

Find your Dog – FiDo: a dialogue system to help its user to decide which dog to get

Project report, Dialogue systems 2, LT2319, Autumn 2022

Max Boholm

1. Introduction

The choice of which type of dog to get might be a complex matter due to the great variability of dog breeds. For example, breeds are different in terms of their level of energy, capacity for being trained and protectiveness.

Find your Dog, FiDo, is a dialogue system which assists its user in deciding which type of dog to get. This goal is pursued by asking the user a series of question about her/his preferences for certain traits (e.g., trainability and energy), thereby filtering out dogs in a database that fit the preferences, i.e. an *incremental search*. Also, breeds can be described and compared with regard to certain traits, thereby giving the user additional information that is relevant for deciding. FiDo is implemented with the Talkamatic Dialogue Manager (TDM) (Larsson & Berman 2016). Theoretically TDM builds on Issue-based Dialogue Management (Larsson 2002). FiDo uses the API-Ninjas' Dog API, but can also be run with a local database. This report describe the development of FiDo (its functionality and implementation in TDM) and discuss the functionality of TDM in relation to the task at hand.

2. Data

2.1. Data collection

For the development of FiDo, role playing dialogue that simulated a dog searching service was observed, involving a information-giving “system” and an information-seeking “user”. In total, ten dialogues has been recorded, transcribed and distilled for this project (Larsson *et al.* 2001). In all except for one of these dialogues, the present author has served the role as system. Two other persons have participated as users. In one dialogue, one of these persons played the role of the system.

Data was collected in three phases. In the first phase (also reported for Lab 1 of LT2319), two dialogues were recorded. Due to limited knowledge of TDM at the time, instructions for the dialogue did not have a clear link to the functionality of TDM. The invited participant was informed of the purpose of the dialogue, i.e. having a “system” assisting a “user” to decide which kind of dog to get; that she would participate in two dialogues, one as the system and one as the user. In order to provide some structure to the dialogue, both participants had a list of features for dogs from the API-Ninjas Dog API (see next section) which were carefully read. Participants had also a list of ten dog names to avoid being stuck on what dogs to refer to. The actual association between breeds and their assumed features was declared to be of no importance. As it turned out, the dialogues of this first naive phase were of limited use for the development of FiDo. Their structure is

not straightforward to implement in TDM. However, they are interesting from the perspective of what FiDo cannot do.

In the second and third phase, dialogues more clearly simulated incremental search as implemented in TDM by applying the domain element `<parameters>` (see sect. 3.3.2). The system-participant had a program, the *simulator* (a Jupyter notebook), to search the API-Ninjas Dog API with the five parameters *trainability*, *shedding*, *energy*, *barking*, and *protectiveness* and thereby filter out the set of dogs satisfying user preferences. Compared with the second phase, the third phase tests more sophisticated questions about feature preferences (see sect. 3.2). Also, the third phase of data collection addressed additional features not addressed in the second phase, namely describing and comparing dogs. Below is an example of a (distilled) dialogue from the third phase:

- (1) S: What can I help you with?
U: I want to buy a dog
S: okay. then I'll ask you a few questions and then we'll see if we can decide which dog that suits you. so first question. to what degree should your dog be trainable
U: to a great extent
S: yes. to what degree should your dog shed
U: low degree
S: to what degree should your dog have energy
U: high degree
S: to what degree should your dog bark
U: quite low
S: to what degree should your dog be protective
U: quite low
S: then I recommend a Volpino Italiano
U: great

2.2. API-Ninjas' Dog API

The API-Ninjas Dog API is a service for information about more than 200 dogs. Requests to the API are defined by at least one of the following parameters `name`, `min_height`, `max_height`, `min_weight`, `max_weight`, `min_life_expectancy`, `max_life_expectancy`, `shedding`, `barking`, `energy`, `protectiveness`, `trainability` and `offset` (i.e. number of results to offset for pagination). The response include some additional parameters of dog entries, which *not* are possible to include in the request (e.g., `good_with_children`, `good_with_other_dogs`, `grooming`, `drooling` and `playfulness`). For many parameters (e.g., `shedding`, `barking`, `energy`, `protectiveness`, and `trainability`), the possible value range from 0 to 5.

While promising at first, the API has a number of limitations for this project:

- i. The API only returns 20 hits, which has consequences for incremental search (the dogs filtered out may not match the dogs in the database that satisfy the user's preferences).

- ii. Some features of dogs which could have been of interest to this project (e.g., `good_with_children` and `good_with_other_dogs`) are not possible parameters of requests.
- iii. The API is unreliable. For example, a search for dogs defined by:
`trainability=5&shedding=3&energy=4&barking=3&protectiveness=3`
 will return nine dogs, which includes Slovakian Wirehaired Pointer, where `shedding=2` (not 3). (The extent of this issue has not been explored in detail.)
- iv. It is not possible to include the same parameter with two different values in the same request. For example, you cannot identify dogs where `trainability=4` and `trainability=5` in *one* request.

