Conversational Recommendations Using Model-based Reasoning

Oliver A. Tazl and **Alexander Perko** and **Franz Wotawa**[1]

Abstract. Chatbots as conversational recommender have gained increasing importance over the years. The chatbot market offers a variety of applications for research and industry alike. In this paper, we discuss an implementation that supports the use of our recommendation algorithm during chatbot communication. The program eases communication and improves the underlying recommendation flow. In particular, the implementation makes use of our model-based reasoning approach for improving user experience during a chat, i.e., in cases where user configurations cause inconsistencies. The approach deals with such issues by removing inconsistencies in order to generate a valid recommendation. In addition to the underlying definitions, we demonstrate our implementation along use cases from the tourism domain.

1 INTRODUCTION

Recommender systems aim to lead users in a helpful and individualized way to interesting or useful items drawn from a large space of possible options. Recommender systems may utilize knowledge-bases for guiding the users through the whole process of finding the right recommendation, i.e., a recommendation that satisfies the user's requirements, needs, or expectations. Most recently, conversational agents like chatbots have gained importance because of the fact that they – in principle – offer a well-known and ideally more intuitive interface for human users, i.e., either textual or speech interaction.

In previous work [17] we introduced the basic foundations and principles behind a chatbot-based recommender system that allows to interact with users in a smart way being capable of finding contradictions during observation and efficiently pruning the overall recommendation process via selecting the right questions to be asked to the user. The basic principles behind our approach rely on classical model-based reasoning. In case, the conversation leads to an inconsistent state, e.g., caused by contradictions between user requirements and the recommendation knowledge-base, the chatbot is able to react and to resolve this issue. For this purpose, the chatbot asks the user which requirements to retract in order to eliminate inconsistencies. In the case where the chatbot has far to many solutions to be presented effectively to the user, the system makes use of an entropy-based approach for selecting those requirements or attributes that have to be fixed in order to reduce the number of possible solutions. When using entropy the number of steps necessary to reach to a solution can be substantially reduced.

This paper is a direct successor of our previous work, where we report on an implementation of our chatbot approach. In particular, we discuss the implementation details, present experiences gained, and finally introduce the results of an evaluation of the implementation. The evaluation is based on a case study from the tourism domain, i.e., a scenario where a user wants to book a hotel in a certain city. The obtained results show that the proposed chatbot approach is applicable and beneficial for the intended purpose. Furthermore, we gained experiences about the limitations of the approach. For example, it seems that entropy is not always the best measure for selecting questions to be answered, and further research is needed.

The main contributions of this paper can be summarized as the follows:

1. An implementation of an algorithm that is based on model-based diagnosis and Shannon's information entropy to solve recommendation problems and
2. the evaluation of the system with synthetic and real-world data sets.

The remainder of this paper is organized as follows: In the next section we give an overview of our algorithmic approach. Afterwards, we present the implementation of the algorithms and show evaluation results in greater detail. Finally, we discuss related research and conclude the paper.

2 FOUNDATIONS AND ALGORITHM

In our previous work [17], we introduced the algorithm EntRecom, which utilizes model-based diagnosis and in particular the ConDiag algorithm [18], and on a method that applies Shannon's information entropy [23]. To be self-contained, we briefly recapitulate the underlying definitions and EntRecom. We first formalize the **inconsistent requirements problem** by exploiting the concepts of Model-Based Diagnosis (MBD) [1, 20] and constraint solving [2].

The inconsistent requirements problem requires information on the item catalog (i.e., the knowledge-base of the recommendation system) and the current customer's requirements. Note that the knowledge-base of the recommender may be consistent with the customer's requirements (i.e., the customer's query) and an appropriate number of recommendations can be offered. In this case, the recommendation system shows the recommendations to the customer and no further algorithms have to be applied. Otherwise, if no solutions to the recommendation problem are available, then the minimal set of requirements, which determined the inconsistency with the knowledge base, have to be identified and consequently offered to the user as explanation for not finding any recommendation. The user can in this case adapt the requirement(s) (relax it/them). Here we borrow the idea from MBD and introduce abnormal modes for the given requirements, i.e., we use Ab predicates stating whether a requirement i is should be assumed valid ($\neg Ab_i$) or not (Ab_i) in a particular context.

[1] Graz University of Technology, Austria, email: {oliver.tazl, perko, wotawa}@ist.tugraz.at

The Ab values for the requirements are set by the model-based diagnosis algorithm so that the assumptions together with the requirements and the knowledge-base are consistent. In the following, we define the inconsistent requirements problem and its solutions.

