UNIVERSITÄT AUGSBURG

A Context-Aware and Preference-Driven Vacation Planner for Tourism Regions

Alfons Huhn[a], Patrick Roocks[a],
Werner Kießling[a], and Martin Soutschek[b]

[a]DBIS, Faculty of Applied Informatics
University of Augsburg, Germany
[b]ALPSTEIN Tourismus GmbH & Co. KG
Immenstadt, Germany

Report 2015-04 December 2015

INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

Contents

Abstract

Taking a Preference SQL approach, a context-aware vacation planner for on-site activities is proposed to automatically generate vacation plans based on user preferences and situational aspects. Using different levels of abstraction, the result of the corresponding preference queries is always optimal and the result size is minimal. It consists of stereotype-specific and context-aware activities which are combined to create daily or even multi-day plans of activities. The correctness, completeness and optimality are assured by a preference calculus of strict partial orders. User preferences are initially collected and defined by a feedback questionnaire. The application is modelled by adequate preference compositions and the Preference SQL runtime system efficiently evaluates the resulting preference queries. The prototype proves that soft run-time requirements are met. Initial tests with real data from the industry-leading outdooractive platform indicate that the database-driven preference technology can successfully be employed to provide added value for vacation planning.

Keywords: vacation planner; timetable; preference modelling; context awareness; optimality; best matches only semantics.

1 Introduction

Up to now, tourists inform about interesting vacation destinations by travel agencies, social relations, and online information. If the destination is chosen, on-site activities are influenced by the tourist information, landlords, and indigenous people. Their experience assigns roles, associated activities and interests to the tourists. These insiders give hints or suggestions, which seem appropriate. Often, even bad weather alternatives are offered to increase tourists' customer satisfaction by taking into account the actual context.

Anyhow the behaviour of clients is changing substituting analogue information by digital. These native digitals expect 24/7 service everywhere. Typical up-to-date applications like www.tripit.com stick fixed points like airports or hotels together with driving directions to generate a linear vacation itinerary. Döring and Preisinger (2008) enhanced the underlying concept of *dynamic packaging* by a personalised search model based on preferences. At first, www.inspirock.com focuses on touristic points of interests which are connected by different modes of transportation and accommodation facilities afterwards. Obeying the precept of dichotomy, tourists often stay or even have to stay in a region and react to opportunities depending on circumstances changing every day. Thus, stereotype-specific and context-aware plans for on-site activities or attractions are desirable. A generic architecture implemented as a prototype is proposed which relies on databases and preferences generating best-suitable plans of on-site activities.

Planning of activities is a well-known branch of Artificial Intelligence as described e.g. by Ghallab, Nau, and Traverso (2004). Optimizing tasks, duty rosters, timetables, and schedules are assignment problems further explained by Burkard, Dell'Amico, and Martello (2012). However, preferences defined as strict partial orders are rarely supported by these approaches. Our prototype relies on the widespread and accepted use of SQL and applies both technologies for tourism applications.

2 Basics of Preference-based Modelling and Querying

The theory of preferences was first introduced by Kießling (2002) and Chomicki (2003).

- **Definition Preference P = (A, $<_P$):**

Given a set A of attribute names, a preference P is a strict partial order P = (A, $<_P$), where $<_P \subseteq \mathrm{dom}(A) \times \mathrm{dom}(A)$.

This initial proposal of Kießling (2002) was implemented by Kießling and Köstler (2002) as *Preference SQL*, a complete run-time system. Preference SQL is still further developed by Kießling et al. (2011). All base preferences are derived from a base preference called *SCORE(anyAttribute).* This preference denotes that all objects whose score value with regard to *anyAttribute* is smaller than others are considered to be better than those.

The use case of section 4 relies on the following specialisations of *SCORE*:

- *LESS THAN(a, v)* or *AROUND(a, v)* using an asymmetric or symmetric distance between the desired value *v* and others of an attribute *a* to be minimised as well as,
- *HIGHEST(a)* or *LOWEST(a)* for maximising or minimising the values of an attribute *a*.

These fundamental preferences as well as any resulting complex preferences (p_i) are combined by the following complex preferences:

- The clause p_i *AND* p_j is stating that both preferences have the equal importance.
- The clause p_i *PRIOR TO* p_j is stating that preference p_i is more important than p_j.

Semantical and syntactical details can be found in the literature mentioned above. The most import feature of preference queries underlying this sound and complete theory of preferences as strict partial orders is the property of having *best matches only* (BMO) that guarantees the correctness and the optimality of the result set of preference queries.

This classical preference theory has to be reinterpreted in the context of hierarchical data structures stemming from the requirements of a planning application. Starting from Aristotle's quotation "The whole is more than the sum of its parts" the relation Aristotle looks like:

Table 1. Hierarchical data structure

Whole_name	Part_name	Part_attribute
…	…	…
a_b_c	a	1
a_b_c	b	2
a_b_c	c	3
…	…	…

This relational representation corresponds to a hierarchical data structure (tree) as shown in fig. 1:

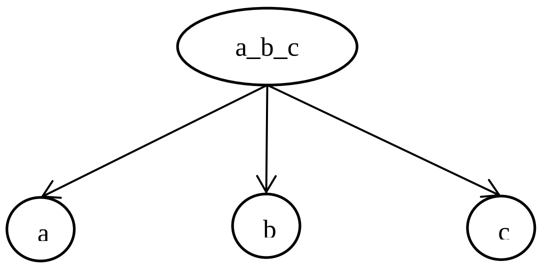


Fig. 1. Part-whole hierarchy

Aristotle's gem may be expressed by:

```
SELECT      whole_name,
            -- A(.)
            SUM(part_attribute)  +  COUNT(part_attribute) - 1
            AS whole_attribute
FROM        Aristotle
GROUP BY    whole_name
```

This query focuses on the higher abstraction level "whole" by taking into account the lower abstraction level "part". Clearly, the result of this query may be used as a subquery of a structural identical query treating the just created wholes as parts of further wholes, etc.

Having a fixed count of iteration this process of self-similarity may be formulated as SQL2-query of iterated subqueries. But the resulting query is complex and the count of iteration is often unknown, however some conditions of termination are a priori known. Thus, the available recursion of SQL3 is the white knight.

Also the relation "Aristotle" can equivalently transformed to a non-standard relation using set-valued attributes like:

Table 2. Hierarchical data with set-valued attributes in SQL3

Whole_name	**Whole_attribute**	**Part_names**	**Part_attributes**
…	…	…	…
a_b_c	8	{a, b, c}	{1, 2, 3}
…	…	…	…

The attributes "Part_names" and "Part_attributes" are set-valued. No order exists among the set members. However, the enumeration of the set implicitly defines a total order on the members of the set which can be interpreted as sequence. Such a stack of activities is represented in condensed form. Both interpretations of partial order or total order are useful in the context of planning, as shown later on.

The reinterpretation of the Kießling's preference definition is intended by following extension.

- **Definition Preference P' = (A, $<_{P'}$):**

Given a *multiset* A of attribute names, a preference P' is a strict partial order defined as

$$P' = (\mathcal{P}(A), <_{P'}),$$

where $<_{P'} \subseteq F(\mathcal{P}(A)) \times F(\mathcal{P}(A))$. $\mathcal{P}(A)$ is the power set of the multiset A, and $F(\mathcal{P}(A))$ is a function whose input parameters are multiset members and the output is of type FLOAT. The involved operators have to be commutative. A(.) acts as an agglomerate like the SQL-aggregation on each member of the power set instance.

Simplifying the multiplicity function for multisets to the set indicator function of normal sets and constraining the power set $\mathcal{P}(A)$ to sets of size 1, each set member is interpreted as an attribute value and the agglomerate A(.) is defined as identity. This construction yields the original definition of the preference P = (A, $<_P$). Zhang and Chomicki already introduced preferences over sets in 2011. But they restricted their work to sets of fixed cardinality. We insist on the whole power set of a multiset.

Aristotle's examples above defines the agglomerate A(.) as sum of "part_attribute" of all set members of $\mathcal{P}(A)$ plus the count of set members of $\mathcal{P}(A)$ minus 1, where A is "part_attribute".

The power set $\mathcal{P}(A)$ urges the cooperation of SQL3 and Preference SQL defined as an extension of SQL2 as independent components to achieve following goals:

- Handling of hierarchical / recursive data structures by recursion of SQL3
- Transformation of set-valued types of SQL3 to atomic types of SQL2 by SQL3

Therefore a smart integration exists, since new preference constructors are not necessary. As first step, the interleafing of Preference SQL and SQL3 can be instrumented by combining SQL3 and Preference SQL in one script materialising the results of SQL3.

3 An Architecture of Context-aware and Preference-based Planners

In this section, a generic preference-driven architecture is proposed from a questionnaire as starting point to preference modelling and preference evaluation finally.

3.1 Questionnaire as Starting Point

A starting point of context-aware and preference-driven planners consists of:

- Defining a controlled vocabulary as basics.
- Using this vocabulary to define a questionnaire by getting feedback from users.