As we will return to below (sect. 3.2.3), the last of these issues (iv) has implications for interpreting user answers about preferences in FiDo.

2.3. Local database

FiDo has been designed to work with the API-Ninjas’ Dog API, but also with a local database. From 30 random requests to the API-Ninjas’ Dog API, a local database (a json file) containing 177 unique dogs has been built. This local database simplifies system development, but also enables variants of incremental search not possible using the web API. We will return to this issue below (sect. 3.2.3).

3. Implementation

In TDM, projects are organized in Dialogue Domain Descriptions (DDD), which consist of:

- a *domain*, which defines goals and plans for achieving those goals
- an *ontology*, which defines individuals, predicates and actions being spoken about
- a *service interface*, which defines the interface between the DDD and a service
- a *grammar*, which defines system utterances and interpretation of user utterances

In addition, a TDM pipeline can be further specified by, for example: an *http-service*, which defines external actions requested by the dialogue manager and their responses, for example, requests and responses from an API; and a statistical model for *natural language understanding* (NLU), trained through RASA (Bocklisch *et al.* 2017). After some notes on incremental search (sect. 3.1) and preference probing (sect. 3.2), each component of FiDo is described (sect. 3.3–3.8). For details, see the code of FiDo available here: https://github.com/mbohlm/lt2319_proj.

3.1. Incremental search

Incremental search is the process whereby the answer, A , to a question, Q , is searched by finding answers, a_1, \dots, a_n , to a series of sub-questions, q_1, \dots, q_n , such that those answers restrict the set of possible values of A , $\{e_1, \dots, e_n\}$. In a computational implementation of incremental search, where sub-questions q_1 to q_n are predefined, there are three possible outcomes of the process:

- i. *Success*: the incremental search results in a remainder set R , such that it contains a single element e_i ; if so, $A = e_i$.
- ii. *Over-populated (OP)*: after q_1 to q_n have been answered, R still contain more than one element, i.e. the cardinality of R is larger than 1 ($|R| > 1$).
- iii. *Under-populated (UP)*: an answer a_i to a sub-question q_i restricts R too narrowly, such that it becomes empty ($R = \emptyset$).

With OP and UP outcomes the incremental search (itself) does not resolve Q . The capacity of FiDo to handle these three situations is described further below.

3.2. Probing for preferences

3.2.1. A basic idea

A key issue to address in this project is how to link features of dogs in the database with the information seeking behavior of the system, i.e. how the system should ask questions about traits (q_1, \dots, q_n) and how user answers (a_1, \dots, a_n) to those questions should be interpreted. Following Larsson (2002), q_1, \dots, q_n can be formalized as wh-questions of the type *?x.preference-score-of-F(x)* (more or less: *what is the preference score of F?*). Answers should be interpreted as numbers from 0 to 5, resulting in propositions, such as *preference-score-of-F(3)*, which can be used to narrow down the the set of dogs. While the basic procedure is straightforward, how to formulate these questions and how to interpret their answers is far from trivial. In fact, how to best structure preference probing in FiDo is an unresolved issue. In what follows, some problems with preference probing and their (failed) solutions are discussed.

3.2.2. Problems, approaches and limitations

The most direct strategy to determine user preferences would be to ask a questions about a trait (F), retrieve a score (from the user's answer), s , and then strictly restrict the set R based on that proposition, $F(s)$. For example, consider (2), where “ $\rightarrow \{\dots\}$ ” represents the interpretation of an utterance with regard to the set defined for incremental search:

- Strict approach*
- (2) S> On a scale from zero to five, what value should your dog have on F ?
 U> three
 $\rightarrow \{\text{dog: dog's value for } F = 3\}$

This approach assumes that the user wants a dog where $F = 3$ – *not more, not less*. While this analysis may be plausible for some preferences, it is untenable as a general approach to preference probing. Rather than being an isolated single value on a scale, preferences sometimes cover a continuous range of values. For example, if someone answers that their preference for barking is “one”, this person might very well prefer (or accept) dogs where *barking* = 0. Moreover, if someone, for example, answers that their preference for trainability is “four”, s/he might accept dogs that are even more so (i.e. *trainability* = 5).

One approach to account for a range of values of preferences would be to address the level of acceptability of traits, for example:

Acceptable-level approach

- (3) S> On a scale from zero to five, how much energy *must* your dog have? (*preferred*)
U> two
→ {dog: dog's value for energy ≥ 2 } (*more than or equal to*)
- (4) S> On a scale from zero to five, how much energy *can* your dog have? (*dispreferred*)
U> two
→ {dog: dog's value for energy ≤ 2 } (*less than or equal to*)

Note that the interpretation of the scores in the acceptable-level approach requires information on whether the feature is preferred or not. This might require additional information seeking in the dialogue to establish knowledge of. Thus, this approach would require two steps of preference probing. In contrast, one might hope for an approach that can capture both *the basic orientation towards a trait* (e.g., does the user like barking or not?) and *a range of preferences* (e.g., how much does the user (dis)like barking?) by a single question-answer pair. What we might call *bipolar gradable questions*, possibly satisfy these criteria. Consider, the questions in (1), sect. 2.1, for an example. The first question is presented below.