More formally, we stating the inconsistent requirements problem as follows:

Definition 1 (Inconsistent Requirements Problem)**.** *Given a tuple (KB, REQ) where KB denotes the knowledge base of the recommender system, i.e., the item catalog, and REQ denotes the customer requirements. The Inconsistent Requirements Problem arises when KB together with REQ is inconsistent. In this case we are interested in identifying those requirements that are responsible for the inconsistency.*

A solution or explanation to the inconsistent requirements problem can be easily formalized using the analogy with the definition of diagnosis from Reiter [20]. We first introduce a modified representation of (KB, REQ) comprising (KB_D, REQ) where KB_D comprises KB together with rules of the form Ab_R for each requirement R in REQ. The solution to the Inconsistent Requirements Problem can now be defined using the modified representation as follows:

Definition 2 (Inconsistent Requirements)**.** *Given a modified recommendation model (KB_D, REQ). A subset $\Gamma \subseteq REQ$ is a valid set of inconsistent requirements iff $KB_D \cup \{\neg Ab_R | R \in REQ \setminus \Gamma\} \cup \{Ab_R | R \in \Gamma\}$ is satisfiable.*

A set of inconsistent requirements Γ is minimal iff no other set of inconsistent requirements $\Gamma' \subset \Gamma$ exists. A set of inconsistent requirements Γ is minimal with respect to cardinality iff no other set of inconsistent requirements Γ' with $|\Gamma'| < |\Gamma|$ exists. From here on we assume minimal cardinality sets when using the term minimal sets.

The second problem occurring during a recommendation session is the availability of too large number of recommendations, which have to be narrowed down to a reasonable number. The **too many recommendation problem** can be solved again using ideas borrowed from model-based diagnosis. In diagnosis, we have the similar problem of coming up with too many diagnoses because of too less observations known. The corresponding problem in case of recommendation is that we do have far too less requirements from the user. Hence, we have to ask the user about adding more information in order to reduce the number of available solutions. In model-based diagnosis Shannon's information entropy [23] is used to come up with observations and in our case requirements that should be known in order to reduce the recommendations as fast as possible.

Algorithm 1 provides recommendations in the context of chatbots making use of diagnosis and Shannon's information entropy computation. EntRecom converts the available knowledge into a corresponding constraint model and checks its consistency. If the knowledge is inconsistent, the algorithm tries to find a requirements that can be retracted by the user in order to get rid of the inconsistency. Afterwards, EntRecom searches for the best requirement to be set by the user in order to reduce the number of solutions if necessary. The algorithm stops when reaching a set of recommendations that has a cardinality of less than n.

With the provide algorithms a chatbot for recommendations can be build that is able to deal with inconsistent requirements as well as missing requirements in a more or less straightforward way making use of previously invented algorithms.

Algorithm 1 EntRecom(KB_D, REQ, n)

Input: A modified knowledge base KB_D, a set of customer requirements REQ and the maximum number of recommendations n
Output: All recommendations S

```
1: Generate the constraint model $CM$ from $KB_D$ and $REQ$
2: Call **CSolver**$(CM)$ to check consistency and store the result in $S$
3: **if** $S = \emptyset$ **then**
4:     Call **MI_REQ**$(CM, |REQ|)$ and store the inconsistent requirements in $IncReqs$
5:     Call **askUser(IncReqs)** and store the answer in $AdaptedReqs$
6:     $CM = KB \cup (REQ \setminus IncReqs \cup AdaptedReqs)$
7:     **go to** Step 2
8: **end if**
9: **while** $|S| > n$ **do**
10:    Call **GetBestEntrAttr**$(A_S)$ and store the result in $a$
11:    $A_S = A_S \setminus a$
12:    Call **askUser(a)** and store the answer in $v_a$
13:    $S = \mathcal{R}(S, v_a))$
14: **end while**
15: **return** $S$
```

3 IMPLEMENTATION AND EVALUATION

For the user, the interaction with the recommender system within the chatbot starts, when he or she formulates a query for a search within the domain. The natural language processing framework Rasa[2] parses this query and passes it on to the EntRecom-algorithm. The recommender algorithm then searches a previously conducted and preprocessed knowledge base. Depending on the satisfiability of the generated constraint model, results are presented to the user. If necessary the user is asked follow up questions regarding his or her requirements until we have results to return to the user.

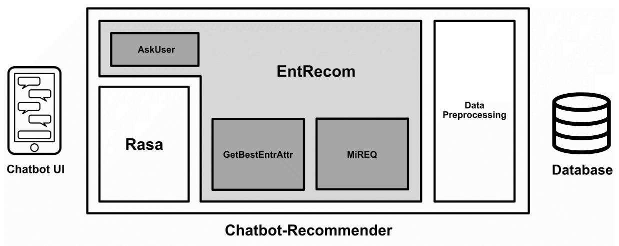


Figure 1: A Chatbot-Based Recommender as Bridge between User Input and Results from a Database

When the query can be answered, EntRecom exits and the user can continue interacting with the Rasa-based chatbot.

As mentioned before, we choose a tourism domain, searching for a hotel to be more specific. The process of searching a hotel in an iterative conversation helps us to see the capabilities and shortcomings of the algorithm.