Controlled vocabularies base on terms having semantic relations among each other which are described by thesauri (ANSI/NISO Z39.19 [ANSI], 2005). Thus application-specific languages are defined on a solid base. Having technical systems implementing an application, the same model can be used by replacing thesauri with taxonomies or even ontologies defining classes, individuals, attributes, and relations. In any case, the *closed world assumption* (CWA) holds relying on a finite description of the world. The CWA states that only facts assigned as true are true. The CWA excludes the emergence of new terms or the use of synonyms known by clients of an application but unknown to the controlled vocabulary. Thus the *open world assumption* is preferable in dynamically changing applications. Having a database-oriented architecture NULL-values are paving the way to tackle this task as described by Gottlob and Zicari (1988). Preferences can also handle NULLs as demonstrated by Endres, Roocks, Wenzel, Huhn, and Kießling (2012). Thus the proposed generic architecture is independent of the closed or open world assumption.

After having defined the controlled vocabulary, it is convenient to gather the feedback of users through a questionnaire. The questionnaire approach can be applied as a web-based or traditional interview, in a stand-alone manner or directed by an interviewer. Interviewed people answer questions formulated in terms of the controlled vocabulary by evoking their rating. The procedure generates a *Likert scale* as defined by Likert (1932). The used terms, as part of the controlled vocabulary, have to be mapped to the attributes of relations relying on a database-oriented architecture.

Since the Likert terms are a subset of the controlled vocabulary and people do not necessarily answer all questions of a questionnaire, the problem of how to handle incomplete feedback arises. Based on the assumption that most terms can be arranged by taxonomies fulfilling the Liskov substitution principle (LSP) introduced by Liskov and Wing (2001), the rating of a term is valid until the rating of a more restricted sub-term got a different rating as shown in fig. 2:

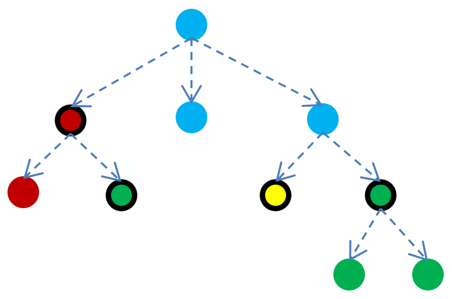


Fig. 2. Incomplete feedback and the consequences of LSP

Any taxonomy of terms is represented by nodes and blue-dotted arrows in a tree as shown in fig. 2. Nodes with black outlines correspond to the feedback of a user. The colour of a node is representing a possible class of feedback. If the root of the tree has not been rated by a user, a default rating has to be assumed (e.g. NULL shown by blue).

The questionnaire may be used as an

- Input for statistics after having collected a lot of interviews or as a
- Behaviour model of stereotypes.

The behaviour model of a stereotype as illustrated in fig. 2 is defined explicitly by a domain expert or through an analysis of user feedback from multiple interviews.

Now, preferences come into play by using the rated taxonomy of fig. 2 in this way:

- All items with the same rating constitute a layer.

This procedure directly results in a *LAYERED*$_l$-preference as defined by Kießling (2002).

Assuming five layers, the transformation of the questionnaire may generate the following *LAYERED*$_5$-preferences:

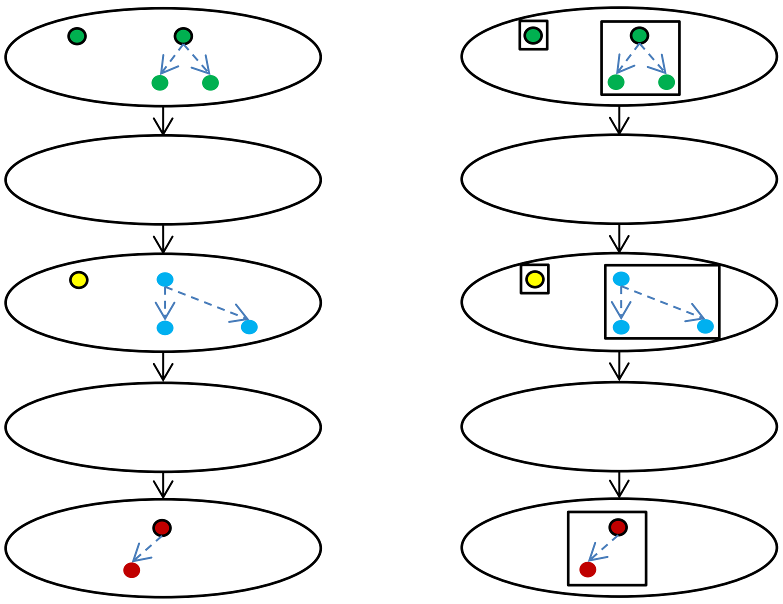


Fig. 3. Construction of two stereotype-specific LAYERED$_5$-preferences

Each layer of each preference consists only of terms. Trees inside a layer just visualise the provenience. Four empty layers exist in fig. 3. NULL values in blue are handled e.g. as yellow values. According to Kießling (2005) all items of the same layer are substitutable. Users are rating all of them as equally good known as *regular*

SV-semantics stating *SV* as acronym for *substitutable values* as shown by the left preference.

Looking at the provenience, the right preference is evident by introducing finer SV-classes shown as black rectangles inside layers. Layers and also SV-classes consist of disjoint sets of items. SV-classes are named by the root of the corresponding subtree. *Specific SV-semantics* is shown by the right preference. Now, only members of the same SV-class are substitutable, but members of different SV-classes are not substitutable, even if they belong to the same layer as introduced by Kießling (2005). This construction is favourable having very different semantic concepts in the original taxonomy. Thus, they are not substitutable and the *semantic diversity* is obtained by composing a complex preference as in case of planning. In summary, specific SV-semantics widens coarse-grained classes to fine grained classes.

Having taxonomies, the importance of the involved taxonomies is reflected by preferences having

- Equal importance or,
- Having more, or having less importance respectively.

Thus, the complex preference constructors of Kießling (2002) can model user or system preferences, which are influenced by the stereotypes and the context as shown in the next section.

3.2 Context-aware Top-k Lists as Basic Set

Already Roocks, Endres, Mandl, and Kießling (2012) have demonstrated a model to construct context-aware preferences by the prioritisation of context-specific preferences assigned to each basic preference.

The planning system needs a basic set of at least k elements to generate plans. Thus, the system relies on the *top-k operator* of the Preference SQL system, guaranteeing that the result set has just size *k*.

3.3 Combinations of Elements of a Basic Set as State Space

From the above mentioned construction, each element of any context-aware result set is optimal with regard to the underlying preference. Next, combinations of these optimal elements are generated.

For planning each combination acts like a group subsuming its parts as a whole and having an appropriate agglomerate function A(.). A combination is a subset of a basic set aka activities. The order of execution inside the combination is out of interest. Each node has an assessment by the agglomerate. The state space is characterized by properties like:

- Repetition of elements of the basic set is forbidden, allowed, or restricted.
- Hard restrictions expressed on state space variables limit the state space size.
- Soft restrictions expressed on state space variables may model user or system preferences.
- The result of an agglomerate function is subject to a stereotype-specific preference to get appropriate combinations fulfilling the BMO-property.

- The state space grows exponentially.

According to Chapman (1987) *nonlinear planning* is sufficient. Thus, a plan is a partial order of operators in which an operator corresponds to an element of a context-aware top-k list mentioned in section 3.2. Hence, the planning result is always a non-linear plan which is less prone to re-planning due to changes of context.

Following the principle of *hierarchical planning* as described by Wilkins (1986) a plan refinement process is proposed by using the output of a lower abstraction level as input of a higher abstraction level iteratively until a plan is created.

After the requirements of the planning task are apparent, the projection (part_name, part_attribute) of the excerpt in table 1 is the basic set in order to generate its power set as state space (see also its representation as set-valued relation shown in table 2), depicted in fig. 4.

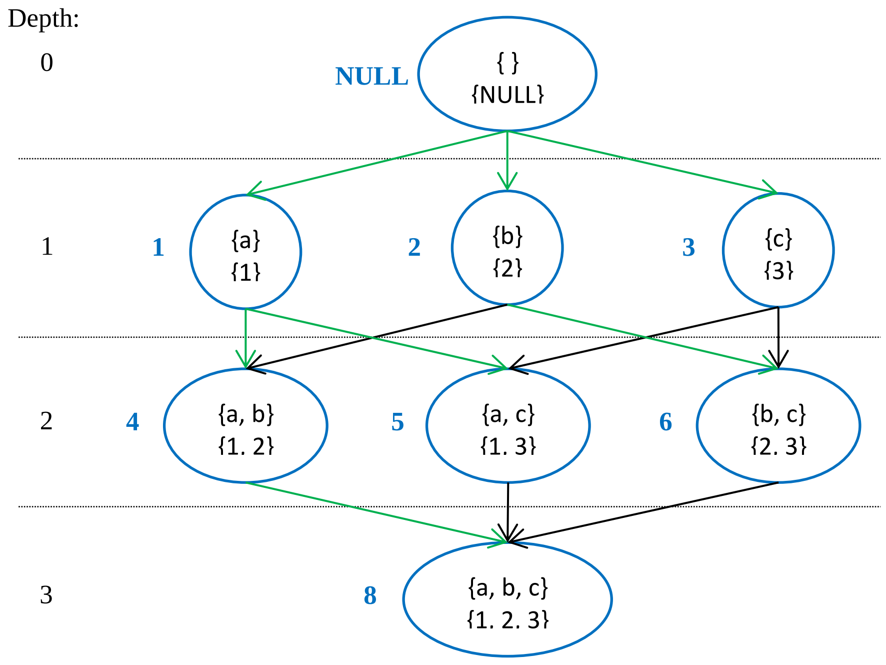


Fig. 4. Power set as a state space of the basic set {a, b, c}

The nodes are the elements of the power set. The edges reflect all possibilities to generate the nodes of layer i with the help of the nodes of layer i- 1. The structure is a lattice. The root of the lattice has per default NULL as value for the considered attribute, since "any attribute of [doing] nothing" is indetermined. Each application has to decide whether this state makes sense. Remember the NULL-handling of Preference SQL (Endres 2012) which maps NULL at runtime to any level or distance without any change of the underlying model.

