.2. Route choice models

Route choice constitutes one of the transportation issues most related to our problem. Roughly speaking, route choice models deal with the way a road network user chooses her/his route i.e., how (from a behavioural viewpoint) a user chooses/selects (respectively builds) her/his path [47].

Usually, route choice models have two components [48]: (i) the generation of a set of alternative routes; and (ii) the choice of a route among the alternative route set. This latter component relies on a utility function (that assigns a score to each alternative) or a set of fuzzy rules (e.g., \( \text{if speed is medium then route utility is good} \)) in order to identify the most likely choice [49–51].

Even if user preferences represent an important factor in the route choice process, few models consider them. Most of the existing propositions attempt to define a generic model and consider that individual characteristics and preferences are, generally, not available to the modeller. Because of this, an exact “true” cost function cannot be defined [10].

To the best of our knowledge, only Ridwan [9] proposes a route choice model based on fuzzy preference relations, where the user’s choice is built from successive selections performed at each intersection throughout the trip.
Clearly, route choice which aims to mimic the human behaviour to choose the best route w.r.t. a cost function by considering users’ experiences and some available pieces of information, is different from our proposal which rather aims at computing the best route(s) according to individual user preferences by applying efficient retrieval procedures.

9. Conclusion

In this paper, we have described the main features of a fuzzy-set-based approach to the modelling and handling of route planning queries involving complex user preferences. A typology of user preferences covering a variety of intuitive preferences has been defined. A very expressive route planning query language based on a fuzzy version of tuple relational calculus is described. In particular, and contrary to the existing approaches, the system we propose captures spatial preferences. On the other hand, in order to increase the user-friendliness of the querying interface, the foundations of an SQL-like language are outlined. To make the processing step more tractable and less time-consuming, an efficient route query evaluation technique is proposed.

A prototype system has been developed and a set of experiments have been carried out. According to these experiments, evaluating fuzzy preferences requires only a very small extra cost w.r.t. the overall cost of a simple RPQ. The experimentation that has been carried out also shows that loading and building the road network is the most time-consuming step. However, some optimization mechanisms could be used to reduce the search space and thus the execution time of this step. This line of research is one of our perspectives for future work. Further experimental measures should also concern the sensitiveness of the system to parameters, beside the value of $k'$. In particular, it would be worthy studying the impact of a change in the definition of the fuzzy sets involved in the query on the set of paths suggested by the planner. Answering this question on a quantitative basis would require defining a distance between paths, in order to assess divergence/similarity of proposed solutions. We also plan to conduct thorough user-centric experiments in order to demonstrate the relevance of the answers returned by the system, as well as the gain (in terms of user satisfaction) with respect to the use of crisp query specifications.

Finally, let us mention that one of the problems that may occur when processing an RPQ is that raised by empty answers, i.e., when no route somewhat fits the (flexible) constraints involved in the query. A potential reason that could lead to this situation is the small size of $S'$. This means that no route among the set $S'$ of the shortest $k'$ routes (provided by the Path Generator) fits the requirements involved in the user’s route query $Q^{h_p}(P, k)$. One can then augment the set $S'$ by additional routes between the locations $\delta_d$ to $\delta_a$, and then evaluate them w.r.t. the user’s constraints. However, this solution which is simple and easy to carry out, is interesting only if searching for additional routes does not require the algorithm for shortest path computation to be run again from scratch. It may happen that even when augmenting the set $S'$, a route query still results in an empty answer. In this case, a solution is to relax some or all preferences expressing flexible constraints (using, for instance, the query relaxation method proposed in [52]).

References