Bipolar-gradable approach (lexical)

- (5) S: to what degree should your dog be trainable?
U: to a great extent

Presumably, in answering *S*'s question, *U* must first decide if her preference for trainability is high or low (positive or negative). Second, *U* indicates a strong preference for trainability by using the adjective *great*, rather than just *high* (compare the other answers by *U* in example 1). In (5) – and (1) – *S*'s question is formulated by words only (hence the term “lexical”). A major disadvantage of implementing such questions for a dog selecting system like FiDo is the ambiguity and vagueness of user answers with regard to values of parameters in the database. For instance, what does *great* mean in this context? Should we interpret *U*'s answer as 4 or 5 (or something else)?

An attempted escape from this vagueness, we might (putting mathematical correctness aside) assume that the scale [0, ..., 5] used for dogs in the database, is transformed to the scale [-3, -2, -1, +1, +2, +3] (no zero). A *numeric* example of bipolar gradable questions would then be:

Bipolar-gradable approach (numeric):

- (4) S> On a scale from minus three to plus three, to what degree should your dog (be) *F*?
U> two → {dog: dog's value for *F* ≥ 4 }

The tentative assumption here is that a positive answer indicates that the trait is preferred, while a negative number that it is not. Moreover, a range of values is identified from the number answered to the minimum/maximum of the scale. Although, there might be some plausibility of the numeric bipolar-gradable approach, it is far from evident that questions like those in (4) are, in fact, interpreted as such. For example, does the polarity of the answer really imply a general orientation for the trait as preferred, or not?

The semantics of preference probing is further complicated by the observation that different traits might be best handled by different strategies. For example, consider barking. For this trait, a range of values might best capture user preferences (e.g., if someone prefers dogs where *barking* = 1, s/he might also accept dogs where *barking* = 0). In contrast, consider a trait like energy. Here a preference like *energy* = 3 is perhaps best understood in the “not more, not less” sense. That is the user might want a dog that is not too calm, but not too energetic either.

3.2.3. Additional issues

There are additional issues with the above approaches, relating to the computational implementation of incremental search. Compared with the preference-as-range approaches, the strict approach will result in more UP outcomes of the incremental search (given the same number of sub-questions). Another problem exists for the preference-as-range approaches, which results from

- i. limitations of the API used in this project,
- ii. the time allowed by TDM for passing a request to the http service and expecting a response back; and
- iii. the way incremental search is implemented in TDM.

As noted above, it is not possible to include the same parameter more than once in a request to the the API-Ninjas’ Dog API (see sect. 2.2). For example, you cannot make *one* request where *barking* = 1 *and* *barking* = 0. Thus, in order to cover a range of values of *the same* trait (parameter), you have to iterate over several values, making one request for each. Moreover, in incremental search in TDM the remaining set (*R*) of the previous step is not remembered for later use. Instead, after each $q_i - a_i$ interaction, restrictions of *R* based on previous steps ($q_j - a_j$, where $j < i$) must be computed anew. Taken together, the need for iteration over values and TDM not storing *R* from previous steps result in preference-as-range approaches requiring several API requests; an increasing number at each step of the incremental search. These API requests take time. As the number of API request reaches above a critical number, the time between TDM’s request to the http-service and the response back from the http-service takes more than 2 seconds, which activates a built in timeout in TDM, and the process fails. Note that increasing the timeout only would be a solution coming a (great) cost, as it would result in turn-taking lagging in the dialogue.

3.2.4. Summary and a preliminary solution

Ultimately, more knowledge on the psychology and semantics of preference probing is needed in order to determine which of the three approaches to preference probing that is best for this project. However, thorough study of this is outside the scope of the present project. To summarize, we are left with the following pros and cons for each approach:

Table 1: Pros and cons of preference probing approaches

Approach	Pros	Cons
<i>Strict</i>	Might capture the psychology	Might not capture preference

		of preference of some traits (e.g. energy) Minimizes the number of API requests needed	psychology of all traits (e.g. barking) Strong tendency towards under-populated outcomes
Preference-as-range approaches	<i>Acceptable level</i>	Might capture the psychology of preference of some traits (e.g. barking) Clear semantics	Might not capture all aspects of the psychology of preference of all traits; e.g., although someone has an acceptable level for energy, the preference might be better described as s, not more, not less Excessive number of API requests Requires extra step of determining basic preference for traits
	<i>Lexical bipolar gradable</i>	Might capture the psychology of preference of some traits (e.g. barking) Compared with numeric variant, clearer orientation towards the (dis)preference of the trait	Might not capture the psychology of preference of all traits (e.g. energy) Excessive number of API requests Compared with numeric variant, unclear mapping to numeric values
	<i>Numeric bipolar gradable</i>	Might capture the psychology of preference of some traits (e.g., barking) Compared with lexical variant, clearer mapping to numeric values	Might not capture preference psychology of all traits (e.g. energy) Excessive number of API requests Compared with lexical variant, unclear semantics with regards to (dis)preference for traits