The blue numbers at the left of each node show the return value of A(.) having the numbers inside of each node as input. The states and their associated agglomerated attribute in blue are settled at a higher abstraction level than their components in black reflecting the structure of table 1. Since the states can be constructed in different ways by following all paths from the root to the interesting state, a spanning tree shown in green avoids redundant work while generating the states. Having a database perspective, only the states and their agglomerated attributes are of interest modelling aspects at the whole-level. Having a hierarchical / recursive data structure, recursion of SQL3 generates the desired data structures. Having no a priori knowledge which states are illegal, the whole state space – a complete lattice – is generated. However, the use of a spanning tree involves an artificial dependency since children depend on the existence of their unique father and furthermore on the existence of all their ancestors. When heuristics (e.g. preferences) come into play at each depth during the generation process, a spanning tree algorithm is inappropriate, when the heuristics cannot guarantee monotony. A bad father swept off by the heuristics kills also its better descendants.

If only states are regarded the independence of its components is implicitly an assumption. No order is defined. The components behave like a set. Regarding the agglomerated attribute at the whole-level the function A(.) has to be independent of the order of the input parameters. No history or a stack trace comes into play. This feature matches nicely the concept of partial plans having no order (dependency) defined on its activities. Details like the runtime of generating the power set of activities as state space are found in the appendix at the section A.7.

Following the principle of self-similarity or hierarchical planning as described by Wilkins (1986), multi-day plans obey the "part-whole" pattern as shown in fig. 5:

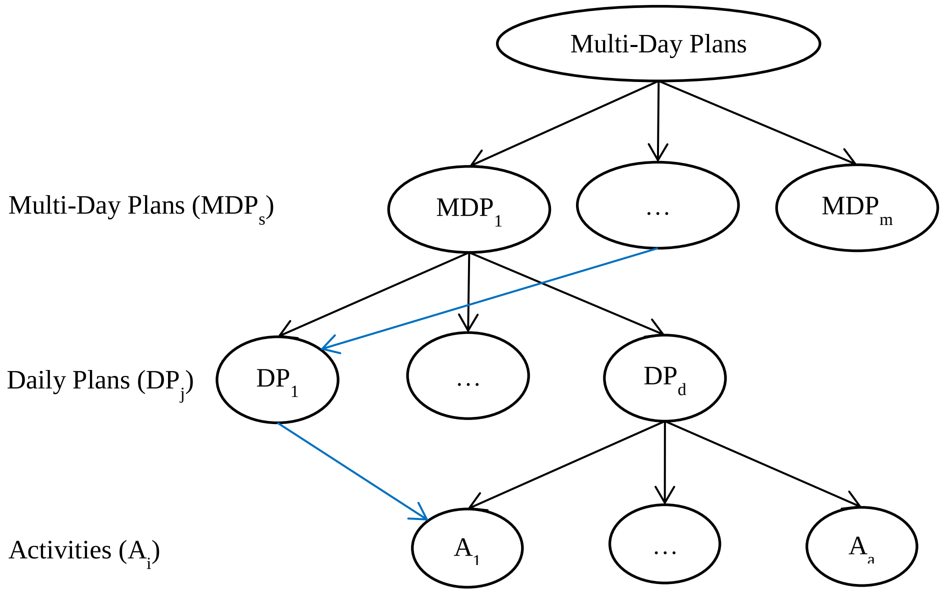


Fig. 5. Hierarchical planning of multi-day plans

Elements of a lower abstraction level may be reused several times by elements of a higher abstraction level indicated by blue arrows issuing a one-to-many relationship. The reuse of components may be interpreted as "conflict" or "shared resource". In the use case of vacation planning, the pragmatics is boredom. A refinement of the planning architecture is depicted in fig. 7 by reusing the concept of state spaces at different abstraction levels following the part-whole pattern. Also a preference evaluation takes place at each abstraction level to receive just the best matching objects.

Regarding the dichotomy of nodes and edges, another interpretation of fig. 4 is possible. The set of all paths starting from the root { } as starting point s and having different length is generated. Now a path p1: s->a->a_c and a path p2: s->c->a_c have the same endpoint, which means they are equivalent in a node-oriented interpretation. In an edge-oriented interpretation paths can be handled different with respect to the agglomerated attribute, which is now associated to a path. The value may depend on the predecessor.

An equivalent transformation of the state space of fig. 4 into a search space is shown in fig. 6:

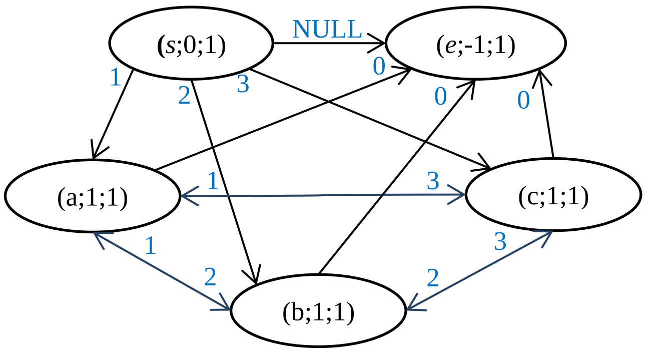


Fig. 6. Equivalent search space of the state space of fig. 4

Each node consists of the triple (activity; nodeAttribute; reuse). Reuse indicates the maximal count of visiting a node. To each directed edge the attribute "edgeAttribute" is assigned. For simplicity, blue bidirectional edges use two numbers: one for each incoming edge. The reuse is constraint to one. Therefore each path contains every node only once. All paths are generated from the starting point *s* to the endpoint *e*. Since the basic set {a, b, c} has three members, the path length ranges from 1 to 4. Each path now encodes the order of generation resulting in a total plan of activities. The count of total plans vastly exceeds the count of partial plans.

Following a path, the function A(.) is defined by:

- SUM(edgeAttribute) + SUM(nodeAttribute).

E.g. the path s->a->b->c->*e* has the agglomerate A(.)= 1+ 1+ 2+ 1+ 3+ 1+ 0+ (-1) = 8 which corresponds to the state {a, b, c} with A(.)= 8 in fig. 4. The "empty" activity encoded as path *s*->*e* has the agglomerate A(.)= NULL -1 = NULL which is handled by the NULL handling of preferences in Endres (2012).

Details like the runtime of generating the power set of a multiset of activities are found in the appendix at the section A.8.

As a special case of fig. 6, predefined graphs can be interpreted as an abstraction of maps. Having specified a starting point s and an endpoint e, the recursion takes care of the predefined transition relation of the graph to generate all paths from *s* to *e* obeying some conditions and preferences. Suitable routes are the outcome of this planning process.

3.4 Preference-based Evaluation of a State Space

As proof of concept the relation "top100" was created as set of (id, attribute)-tuples ranging from 1 to 100. By creating a view, any top-k list with k<= 100 can be created as the basic set of experiments by creating different state spaces and by evaluating preferences on these state spaces.

First, we assume no repetition of elements. Thus the power set of the basic set having *k* elements is the desired state space.

Table 3. Execution time [s] of generating the power set based on *k* elements

k	Execution time [s]
5	0,138
10	0,144
15	1,057
20	28,567

Regarding table 3, an Intel i7-3540M notebook with 3.0 GHz and 16 GB was used. Obviously, smaller values of *k* fulfil the soft time constraints to generate the power set.

Second, we release the hard restriction of having no repetitions. A multiset is a representation of the state space whereupon the count of an element states the maximum of repetitions of this element. A power set is clearly a specialization of a multiset. The execution time of smaller multisets is similar to that of power sets shown in table 3.

Preference queries are executed on these state spaces whereupon state space variables may be involved and interpreted by the application as shown in section 4.

4 Use case - Vacation Planner from Backend to Frontend

Having a database with relations like tours, points of interest (POIs), offers, and taxonomies of activities or POIs, the vacation planner constructs partial plans as suggestions of activities and delivers the result through a user-interface (UI) as illustrated in fig. 7. The resulting timetable of the vacation planner considers the stereotype assigned to the user, the region, and the situational context of users with the weather situation clearly having the most important impact.

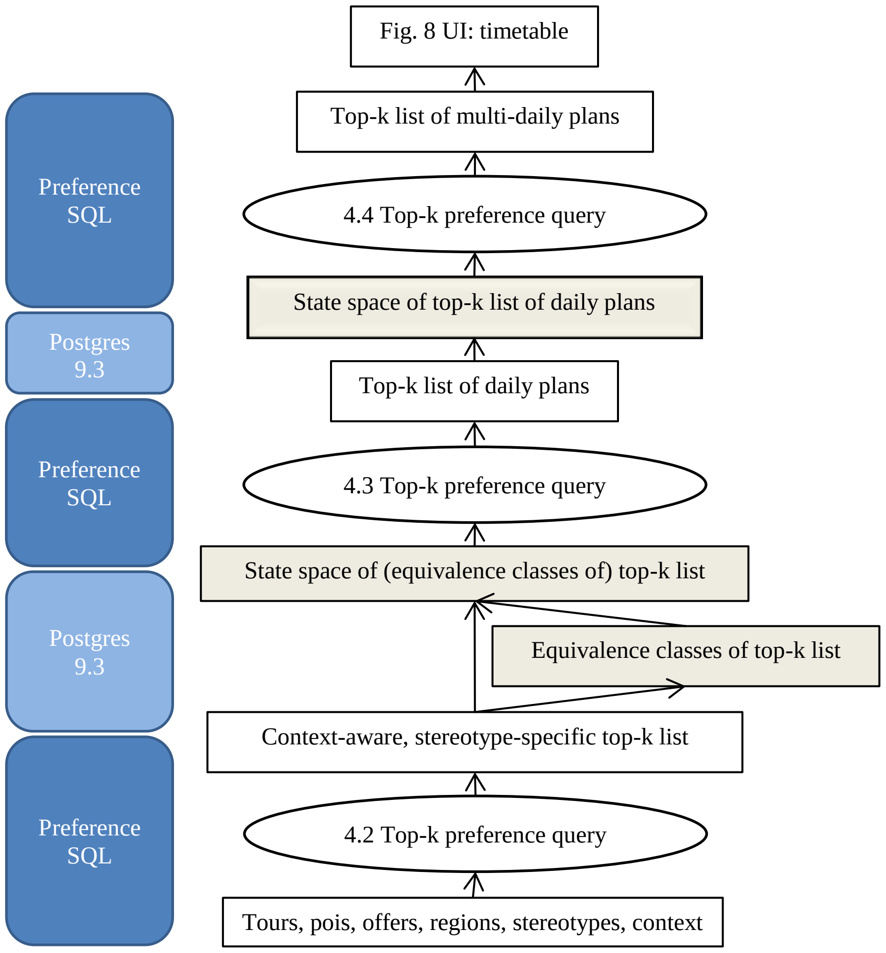


Fig. 8 UI: timetable
Top-k list of multi-daily plans
4.4 Top-k preference query
State space of top-k list of daily plans
Top-k list of daily plans
4.3 Top-k preference query
State space of (equivalence classes of) top-k list
Equivalence classes of top-k list
Context-aware, stereotype-specific top-k list
4.2 Top-k preference query
Tours, pois, offers, regions, stereotypes, context
Preference SQL
Postgres 9.3
Preference SQL
Postgres 9.3
Preference SQL

Fig. 7. Software architecture of a vacation planner from backend to frontend

Top-k preference queries (oval) are used to get the k-best results based on the input relations and the context-aware and stereotype-specific preferences. The preference queries are refined in the following section 4.2, 4.3, and 4.3. Some transformations (grey) like the building of equivalence classes to reduce the state space, as well as the following generation of state spaces, are necessary to construct input relations for successive preference queries. Three stages corresponding to *Hierarchical Planning* (Wilkins 1986) are pipelined to generate multi-day plans of which one is delivered to the customer per UI.

4.1 Questionnaire

Analysing the database of outdooractive.com, the richness of its semantics is expressed by following numbers of concepts (see table 4). The value of 1 for *depth of taxonomy* indicates that no hierarchy of concepts exists.

Table 4. Size and kind of Controlled Vocabulary

Semantics	Count of concepts	Depth of Taxonomy
POIs	654	4
Activities	68	4
Offers	112	3
Relations	162	1
Annotations	194	1

Next, concepts and stereotypes have to be matched by bias and aversion i.e. preferences. Influenced by the *stereotypes* described by Gibson and Yiannakis (2002), each semantic concept is rather explicitly denoted by numbers between 1 and 5 according to our model-driven approach and using a Likert scale.

Context is modelled by

- Weather (good / bad) and
- Season (summer / winter).

The weather forecast obtained through the outdooractive.com API is valid for three days and structured in time intervals of three hours. Changes of weather require and trigger the re-planning of vacation plans according to the new conditions. At the moment of planning, the actual forecast is used and the weather conditions serve as guards of integrity to be checked.

The *region* of any POI, activity, and offer is delivered by some of the above mentioned semantic relations.

Stereotype, region, and context act as keys to retrieve the context-aware and stereotype-specific top-k lists and are generated by a preference composition as shown next.

4.2 Preference Composition

Having an application-specific view, those tours, attractions, and offers are preferable, which fulfil the following properties best:

- Since each stereotype has rated the available activities and they are convenient with respect to season and weather, only those activities were preferred which are better rated by its associated questionnaire.
- Activities are as important as POIs.
- Following the same principle as for activities, the hierarchy of POIs is transformed to a hierarchy of layers. Thus, those activities are preferred, which guarantee more attractive POIs than others.
- The knowledge of the provider is encoded in a ranking for each touristic object. Its experience is as important as activities or POIs.

The above specification creates the following, syntactically reduced, preference expression:

```
PREFERRING
        activity LAYERED ($best, $good, $equal, $bad, $worst)       AND
        ((count_of_best_pois HIGHEST      PRIOR TO
          count_of_good_pois HIGHEST      PRIOR TO
          count_of_equal_pois HIGHEST)    PRIOR TO
         (count_of_worst_pois LOWEST      PRIOR TO
          count_of_bad_pois LOWEST))                                 AND
        ranking HIGHEST
```

The result set consists of objects of the types “tours”, “pois”, and “offers” and it contains the prerequisites as keys to deliver the context-aware and stereotype-specific best objects as top-k lists. The terms “$best”, … , “$worst” are Preference SQL-specific macros which substitute the macro names by sets of concepts resulting from adequate queries to get just those concepts of the questionnaire with a specific rating. The middle term of the above expression in brackets is a priority chain and generates an ordered result list. The resulting specific top-k lists are used as basic set to generate daily plans.

4.3 Daily Plan as a Parametric Component of Total Duration

After generating a context-aware and stereotype-specific relation per region, the BMO-property of preferences guarantees the optimality of each tuple. Thus, the task consists of finding appropriate combinations of tuples to fill the time slots of a daily plan.

The attribute “duration” of type “float” is evaluated for all activities such as tours, POIs, and offers - i.e. activities. The combination of activities requires an agglomerate to obtain a *total duration*. Since transfer times were excluded to reduce the complexity of the initial prototype, the implemented agglomerate generally assumes 1 h as transport time between the activities that are suggested for. Clearly, each single activity may be used to generate the state space of its power set but often the basic set is already too large. Thus, equivalence classes of activities having *n* hours of duration are used to avoid the complexity of the state space. Since the equivalence classes may contain several elements, a multiset is used as state space. This discretization nicely corresponds to time slots of a timetable shown at UI. A finer discretization generates more equivalence classes. The present prototype hints that a lower bound of ½ hour is computationally manageable.

With regard to different weather conditions during a day, the *total duration* is handled as a parameter. Like a target function to be maximised, we formulate the specification that the total duration should be *para_duration*. It is always more agreeable if the duration is shorter than longer. The according preference looks like:

```
PREFERRING
        (total_duration LESS THAN $para_duration                PRIOR TO
         total_duration AROUND $para_duration) AS duration_preference
```

These heuristics yields the desired behaviour of "good" daily plans. The name of the preference is *duration_preference*. According to the BMO-property tourists get perfect plans just having a total duration of *para_duration* or, alternatively, shorter ones. Since Preference SQL contains the WHERE-clause of SQL, the total duration is restricted by 24 hours reducing the size of the state space.

The resulting plans are *nonlinear plans* (Chapman 1987). There is no order of execution inside a combination of activities. Also the starting time is out of the scope having stereotypes and no individuals. The arrangement of activities is left to users at UI. At the end, the involved equivalence classes and their count of use are mapped to the atomic activities.

The above preference may be enriched for the stereotype "young & fun" by adding a bias of diversity having the same importance:

```
PREFERRING
        duration_preference                                     AND
        count_activity HIGHEST
```

In summary, the algorithm generates context-aware and stereospecific daily plans of a region and presents them as a vacation planner timetable for tourists.

4.4 Multi-Day Plan as a Parametric Component by Number of Days

As shown in the previous section, the preference-driven approach generates a set of optimal daily plans with respect to the total duration. The size of this set may be controlled by a TOP-*k* clause. The parameter of the next planning stage is the number *n* of days. Regarding the reliability of weather forecasts, the number of days is set to 3 per default, without loss of generality.