As seen from Table 1, there is no perfect solution for preference probing to be implemented in FiDo. However, a decision has to be made, despite its shortcomings. In the present implementation, it is possible to run FiDo in one of two *database modes*:

- i. *local mode*, which uses the local database (see sect. 2.3) for incremental search and where user answers to system preference probing questions are interpreted along the lines of (numeric) bipolar gradeables, i.e. covering a range of values
- ii. *API mode*, which uses the web API (see sect. 2.2) for incremental search and where user answers to system preference probing questions are interpreted strictly.

Note that system questions are the same in both database modes, only the interpretation of user answers differs, i.e. the TDM grammar is the same in both database modes. The two modes can be set by a global parameter of the `http_service.py` file: for *API mode* set `db_mode_is_API` to `True`; for *local mode* set `db_mode_is_API` to `False`. Interaction tests presuppose *local mode*.

3.3. Domain

3.3.1. Overall goal

The overall goal of FiDo is to perform the action `suggest_dog`. This goal contains a plan to

- i. inform the user about the procedure for the task
- ii. find out *what dog the user should get*
(`<findout type="wh_question" predicate="what_dog_to_get"/>`),
- iii. invoke a service action (`<invoke_service_action name="SuggestDog"/>`).

Presently, `SuggestDog` does not involve anything other than returning a `successful_action_response`. It serves the role of a placeholder that enables other functionalities of TDM, such as a `<report>` (in the grammar) after the action is performed. In FiDo this is used to inform the user of the dog identified by incremental search. Indeed, this might seem to be a tortuous solution. However, there are two reasons for defining the overall goal of FiDo this way. First, this design is compatible with TDM's built in mechanisms for incremental search (see next section). My first intuition was to define the overall task of FiDo as a goal to resolve the question *what dog should the user get?*. However, as far as I can tell, this is not a viable option, since the `<parameters>` of incremental search (see next section) must be linked to a predicate of a `<findout>`; it cannot be the predicate of a `<goal type="resolve">`.¹ Second, although `SuggestDog` is a placeholder in the present implementation of FiDo, we can imagine developments of this service so that `SuggestDog` serves some additional purpose, for example, providing contact information to dog breeders nearby.

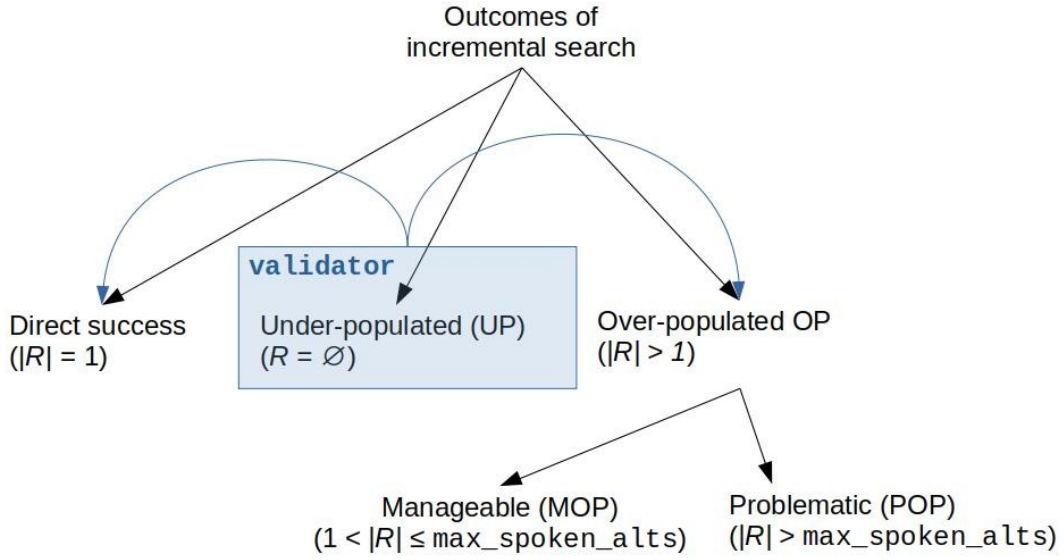
3.3.2. Incremental search in TDM

In TDM there is built-in support for incremental search. This is done by defining an element `<parameters>` (in the domain), with `incremental="true"`, which is linked to a predicate of a `<findout>`, in our case: `what_dog_to_get`. As children of `<parameters>` there are five `<ask_feature>` elements. i.e. the sub-questions (q_1 , etc.). For each `<ask_feature>` there is a defined predicate, one for each of the traits *trainability*, *shedding*, *energy*, *barking* and *protectiveness*, named `pf_trainability` etc. (following the convention "pf_" followed by the parameter name in the database).