The repetition of the same activity is termed as a *conflict* to avoid that tourists are bored by a repeating suggestion. At this abstraction level, the state space offers the concepts of:

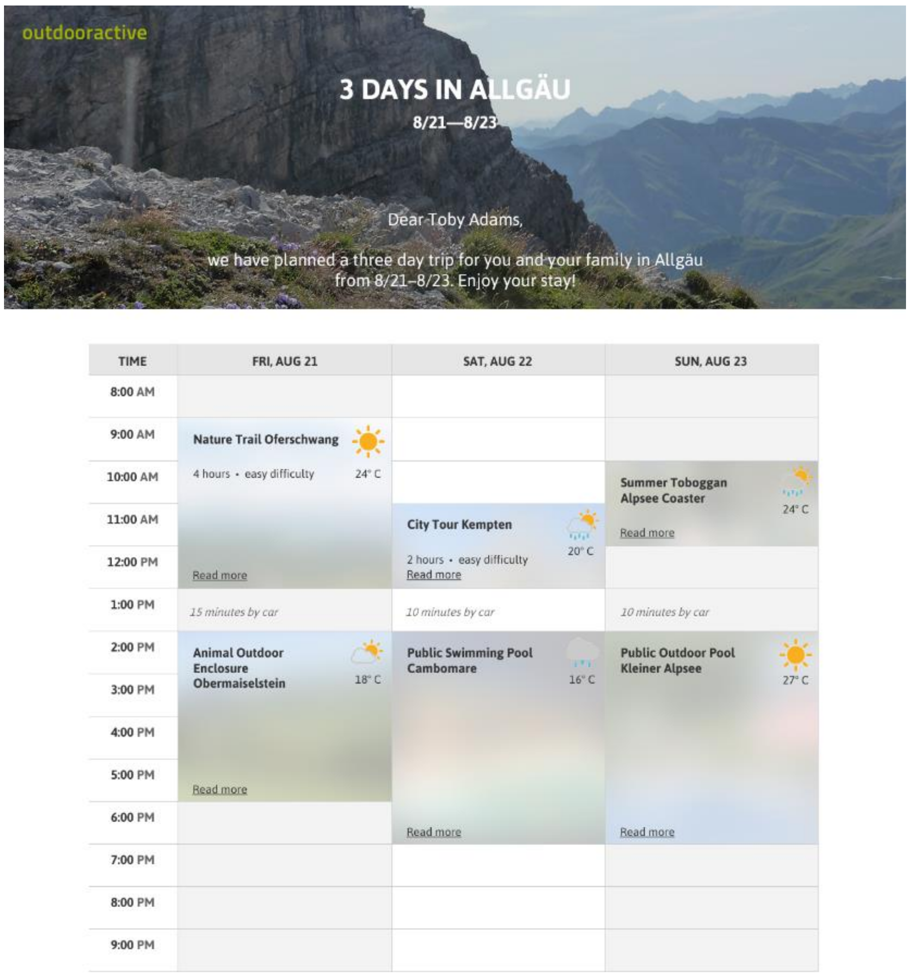
- Number of conflicts and
- Average distance of conflicting parties.

The specification demands a minimal number of conflicts. In case of conflict, a configuration of daily plans should maximise the average distance between the conflicting parties. Consequently, the target function is translated in the following preference:

```
PREFERRING
        conflict_count LOWEST                                  PRIOR TO
        average_distance_among_conflicts HIGHEST
```

Following the self-similarity of our design the BMO-property once more guarantees the optimality of the result set. It contains all optimal context-aware and stereotype-specific multi-daily plans per region to be displayed by a timetable at UI.

Finally, a linear timetable is constructed according to the extension theorem of Szpilrajn (1930). For every strict partial order, the generated (multi-)day plan is contained into a total order as shown in fig. 8:



outdooractive

3 DAYS IN ALLGÄU

8/21—8/23

Dear Toby Adams,

we have planned a three day trip for you and your family in Allgäu from 8/21–8/23. Enjoy your stay!

TIME	FRI, AUG 21	SAT, AUG 22	SUN, AUG 23
8:00 AM			
9:00 AM	Nature Trail Oferschwang		
10:00 AM	4 hours • easy difficulty 24° C		Summer Toboggan Alpsee Coaster 24° C
11:00 AM		City Tour Kempten	Read more
12:00 PM	Read more	2 hours • easy difficulty 20° C Read more	
1:00 PM	*15 minutes by car*	*10 minutes by car*	*10 minutes by car*
2:00 PM	Animal Outdoor Enclosure Obermaiselstein	Public Swimming Pool Cambomare	Public Outdoor Pool Kleiner Alpsee
3:00 PM	18° C	16° C	27° C
4:00 PM			
5:00 PM	Read more		
6:00 PM		Read more	Read more
7:00 PM			
8:00 PM			
9:00 PM			

Fig. 8. UI as timetable of the stereotype "Family" in the region Allgäu

The timetable is a marketing vision of outdooractive. The itinerary offers context-aware and stereotype-specific suggestions to tourists. It provides an interactive interface to arrange activities and to augment the plan with social events, meals, etc. It is evident that all activities follow the weather timeline of the forecast generating blocks of parametric total length (sub-daily plans).

4.5 Implementation and Performance

Details of the implementations are found in the appendix. The implementation relies on Preference SQL and SQL3. Both systems are running on Postgres databases. The run-time of the entire planning process is about one second for typical examples. Note that this process is not even a time-critical task.

The reactivity of the prototype already fulfils the soft time constraints of the application. Further runtime improvements are surely achieved if the Preference SQL prototype is exchanged by EXASolution 5.0. This preference implementation is supplied by the world's fastest analytic database of www.exasolution.com as proven in the TPC-H contest. The preference implementation of EXASOL AG is due to the project P-SOL in conjunction with the department of Prof. Kießling at the University of Augsburg. This project has also been funded by *Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie* (grant no. IUK-398/002). Mandl, Kozachuk, Endres, and Kießling (2015) show the performance of skyline queries in EXASOL's distributed server farm by scaling the data volumes of the TPC-H benchmark[1]. Thus this preference implementation may pave the way for huge state spaces and their preference evaluation.

5 Summary and Outlook

Similar to suggestions given in tourism information centres or provided by local residents, the proposed preference-based architecture uses a database and Preference SQL to generate appropriate activity suggestions for tourists as daily plans or even multi-day plans. The prototype implements a generic planning architecture by defining a controlled vocabulary, and then by deriving a questionnaire. Stereotypes and context are associated to the questionnaire. The feedback of the questionnaire is afterwards transformed into preferences. They are composed to complex preferences at different abstraction levels. The preferences are further enriched by soft requirements of the application modelled as preferences, too.

The preference theory guarantees the correctness, completeness, and optimality of the result. This BMO-characteristic reduces also the size of the result of an abstraction level to be delivered as input to the next stage. For the whole chain of queries, there are evidently no unnecessary, missing, or even better results!

The inclusion of hard constraints, such as opening hours, vacancies or the availability of tickets, will be the next step to extend the prototype with the option to also book

[1] http://www.tpc.org/tpch/spec/tpch2.17.0.pdf

offers. In addition, the transfer times between starting points and endpoints of subsequent activities have to be integrated to generate more realistic time schedules.

While preferences in the current version of the prototype are solely based on context and stereotype that each user is assigned to, embedding individual preferences will further improve the quality of plans generated for each user in the future. Personalisation can be then improved by adding sequential or temporal preferences, for example the preferred time to have lunch, or personal habits of doing activities in a specific order.

References

ANSI/NISO Z39.19 (2005). *Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies*.

Burkard, R., Dell'Amico, M. & Marello S. (2012). *Assignment Problems*. Society for Industrial and Applied Mathematics.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence* 32 (3): 333-377.

Chomicki, J. (2003). Preference formulas in relational queries. *ACM Transactions on Database Systems* 28(4): 427-488.

Döring, S. & Preisinger, T. (2008). Personalisation and Situation Awareness of the Search Process in Tourism. In P. O'Connor, P., Höpken, W. and Gretzel, U. (Eds.), *Information and Communication Technologies in Tourism 2008: Proceedings of the International Conference in Innsbruck, Austria*. Springer Vienna: 497-598.

Endres, M., Roocks, P., Wenzel F., Huhn, A., & Kießling, W. (2012). Handling of NULL Values in Preference Database Queries. *6th Multidisciplinary Workshop on Advances in Preference Handling in conjunction with ECAI 2012*. Montpellier, France.

Ghallab, M., Nau, D. & Traverso, P. (2004). *Automated planning theory and practice*. Morgan Kaufmann Publishers.

Gibson, H. & Yiannakis, A. (2002). Tourist roles, needs and the life course. *Annals of Tourism Research* 29: 358-383.

Gottlob, G., & Zicari, R. (1988). Closed World Databases Opened Through Null Values. In F. Bancilhon and D. J. DeWitt (Eds.), *VLDB '88 Proceedings of the 14th International Conference of Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA: 50-61.

Kießling, W., Hafenrichter, B., Fischer, S. & Holland, S. (2001). Preference XPath: A Query Language for E-Commerce. In H. U. Buhl, A. Huther and B. Reitwiesner (Eds.), *Information Age Economy*. Physika-Verlag: 427-440.

Kießling, W. (2002). Foundations of Preferences in Database Systems. In P. A. Bernstein, Y. E. Ioannidis, R. Ramakrishnan and D. Papadias (Eds.), *VDLB '02: Proceedings of the 28th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA: 311-322.

Kießling, W. & Köstler, G. (2002). Preference SQL – Design, Implementation, Experiences. In P. A. Bernstein, Y. E. Ioannidis, R. Ramakrishnan and D. Papadias (Eds.), *VDLB '02: Proceedings of the 28th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA: 900-1001.

Kießling, W. (2005). Preference Queries with SV-Semantics. In J. R. Haritsa and T. M. Vijayaraman (Eds.), *Advances in Data Management 2005 - Proceedings of the 11th International Conference on Management of Data*. Computer Society of India: 15-26.

Kießling, W., Endres, M. & Wenzel, F. (2011). The Preference SQL System – An Overview. In J. R. Haritsa and T. M. Vijayaraman (Eds.), *Bulletin of the Technical Committee on Data Engineering* 34(2): 11-18.

Likert, R. (1932). A Technique for the Measurement of Attitudes. *Archives of Psychology* 140: 1-55.

Liskov, B. H., & Wing, J. M. (2001). Behavioral Subtyping Using Invariants and Constraints. In: H. Bowman and J. Derrick (Eds.), *Formal Methods For Distributed Processing: A Survey of Object-Oriented Approaches.* Cambridge University Press New York, NY, USA: 254-280.

Mandl, S., Kozachuk, O., Endres, M. & Kießling, W. (2015). Preference Analytics in EXASolution. 16. Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW 2015), Hamburg, Deutschland.

Roocks, P., Endres, M., Mandl, S. & Kießling, W. (2012). Composition and Efficient Evaluation of Context-Aware Preference Queries. In S. Lee, Z. Peng, X. Zhou, Y. Moon, R. Unland and J. Yoo (Eds.), *Database Systems for Advanced Applications – Proceedings of the 17th International Conference DASFAA 2012.* Springer Heidelberg Dordrecht London New York: 82-96.

Szpilrajn, E. (1930). Sur l'extension de l'ordre partiel. *Fundamenta Mathematicae* 16: 386-389.

Wilkins, D. E. (1986). Hierarchical Planning: Definition and Implementation. In J. B. H. du Boulay, D. Hogg and L. Steels (Eds.), *Advances in Artificial Intelligence II, Proceedings of the 7th European Conference on Artificial Intelligence, ECAI 86*. North-Holland: 659-671.

Zhang, X. & Chomicki, J. (2011). Preference Queries over Sets. In Proceedings of 27th International Conference on Data Engineering (ICDE), IEEE: 1019-1030.

Acknowledgements

The project TSMP has been funded by *Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie* (grant no. IUK-1307-0004). Thank students Emanuel Huber, Uli Schweikl, Johannes Kastner, and Natalia Odolad in chrono-logical order for implementation support in TSMP.

A Appendix: Runtime Measurements of a Preference-Driven Planner

A.1 Configuration and Software Architecture

Table 5. Preference SQL Server and Preference SQL Client

	Preference SQL Server	**Preference SQL Client**
CPU	Intel(R) Xeon(R) CPU E5540	Intel i7-3540M
RAM	74 GB	16 GB
Frequency	2.53 GHz	3.0 GHz
Hard Disk	2000.4 GB	1 GB
Database	Postgres 8.4.13	Postgres 9.3.5
Name	Server (ursamajor)	local

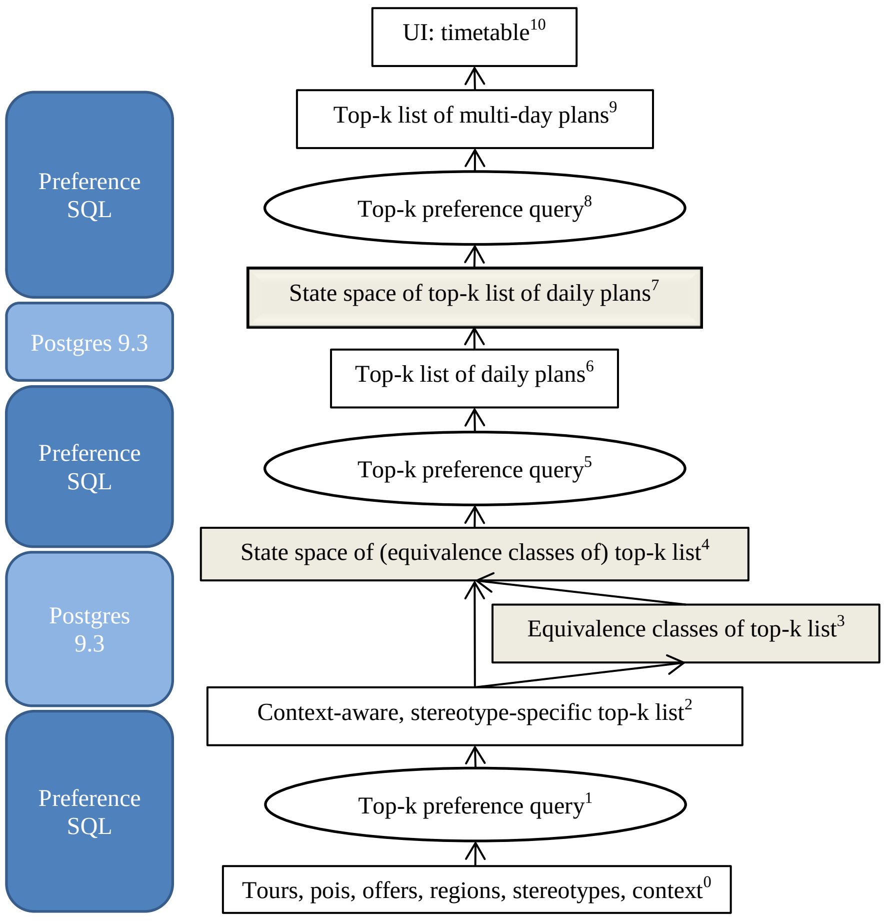

UI: timetable10
Top-k list of multi-day plans9
Top-k preference query8
State space of top-k list of daily plans7
Top-k list of daily plans6
Top-k preference query5
State space of (equivalence classes of) top-k list4
Equivalence classes of top-k list3
Context-aware, stereotype-specific top-k list2
Top-k preference query1
Tours, pois, offers, regions, stereotypes, context0
Preference SQL
Postgres 9.3
Preference SQL
Postgres 9.3
Preference SQL

Fig. 7. Software architecture of Vacation Planner from backend to frontend

A.2 Basic Relations of the Outdooractive Database

The planner relies on following basic relations and their tuples:

oa_tours	# 343248
oa_poi	# 591957
oa_offer	# 2841
oa_tourismarea	# 667

Concepts are related to each other by a semantic net which is implemented by the relation:

bc_relationrole	# 14420831

All relations have indices with regard to the primary key. A typical SQL query uses the semantic net to establish a semantic relation between concepts like activities and regions as follows:

```
SELECT    t.oa_category_id, ta.pid
FROM      oa_tour t, bc_relationrole r, oa_tourismarea ta
                       -- TourIsInTourismArea (26122)
WHERE     t.pid = r.source_id and r.relationtype_id = 26122 and r.target_id =
          ta.pid and t.state = 1 and ta.state = 1
```

Table 6. Runtime of SQL queries relying on bc_relationrole

	Server	**Runtime**	**I/O**
bc_relationrole	ursamajor	8.502	0.090

All joining attributes are indexed. Nevertheless, in a pre-processing stage the semantic net was eliminated by creating new relations which dispose of foreign keys referring the involved relations.

A.3 Taxonomies

The planner relies on activities which are modelled by the attribute "*category_id*" in the relations "*oa_tour*", "*oa_poi*", and "*oa_offer*" having the semantics of *doing something*. Activities, points of interests (POIs), and offers are arranged in just one taxonomy:

bc_category	# 76990

Starting with any concept as root, more specialised concepts which area modelled as children point to their unique father and so forth until no more children exist. The resulting tree is implemented within a relational schema. Since Preference SQL is incapable of handling hierarchical data structures as Preference XPath (see Kießling, Hafenrichter, Fischer and Holland, 2001), "flat" relations were created by a SQL3 query as e.g.

```
WITH RECURSIVE category(extended_name, id, niveau) AS (
SELECT       cast (name as text), pid, 1
FROM         bc_category
WHERE        pid = 2002  -- root
UNION
SELECT       c.extended_name || '/' || bcc.name, bcc.pid, c.niveau +1
FROM         bc_category bcc, category c
WHERE        bcc.parent_id = c.id
)

SELECT       *
FROM         category
ORDER BY     extended_name;
```

Table 7. Runtime of SQL3 query flattening the hierarchical relation "bc_category"

	Server	Runtime	I/O
bc_category	ursamajor	0.483	0.005

The attributes *pid* and *parent_id* have indices.

The semantic analysis of the outdooractive database is summarised by table 8:

Table 8. Size and kind of Controlled Vocabulary

Semantics	Count of concepts	Depth of Taxonomy
POIs	654	4
Activities	68	4
Offers	112	3
Relations	162	1
Annotations	194	1

The terms of the controlled vocabulary are only partially used in queries and all queries are self-contained by having eliminated the vast *ua_relationrole* relation and relying on indexed foreign keys.