As mentioned above there are three principal outcomes of incremental search: success ($|R| = 1$), UP ($R = \emptyset$) and OP ($|R| > 1$) (see Figure 1).

1 This suggest that TDM only have direct support for incremental search for *system* questions, but note that the task addressed by FiDo could, independent of TDM, be conceived as an incremental searching for the answer to a *user* question, namely *what dog should I [= the user] get?*.

Figure 1. Outcomes of incremental search



In case of the success outcome, this will invoke the service action `SuggestDog` and the system will utter an end report for that action, having the predicate `what_dog_to_get` as a variable. For example, in an incremental search resulting in $R = \{\text{Rottweiler}\}$, the system will say: “You should get a Rottweiler”. The next sections describe how the domain has been defined to handle UP and OP outcomes.

3.3.3. Two variants of $|R| > 1$: MOP and POP

When $|R| > 1$, the system will ask its question for the predicate `what_dog_to_get` (defined in the grammar) followed a listing of the set R . Moreover, this behavior can be conditioned by the attribute `max_spoken_alts` of `<parameters>`, which defines the maximum size of R that is read out. In FiDo, `max_spoken_alts="5"`. Thus, we can distinguish two types of OP outcomes, treated differently in FiDo:

- i. *Manageable* OP (MOP), where $1 < |R| \leq \text{max_spoken_alts}$
- ii. *Problematic* OP (POP), where $|R| > \text{max_spoken_alts}$

3.3.4. MOP outcomes ($1 < |R| \leq \text{max_spoken_alts}$)

If $1 < |R| \leq \text{max_spoken_alts}$, the system will produce the system question for `what_dog_to_get`, which in FiDo is “There is more than one dog matching those preferences” (i.e., not a question in the ordinary sense) followed by a list of the dogs. To move on in this situation, FiDo supports additional information seeking by the user to eventually find a candidate dog to get. Four sub-goals are defined to enable this information seeking:

- i. *describe dog*: the system describes key characteristics of a dog
- ii. *compare dogs*: the system provides a comparison of a target dog and another dog with regard to some trait
- iii. *tell most dog*: the system tells which dog (or dogs) of the remaining ones in R that has (have) the highest value for some trait

- iv. *tell least dog*: the system tells which dog (or dogs) of the remaining ones in R that has (have) the lowest value for some trait

These sub-goals are implemented as resolve goals for the predicates `describe_dog`, `compare`, `most_dog` and `least_dog`. Each such goal has a plan consisting of:

- i. one or several findouts (e.g., what dog to describe in the sub-goal *describe dog*),
- ii. invocation of a service query (for the predicate of the goal), and
- iii. `<forget>` elements for each predicate found out, so that the goal can be re-activated with new variables (e.g., a description of another dog).

3.3.5. POP outcomes ($|R| > \text{max_spoken_alts}$)

There is an optional attribute of `<parameters>`, `on_too_many_hits_action`, which defines an action to which the system should transition in cases of POP outcomes of incremental search. In FiDo, an action for `on_too_many_hits_action` has been defined, `solve_overflow`, which enacts the “minimal” procedure of, first, *informing* the user that there are too many dogs and that the process will re-start from the beginning and, second, *forgetting all* predicates and starting from the beginning. However, I have not been able to implement a proper transition to `solve_overflow`. As noted above, the default moves of TDM in cases of OP outcomes of incremental search are (i) an ask move for the predicate, and (ii) a set of propositions construed from the predicate and each elements of R . In cases of MOP outcomes, there is a default utterance for those moves, as described above. In cases of POP, the utterance for these moves is an empty utterance. When running the Web application of FiDo, this causes the problem that the system is stuck in this empty turn. I have not found a solution for this. When interacting directly with TDM, i.e. using the `http://tdm/interact` mode, there is a work around, as the system will not be stuck in a turn, but handle over the turn to the user. At hits point, saying anything to the system will transition to `solve_overflow`.

3.3.6. UP outcomes

Parallel to the attribute `on_too_many_hits_action`, there is an attribute `on_zero_hits_action` for specifying an action transition in the case of UP outcomes. FiDo has an action `solve_zero` for `on_zero_hits_action`.² However, this transition is shadowed by another strategy in FiDo, namely the use of `<validator>`s for the parameters of the incremental search (see sect 3.5). Through this anticipatory approach UP outcomes are avoided in the first place.

3.4. Ontology

FiDo uses four sorts: `dog` (dynamic), `p_score` (i.e. preference score), `feature`, and `explanation`; and 15 pre-defined individuals: one for each preference score ranging from 0 to 5, one for each feature (*trainability* etc.) which are used in comparisons of dogs (see sect. 3.3.4), and three explanations that are used with `<inform>` elements, for example,

² Like `solve_overflow`, `solve_zero` is only informs the user and then re-start from the beginning.

explaining the procedure to the user. In addition, the ontology defines a number of actions and predicates required by the definition of the domain. See code for details.