A.4 Top-k Query to Generate the Basic Set for Planning

The planner needs a basic set of activities which are optimal with regard to a choosen stereotype and the actual context. Consider fig. 7 and the step indexed by 1.

To achieve this goal a preference is generated having the following generic structure: modelled by

- activity PARETO poi_asssessment PARETO ranking

The Preference SQL query is

```
SELECT          sta.*, level(p_activity) as level_activity, level(p_ranking) as
                level_ranking
FROM            stereotype_tour_assessment sta
PREFERRING
                (activity_layer lowest 1,1 as p_activity)
                and
                ((poi_count_of_layer_1 highest prior to poi_count_of_layer_2
                  highest prior to poi_count_of_layer_3 highest prior to
                  poi_count_of_layer_5 lowest prior to poi_count_of_layer_4
                  lowest) as p_poi)
                and
                (ranking between 80, 100, 20 as p_ranking)
```

Table 9. Runtime of a top-k query for generating the basic set of activities

	Server	Runtime [s]	I/O [s]	# Tuples
BMO	ursamajor	0.648	0.000	5
Top 10	ursamajor	0.632	0.002	10
Top 20	ursamjor	0.695	0.001	20
Top 100	ursamajor	0.703	0.015	100

The *sterotype_tour_assessment* relation is a further condensed relation of tours and POIs having 1644 tuples. The region is *Allgäu* and the stereotype is *athlete* having *good weather* conditions. The preference is evaluated by BNL.

By pre-processing the underlying database, the context-aware preferences are evaluated for each configuration defined by context and stereotype. These parameters are stored together with the result set of the context-aware preference evaluation. Later on, they are used as filters to get the correct result set for any context. The context-aware result set defines the basic set to generate combinations of its set elements as nodes of an abstract state space as shown in the next steps of fig. 7.

A.5 Generation of Equivalence Classes

After having generated the basic set of the planner, the most interesting attribute is *duration* being of type *FLOAT*. All values of this attribute are transformed to *INT* by *CEIL(anyAttribute/divisor)::int* ➔ *equivClass_ID* and counted per the discretised attribute. These tuples (anyDiscreteAttribute, count) are a representation of a multiset used as input to a state space of equivalence classes. Consider fig. 7 and the step indexed by 3.

Following query is used:

```
SELECT          discreteDuration as equivClass_ID, count(*)
FROM            (SELECT ceil(duration/60)::int4 as discreteDuration  -- n hours
                 FROM stereotype_tour_assessment
                ) as tmp
GROUP BY        discreteDuration
```

The costs of this transformation are negligible. The equivClass_ID is the primary key of a new relation and acts as filter by

- WHERE anyAttribute > equivClass_ID - 1 AND anyAttribute <= equivClass_ID

in order to identify all members of an equivalence class in the original relation.

A.6 Generation of a State Space for Daily Plans

The combination of top-k elements of the basic set is modelled by a state space. Any attribute of the basic set may be agglomerated:

- Duration ➔ total duration of combination
- Multilevel ➔ overall quality

E.g. the *duration* attribute is used.

A.7 Power Set of a Set of Activities

A power set guarantees that each element of the basic set appears only uniquely in each combination. Consider fig. 7 and the step indexed by 4.

The basic structure of the query is:

```
CREATE          temp sequence seq;

CREATE          table ua_day_statespace  as
WITH RECURSIVE stateSpace (ancestors, ego, agglomerate, iteration, id, resources)
AS (
        SELECT          ARRAY[0::int4] as ancestors, t.id as ego, t.duration as
                        agglomerate, 1 as iteration,  nextval('seq') as id,
                        cast(2 ^ (t.id-1) as bigint) as resources
        FROM            basic_set t
        WHERE           t.id <= 5 -- parameter: 5, 10, 15, 20, …
                        -- AND t.duration <= 24                 -- restriction

        UNION
```

```
          SELECT         array_cat(s.ancestors, ARRAY[s.ego]) as ancestors, t.id as
                         ego, s. agglomerate + t.duration as agglomerate,
                         s.iteration +1 as iteration,   nextval('seq') as id,
                         resources | cast(2 ^ (ego-1) as bigint) as resources
          FROM           basic_set t, stateSpace s

          WHERE          t.id > s.ego
                         -- AND s. agglomerate + t.duration <= 24     -- restriction
                         AND t.id <= 5                  -- parameter: 5, 10, 15, 20, …
)

SELECT *
FROM stateSpace
UNION
 -- NULL-Element nach Bedarf
SELECT NULL, NULL, 0, 0, 0, 0
ORDER BY iteration, ancestors, ego
```

Table 10. Runtime of generating the state space of a set of activities

Size of basic set: k	Server	Runtime [s]	I/O [s]	# Tuples
5	local	0.062	0.000	32
10	local	0.194	0.000	1024
15	local	0.657	0.000	32768
17	local	2.238	0.000	131072
20	local	23.985	0.000	1048576

A.8 Power Set of a Multiset of Activities / Equivalence Classes

A multiset allows the reuse of any element in the basic set several times in each combination. For each element the maximal reuse is constricted as individual parameter. Consider fig. 7 and the step indexed by 4.

The basic structure of the query is:

```
CREATE          temp sequence seq;
-- ALTER SEQUENCE seq RESTART WITH 1;
```

```
CREATE          table ua_day_statespace
AS
WITH RECURSIVE         stateSpace (id, ancestors, agglomerate, iteration,
                                  freedom_degree)
AS (
        WITH
        para AS (
                SELECT          5 as cardinal_number     --Parameter: 5, 10, 15, 20
        ),

        restriction AS (
                SELECT          ARRAY (select 1          -- constant 1 =  Power set
                FROM            basic_set t, para p
                WHERE           t.id <= p.cardinal_number
                ORDER BY        id) as init
         ) -- WITH restriction_init

        SELECT          nextval('seq') as id, array_cat(ARRAY[0::int4],
                        ARRAY[t.id::int4]) as ancestors, t.duration as
                        agglomerate,
                        1 as iteration, array_cat( array_cat( r.init [1:t.id-1],
                        ARRAY [r.init[t.id] -1]),
                        r.init [t.id +1:array_length(r.init, 1)]) as freedom_degree
        FROM            basic_set t, restriction r, para p
        WHERE           t.id <= p.cardinal_number
                        -- and t.duration <= 24                      -- restriction

        UNION
```

```
          SELECT          nextval('seq') as id, array_cat(s.ancestors,
                          ARRAY[t.id::int4]) as ancestors,
                          s. agglomerate + t.duration as agglomerate,
                          s.iteration +1 as iteration,
                                                        -- DEC() after use
                          array_cat( array_cat( s.freedom_degree [1:t.id-1],
                          ARRAY [s.freedom_degree[t.id] -1]),
                          s.freedom_degree [t.id +1:array_length(s.freedom_degree,
                          1)]) as freedom_degree
          FROM            basic_set t, stateSpace s, para p

          WHERE            t.id <= p.cardinal_number                -- Parameter
                           -- AND  s. agglomerate + t.duration <= 24   -- restriction
                          AND s.freedom_degree[t.id] > 0        -- restriction of reuse
                                  -- only 1 representant
                          AND t.id >=  s.ancestors[s.iteration+1]
)

SELECT           *
FROM            stateSpace
UNION
 -- NULL, if necessary
SELECT          0, ARRAY[0::int4], 0, 0, init
FROM
                (SELECT         ARRAY (select 1    -- constant 1 =  Power set
                 FROM           basic_set t
                WHERE           t.id <= 5                    -- Parameter
                ORDR BY         id) as init
                ) as restriction
ORDER BY iteration, ancestors;
```

Table 11. Runtime of generating the state space of a multiset of activities / equivalence classes

Size of basic set: k	Server	Runtime	I/O	# Tuples
5	local	0.129	0.000	32
10	local	0.172	0.000	1024
15	local	1.029	0.000	32768
17	local	3.345	0.000	131072
20	local	28.657	0.000	1048576

The reuse of elements is restricted to 1. Thus a power set is generated in order to compare the runtime of both generation algorithms, easily.

A.9 Restrictions of State Spaces

Normally, every application implies hard constraints which cut the state space. Creating a daily plan with hourly activities, the total duration is always less equal 24 hours. The hard constraint is:

- WHERE agglomerate <= 24

This restriction is valid for a power set based on a set (S) as well as based on a multiset (MS). The queries of g) and h) contain this restriction as comment.

Table 12. Runtime of generating a restricted state space

Size of basic set: k	Server	# Tuples	Reduction	Runtime [s], S	Runtime [s], MS
5	local	32	0.00%	0.099	0.104
10	local	394	61,52%	0.109	0.126
15	local	676	97,94%	0.121	0.132
20	local	751	99,43%	0.146	0.185
25	local	762	99,93%	0.136	0.143

Activities are discretised to get half-hour or hourly activities as equivalence classes. Clearly, a finer discretisation achieves a greater state space and therefore a longer runtime.

A.10 Generation of Daily Plans by Preference Evaluation

Consider fig. 7 and the step indexed by 5. Following two preferences are of interest:

```
1) PREFERRING
       (total_duration LESS THAN 8 , 1 PRIOR TO
        total_duration AROUND 8, 1) AS duration_preference


2) PREFERRING
       (activity_count HIGHEST) AS alternation_preference
```

The agglomerated attribute corresponds to the total duration. It is handled as a parameter depending on stereotype and context.