3.5. Service interface

As already mentioned, FiDo uses a service action (`SuggestDog`) and several service queries: one for the incremental search and four for describing and comparing dogs (see sect. 3.3.4). In addition, the `service interface` contains five `<validator>`s, one for each of the parameters of the incremental search. Each validator checks if the user's preference score for a parameter results in $R = \emptyset$, if used as the next step of the incremental search. If the preference score would result in an empty set, a warning to the user is triggered, a `<validity>` in the grammar, saying that there is not dog satisfying the provided score, enabling the user to providing another one.

3.6. Grammar

The grammar defines how the FiDo asks questions, reports ended actions, explains situations, and warns about invalid parameters. Moreover, it defined how user utterances are to be interpreted as questions, answers, requests for action and reference to individuals. See code for details. The grammar is input to data for training a RASA model (see sect. 3.8).

3.7. HTTP service

The functions for handling requests for the service action `SuggestDog`, service queries, validators and help functions are defined in the `http service`. Here, these cannot be described in detail (see code for that). However, five general comments are in place.

- The global variable `db_mode_is_API` determines the overall mode of the `http service` as *local* or *API* (see sect. 3.2.4). For *API mode* set `db_mode_is_API` to `True`; for *local mode* set `db_mode_is_API` to `False`. The help function `go_lden_retriever` handles various types of queries to the database (*API* or *local*) and returns a response in the defined format, for example, a set of dogs or a database entry for a particular dog.
- `filter_dogs` identifies R based on a list of trait–score pairs.
- `interpretation` interprets a preference score of a feature in accordance with one of three approaches *strict*, *acceptable level*, or *graded bipolar* (see sect. 3.2.2).
- The functions `dog_describer`, `dog_comparator`, `most_dog_finder`, and `least_dog_finder` defines the targets of the service queries for dog descriptions and comparisons. They behave similarly by (i) identifying the parameters for their associated query, (ii) use those parameters to identify the relevant information from the database, (iii) re-packing that information as a (grammatical) string and (iv) returning that string to TDM. Note that `most_dog_finder`, and `least_dog_finder` uses R as the domain of the comparison. R is rebuilt from the facts of the payload.

- checker is a function shared by all five validators. For a parameter F and a value s , checker checks if R (filtered by previous parameters) and the set of dogs satisfying $F(s)$ is the empty set. If so, it returns False, otherwise True.

3.8. NLU model trained with RASA

Data for training an NLU model was generated using the parameter lookup-entries, more precisely all the dogs listed on Wikipedia's entry for *List of dog breeds* (https://en.wikipedia.org/wiki/List_of_dog_breeds) saved as an csv file (dogs.csv). The model was trained as instructed in the course LT2319 (ds2-training-syncing-and-testing.md).

4. FiDo in action

4.1. Overview

Table 2 gives a summary of dialogue types handled by FiDo. POP outcome are not managed in pipeline mode (incl. the web application), but partly managed in tdm mode (see sect. 3.3.5). There are problems with comparisons in pipeline mode. The problem results from some aspect of using the RASA model for NLU, but is not more precisely diagnosed at the moment.

Table 2. Dialogue types managed by FiDo

Dialogue type	tdm	pipeline
Success, along the way (asking q_1-q_j , where $j < 5$)	✓	✓
Success, all the way (asking q_1-q_5)	✓	✓
MOP Comparison	✓	✗
MOP Description	✓	✓
MOP Tell least	✓	✓
MOP Tell most	✓	✓
System induced answer-revision (anticipating and avoiding UP outcomes)	✓	✓
POP (minimal solution)	(✓)	(✗)

4.2. Examples

4.2.1. Successful outcomes

FiDo supports success of incremental search. Note that as soon as $|R| = 1$, the incremental search will stop and remaining questions in q_1-q_5 will be ignored; FiDo will produce a suggestion of the single dog in R . Below is an example of successful incremental search (in local database mode), where all five questions are asked before $|R| = 1$.

- (6) S> What would you like to do?
 U> I want help with deciding which dog to get
 S> I would be happy to help you! I will ask you a couple of questions to figure out what dog that might fit your preferences. Please answer from a scale from minus three to plus three, but not zero. First question. To what degree should your dog be trainable?
 U> three

S> To what degree should your dog shed?
 U> one
 S> To what degree should your dog have energy?
 U> one
 S> To what degree should your dog bark?
 U> minus two
 S> To what degree should your dog be protective?
 U> three
 S> You should get a Rottweiler

4.2.2. MOP outcomes

In case of MOP outcomes, FiDo can describe and compare dogs for the user. Comparisons can be binary, i.e., comparing a target dog with another dog with regard to some trait, or it can identify the dog among R which has the highest or lowest value for some trait. The example below illustrates all four functions.