The preference query P1 with 1) is:

```
CREATE TABLE ua_daily_plan_1dim
AS
SELECT          id, agglomerate, level(duration_preference)
FROM            ua_day_statespace
PREFERRING
                (agglomerate LESS THAN 8 , 1 PRIOR TO
                  agglomerate AROUND 8, 1) AS duration_preference
```

The preference query P2 with 1) PARETO 2) is:

```
CREATE TABLE ua_daily_plan_2dim
AS
SELECT          id, agglomerate, iteration, level(duration_preference),
                level(alternation_preference)
FROM            ua_day_statespace
PREFERRING
                (agglomerate LESS THAN 8 , 1 PRIOR TO
                  agglomerate AROUND 8, 1 AS duration_preference)
                AND
                (iteration HIGHEST 100, 1 AS alternation_preference)
```

Table 13. Runtime of two instances of the preference evaluation of the same state space for daily plans defined on basic sets of different size k

Size of basic set: k	Server	P1: BMO of #, Runtime [s]	P2: BMO of #, Runtime [s]:
5	local	3 of 32, 0.089	4 of 32, 0.077
10	local	6 of 394, 0.166	5 of 394, 0.132
15	local	6 of 676, 0.115	5 of 676, 0.219
20	local	6 of 751, 0.187	5 of 751, 0.088
25	local	6 of 762, 0.150	5 of 762, 0.085

BMO is the number of the best matching objects of a preference P. # is the count of daily plans. With regard to preference P1 all plans are perfect because they all have a total duration of 8 hours.

A.11 Generation of a State Space for Multi-Day Plans

Consider fig. 7 and the step indexed by 7. Now combinations of daily plans are constructed. The basic set of the state space generation for multi-day plans is the result set of the best suitable daily plans generated by a top-k preference query.

Following query generates the state space of multi-day plans as power set, since each outcome of a daily plan is unique:

```
CREATE temp SEQUENCE seq;

-- ALTER SEQUENCE seq RESTART WITH 1;

CREATE TABLE ua_week_statespace

AS

WITH RECURSIVE stateSpace (weeklyplan_id, ancestors, agglomerate, iteration)

AS (

        WITH init_wp

        AS (

                SELECT          DISTINCT id AS dailyplan_id, agglomerate

                FROM            ua_daily_plan

        )


        SELECT          nextval('seq') as weeklyplan_id, ARRAY[w.dailyplan_id]

                        as ancestors,  agglomerate, 1 as iteration

        FROM            init_wp w
```

```
          UNION

          SELECT        DISTINCT nextval('seq') as weeklyplan_id,
                        array_cat(ARRAY[w.dailyplan_id],
                        s.ancestors) as ancestors,
                        s. agglomerate + w. agglomerate as agglomerate,
                        s.iteration +1 as iteration
          FROM          init_wp w, stateSpace s
          WHERE         w.dailyplan_id > s.ancestors[1]   --StateSpace=PowerSet
)

SELECT        *
FROM          stateSpace
ORDER BY      iteration, ancestors
```

The runtime behaviour is similar to table 10, since a power set is generated as state space. The result size of the underlying top-k query with TOP LEVEL = 0 ranges from 3 to 6 as shown in table 13, therefore 0.2 sec seems to be an appropriate upper limit of the runtime.

A.12 Assignment of Quality by Counting Conflicts

Consider fig. 7 and the step indexed by 8. Now adequate combinations of daily plans are needed. Adequateness is modelled by following preference:

- PREFERRING count_of_conflicts LOWEST

A conflict is defined as the repetition of an activity to avoid ennui of tourists.

The number of conflicts is associated to each state of the state space as a quality assignment by following query:

```
CREATE TABLE ua_week_conflicts
AS
          WITH para
          AS (
                    SELECT        3 as count_of_days
          ),
-- WP => {DP} => {Activity}
partner
AS (
        SELECT          weeklyplan_id, dailyplan_id, activity
        FROM
        (
                SELECT          weeklyplan_id, dailyplan_id,
                                unnest(d.ancestors) as activity
                FROM            ua_day_statespace_10 d,
                (
                        SELECT          w.weeklyplan_id,
                                        unnest(w.ancestors) as
                                        dailyplan_id
                        FROM            ua_week_statespace_1dim_10
                                        w,  para
                        -- WHERE        w.iteration =
                                        para.count_of_days
                                        --Parameter:  Size of time table
                ) as wp2Ndp
                WHERE           dailyplan_id = d.id
        ) as wp2Ndp2Nactivity
        WHERE           activity <> 0
),
-- relation graph per WP
pairing
AS (
            SELECT          wp1.weeklyplan_id as weeklyplan_id,
                            wp1.dailyplan_id as dailyplan_id_1,
                            wp1.activity as activity_1,
                            wp2.dailyplan_id as dailyplan_id_2,
                            wp2.activity as activity_2
            FROM            partner wp1, partner wp2
            -- upper triangular matrix (symmetry)
            WHERE           wp1.weeklyplan_id = wp2.weeklyplan_id
                            AND wp1.dailyplan_id < wp2.dailyplan_id

UNION

 -- Plus combinations of ONE element
SELECT          weeklyplan_id as weeklyplan_id,
                dailyplan_id as dailyplan_id_1, activity as activity_1,
                null as dailyplan_id_2, null as activity_2
FROM            partner
WHERE           weeklyplan_id IN
                (SELECT         weeklyplan_id
                 FROM           ua_week_statespace_1dim_10
                WHERE           iteration = 1)
),
          -- conflict graph
          conflict_graph_per_wp
          AS (
                    SELECT          weeklyplan_id, dailyplan_id_1, dailyplan_id_2,
                                    activity_1, activity_2,
                                    CASE    WHEN activity_1 = activity_2
                                            THEN 1   -- conflict
                                            ELSE 0
                                    END AS conflict
                    FROM            pairing
          ),

          agg_conflict_graph_in_wp
          AS (
                    SELECT          weeklyplan_id, dailyplan_id_1, dailyplan_id_2,
                                    sum(conflict) as agg_count_conflict_in_days
                    FROM            conflict_graph_per_wp
                    GROUP BY        weeklyplan_id, dailyplan_id_1, dailyplan_id_2
          )

SELECT          weeklyplan_id, sum(agg_count_conflict_in_days)
                as count_conflict_of_wp
FROM            agg_conflict_graph_in_wp
GROUP BY        weeklyplan_id
ORDER BY        weeklyplan_id;
```

Table 14. Runtime of counting conflicts of a 3-day plan

Size of basic set: k	Server	Runtime [s]
5	local	0.143
10	local	0.139
15	local	0.187
20	local	0.203
25	local	0.151

A.13 Preference Evaluation of Multi-Day Plans

Consider fig.7 and the step indexed by 8, again. Now all information is available to appraise the adequate combinations of daily plans. Adequateness is modelled by following preference:

- PREFERRING count_of_conflicts LOWEST

The following preference query minimises the count of conflicts:

```
CREATE TABLE ua_weekly_plan

AS

SELECT      w.*, level(p_conflict) AS conflict_level

FROM        ua_week_conflicts c, ua_week_statespace w

WHERE       c.weeklyplan_id = w.weeklyplan_id

            -- Parameter: number of days in the weekly plan / time table

            AND w.iteration = 3

PREFERRING

            c.count_conflict_of_wp LOWEST 0 , 1 AS p_conflict
GROUPING    iteration

-- TOP 10;
```

Table 15. Runtime of preference evaluation in a state space for multi-day plans defined on a basic set of size k

(Size of initial basic set, # optimal daily plans)	Server	Runtime [s]	# Conflicts	|BMO|
(5, 3)	local	0.080	3	1
(10, 6)	local	0.101	0	5
(15, 6)	local	0.109	0	5
(20, 6)	local	0.099	0	5
(25, 6)	local	0.103	0	5

If the k-parameter of the top-k-operator is greater than 10, sufficiently many alternatives exist to avoid conflicts. The number of days in the time table was restricted to 3, but the state space also disposes of combinations having less or even more than 3-day timetables.

If conflicts exist, the shared activities should have a maximal distance between their occurrences.

A.14 Total Runtime as Summary

To measure the total runtime, a consistent use case was defined by:

- Size of basic set = 10
- Size of best activities for 1 day = 10
- Maximal size of alternative multi-day plans = 10
- Maximal days = 5

TOP 10 as part of Preference SQL achieves the above requirements. The planner is parameterised by a hard constraint of having <= 5 days.

Table 16. Total Runtime of the use case

Query	SQL3	PSQL	Response time
h) + i) Restricted power set of daily activities	0.200		
j) Preference wrt. daily total duration (P1)		0.185	
k) Power set of daily plans	0.240		
l) Quality assessment by count of conflicts	0.101		
m) Preference wrt. minimal conflicts		0.315	
Total runtime of component	**0.541**	**0.500**	
Total runtime of application			**1.041**

The total runtime of about 1 second is sufficient to achieve the runtime requirements of a planning application on pre-processed data of the Alpstein database.

Caveat

The spreading of multiple measurements may surpass 10 percent of the minimal measurement.