(7) S> What would you like to do?
 U> I want help with deciding which dog to get
 S> I would be happy to help you! I will ask you a couple of questions to figure out what dog that might fit your preferences. Please answer from a scale from minus three to plus three, but not zero. First question. To what degree should your dog be trainable?
 U> two
 S> To what degree should your dog shed?
 U> one
 S> To what degree should your dog have energy?
 U> one
 S> To what degree should your dog bark?
 U> minus two
 S> To what degree should your dog be protective?
 U> three
 S> There is more than one dog matching those preferences : Bullmastiff, Hovawart or Rottweiler?
 U> Can you help me compare the dogs
 S> What dogs do you want to compare?
 U> Hovawart and Rottweiler
 S> With regard to what feature do you want to compare those dogs?
 U> trainability
 S> Hovawart is harder to train than Rottweiler.
 U> Of these which bark least
 S> Those dogs are the same with regard to barking.
 U> Tell me more about Rottweiler
 S> Rottweiler is a dog which is very easy to train. It sheds to some extent. It is quite high in energy and barks very little. It is very protective. It is good with children.
 U> Of these which has most energy
 S> The dog with most energy is Bullmastiff.
 U>
 S> Returning to Find dog. There is more than one dog matching those preferences : Bullmastiff, Hovawart or Rottweiler?
 U> I think I will get a Bullmastiff then
 S> You should get a Bullmastiff

4.2.3. POP outcomes

Presently, FiDo is underdeveloped with regard to POP outcomes. First, there is a problem with transition to the `on_too_many_hits_action` (see sect 3.3.5). When run as a web application, FiDo will be stuck in an empty turn. Second, the action defined for `on_too_many_hits_action`, i.e. `solve_overflow`, is only minimally helpful to the user since it does not pursue the task of finding a dog, but rather gives up and informs the user of that decision. Below is an example of FiDo run in `http://tdm/interact` mode (and local database mode). Note that a user turn – in (8) “what is next” – is required for FiDo to transition to `solve_overflow`.

```
(8) S> What would you like to do?
    U> I want help with deciding which dog to get
    S> I would be happy to help you! I will ask you a couple of questions to figure out what
        dog that might fit your preferences. Please answer from a scale from minus three to plus
        three, but not zero. First question. To what degree should your dog be trainable?
    U> one
    S> To what degree should your dog shed?
    U> minus one
    S> To what degree should your dog have energy?
    U> one
    S> To what degree should your dog bark?
    U> one
    S> To what degree should your dog be protective?
    U> three
    S>
    U> what is next
    S> I heard you say what is next. I don't understand. Oops! There are too many dogs
        matching that profile. Let us start from the beginning.
```

4.2.4. Avoiding UP outcomes: System induced answer-revision

FiDo uses validators to avoid UP outcomes. As such, FiDo supports a type of answer-revision (Larsson 2016), namely system induced answer-revision, where the user revises an answer after the system has informed her that there is a problem with the original answer. The behavior is exemplified below (local database mode):

```
(9) S> What would you like to do?
    U> I want help with deciding which dog to get
    S> I would be happy to help you! I will ask you a couple of questions to figure out what
        dog that might fit your preferences. Please answer from a scale from minus three to plus
        three, but not zero. First question. To what degree should your dog be trainable?
    U> minus one
    S> To what degree should your dog shed?
    U> minus one
    S> To what degree should your dog have energy?
    U> minus one
    S> Given your previous preferences, there is no dog that fits that value for energy. To what
        degree should your dog have energy?
    U> one
    S> To what degree should your dog bark?
    U> one
```

S> You should get a Afghan Hound

5. Discussion and future work

FiDo has several limitations. Below these are discussed.

5.1. A better understanding of preference probing

FiDo would clearly benefit from a better understanding of preferences and the semantics of questions about them. The present implementation is unsatisfactory considering both psychology and algorithm. Experiments and consultations with participants after dialogue (“to question so-and-so your answer was this-and-that; hat did you mean?”) are possible methods for future research.

5.2. Remembering R

Whatever knowledge that is gained from further study of preferences, some form of preference-as-range is likely to be part of a system like FiDo. As noted above, such approach to preferences causes a problem when combined with an API not permitting requests with multiple values for a single parameter and TDM’s present implementation of incremental search. In most cases, there is little hope to affect the interface of the API, but there is a possibility to improve this situation from TDM. In present TDM, at each step of incremental search, the values of parameters of previous steps is known from the request (payload), but the result (response) from previous steps is not stored. This means that although the result of previous steps can be re-built at every step of the process, it has to be built anew. Given the need for multiple API requests related to each parameter and an increasing set of parameters at each step, the time for the process will eventually exceed the permitted timeout of TDM’s service queries. A possibility for avoiding this growing number API requests would be to remember the result of previous steps, i.e. R , thereby escaping the need to rebuild the result from previous steps.

5.3. A better approach for POP outcomes

FiDo is underdeveloped with regard to POP outcomes. Both conceptually, i.e., what action plan should be used for proper management of large numbers of $|R|$; and technically, i.e. how to transition to the action defined. Conceptually we might consider several options to manage POP outcomes. For instance, the system could initiate a process to “provoke” the user to revise her preferences, so that success or MOP outcomes are attained. Note that POP outcomes are most likely to result from values in the mid range of the normal distribution. More extreme values would filter out a smaller set of dogs. Another option could be to initiate a new incremental using additional traits. Considering the technical issue, I have no idea at the moment how to solve the of failed transitions.

5.4. What predicates?

An improved version of FiDo should re-consider what traits to be addressed in q_1 etc. For example, taking the survey *Dog Breed Selector* (at <https://dogbreedo.com/dog-breed-selector/>), one is asked “What describes your home the best?”, with alternatives for answers being: “Apartment”, “House with small outdoor space”, “House with big yard”.

Rather than identifying traits of dogs directly, users are here asked to consider aspects of their everyday life. This form of indirect probing for preferences might be a more natural way to search for possible dogs, than the direct and admittedly odd questions implemented in FiDo. I guess a dog expert given the task to find a dog for a client is unlikely to start asking questions like the ones raised by FiDo (“one a scale from minus three to ...”). Perhaps, targeting ways of living and personality traits of the user is a more natural approach to find information to infer a candidate dog. However, a computational implementation of the indirect (inferential) approach requires a clear idea of how to link questions to parameters of the database.

Another aspect of what predicates that should guide preference probing in FiDo was observed in phase 1 dialogues. Consider

- (10) S: what can I help you with
U: I would like to buy a dog
S: okay / have you thought about which dog you want
U: no / but I have been thinking about certain qualities
S: aha / okay / what do you think is important for a dog then
U: it is important that it is good with children
S: okay
...

In (10), *R* is not restricted by a predefined predicates. Rather, they are identified by an open search for preferred predicates. An interesting development of FiDo would be to implement something similar. However, how to do that in TDM is far from clear since the predicates of incremental search must be predefined. Perhaps some strategy using `<bind>` elements for every possible parameter of the API, rather than predefined `<ask_feature>`s of `<parameters incremental="true">`, could be a path worth pursuing.

5.5. Guidance to the user in case of MOP outcomes

As it stands, FiDo does not give any guidance to the user for how to proceed in cases of a MOP outcome. Consider the following passage from the observed dialogues (distilled):

- (11) ...
S: okay. then there are three dogs that fit that profile. a Bergamasco sheepdog. an irish water spaniel. and a briard
U: yes
S: would you like to know anything more about these dogs
U: yes
S: like for example
U: size
S: I'm having a bit of a hard time getting that information here right now. but I can offer you the following. I can describe any of these dogs in detail. I can compare them in some respect.
U: mm
S: and I can also find the dog that has the most or least of any of these characteristics
U: well. eh. but this last briard I would like to know a little more about

S: okay. briard is a dog ...

Like S in this example, FiDo should give some guidance to the user of on how they can proceed (describe and compare dogs); presently, FiDo does not. An `<inform>` element could in principle give the desired information, but I have not figured where to place such element in the plan to achieve the desired outcome.

5.6. Is incremental search really the best approach for this problem?

A final issue to be raised in this report is if incremental search really is the best conceptualization for the problem at hand. The clearest motivation for incremental search is a search problem which aims to find the “shortest path” to a unique individual in a database. A typical case is a service to find (and contact) a person or organization (see demo for incremental search in the TDM documentation). As long as the system cannot uniquely identify a contact, the system raise a new question in the “stack” q_1, \dots, q_n . As soon as a unique contact is identified the process stops; there is no need to ask more questions. I am not sure that assisting a client to decide which dog to get really is best conceived as this type of problem. An alternative approach would be to, first, identify preferences of the user which could be represented as a multidimensional data point. Second, that data point is mapped to the closest dog (data point) in multidimensional space of dogs. While such approach would simplify the semantics of preference probing above, it is only compatible with an API which, unlike API-Ninjas’ Dog API, supports a multidimensional mapping. With an API like API-Ninjas’ Dog API, multidimensional mapping would have similar problems as FiDo, with excessive API requests.

6. Code and data

The github repository for this project is found here:
https://github.com/mboholm/lt2319_proj.

7. References

- Bocklisch, T., Faulkner, J., Pawlowski, N., & Nichol, A. (2017). Rasa: Open source language understanding and dialogue management. NIPS 2017 Conversational AI workshop, Long Beach, CA, USA. arXiv:1712.05181.
- Larsson, S. (2002). *Issue-based dialogue management*. Department of Linguistics, Göteborg University.
- Larsson, S. (2015). The State of the Art in Dealing with User Answers. *SEMDIAL 2015 goDIAL*.
- Larsson, S. and Berman, A. (2016). Domain-specific and General Syntax and Semantics in the Talkamatic Dialogue Manager. *Empirical Issues in Syntax and Semantics* 11.
- Larsson, S., Santamarta, L., and Jönsson, A. (2000). Using the process of distilling dialogues to understand dialogue systems. *Proceedings of ICSLP 2000*.