[2] see https://rasa.com/

3.1 Framework and Data Preprocessing

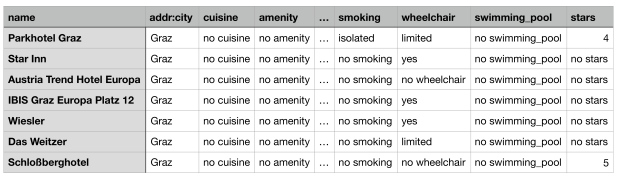
For our tests, we chose the community-curated data from Open Street Maps (OSM). We use the python bindings for Microsofts' Z3 framework as a constraint solver. For language processing and basic interaction with the user, we utilize the Rasa framework. Rasa is used for natural language understanding by extrating the users' intend and requirements but can also respond immediatly with an answer (without calling EntRecom) when it detects the users' intend to chitchat.

In a first step, we query the OSM-API for hotels in a predefined region. The size of the region is a major factor for performance. Because of this, it is reasonable to let the user select a region on initialization.

```
{
  "name": "Pension Kohlmayr",
  "tourism": "hotel"
},
{
  "fixme": "exact location",
  "name": "Hotel am M\u00fchlengrund",
  "tourism": "hotel"
},
{
  "addr:city": "Graz",
  "addr:country": "AT",
  "addr:housenumber": "89",
  "addr:postcode": "8020",
  "addr:street": "Bahnhofg\u00fcrtel",
  "name": "Austria Trend Hotel Europa",
  "phone": "+43 316 7076-601",
  "tourism": "hotel"
},
```

Figure 2: Data from Open Street Maps

To conduct our internal knowledge base, we process the exported data set in the following way: As some attributes do not add human-readable information or tend to mislead users, we have to filter them out. This is especially necessary because some classes of attributes have a severe impact on later steps in our recommender system. For example, categories like ”fixme” (annotation from an OSM user) would give the user no advantage in a real-world scenario and in this case, would lead to unwanted recommendations. We use a whitelist-filter for the attributes of every entry in the data set. Furthermore, not every entry in the OSM data has the same fields, why we maintain a list of all attributes in a data set. If an entry does not include a certain attribute, we add it with a value of False. This leaves us with a ”normalized” data set.



name	addr:city	cuisine	amenity	...	smoking	wheelchair	swimming_pool	stars
Parkhotel Graz	Graz	no cuisine	no amenity	...	isolated	limited	no swimming_pool	4
Star Inn	Graz	no cuisine	no amenity	...	no smoking	yes	no swimming_pool	no stars
Austria Trend Hotel Europa	Graz	no cuisine	no amenity	...	no smoking	no wheelchair	no swimming_pool	no stars
IBIS Graz Europa Platz 12	Graz	no cuisine	no amenity	...	no smoking	yes	no swimming_pool	no stars
Wiesler	Graz	no cuisine	no amenity	...	no smoking	yes	no swimming_pool	no stars
Das Weitzer	Graz	no cuisine	no amenity	...	no smoking	limited	no swimming_pool	no stars
Schloßberghotel	Graz	no cuisine	no amenity	...	no smoking	no wheelchair	no swimming_pool	5

Figure 3: Preprocessed Data Set

After our preprocessing step, every entry in the dictionary has the same number of attributes we add uniform clauses to our constraint model. An exemplary normalized clause looks like this: $(amenity = "none" \land addr : city = "Graz" \land name = "ParkhotelGraz" \land cuisine = "none" \land \ldots \land smoking = "isolated" \land wheelchair = "limited" \land swimming-pool = "none" \land stars = "4" \land tourism = "hotel")$. Regarding data types, we choose between two different approaches, when creating the Z3 constraint model. The first one being data type selection for every attribute in the domain and the second one being string translation for every value, regardless of the specific class of an attribute. In our implementation, every attribute and value is translated to a Z3 string. While this may result in slower runtimes, we ensure flexibility, as data types may vary across attributes. Usually, the user is asked to select a region of interest. For our tests, we use a data set of a specific size. This is a very critical and, depending of the size of the test set, time consuming step. Therefore, we try to do it only once when the user initially specifies an area. This Z3 constraint model, consisting of OR-connected uniform clauses of Z3 strings is our main knwoledge base within $EntRecom$ and shall be refered to as kb and subsets therof as S in the remainder of this paper.

The first interaction the user has with our chatbot implementation is handled via the trained natural language understanding (NLU) model within Rasa. If the user's intent to search within the domain ”hotels”, our recommender is called via a webhook, and the parameters are passed over. This call to our internal API is the only interaction between EntRecom and the chatbot framework, Rasa, in our case, which leads to very low coupling and a high degree of flexibility. The parameters represent the NLUs interpretation of the users' query and give us our initial set of requirements.

3.2 Recommender Algorithm

In this section, we are describing the implementation of the previously introduced algorithm EntRecom.

Before the algorithm ready to use, we have to prepare our data set, generate the constraint model and interpret the user's intent. Then we enter the EntRecom-implementation. As described before, if the query is satisfiable for our knowledge base and the given maximum number of results, we return our recommendations and exit EntRecom.A maximum of n hotels is presented to the user, we retrieved from the knowledge base, with n representing a preselected maximum number of results.

Though, as stated in [17], we are confronted with two potential problems at this point. Given a knowledge base kb, a maximum number of results n and a user-defined set of requirements REQ:

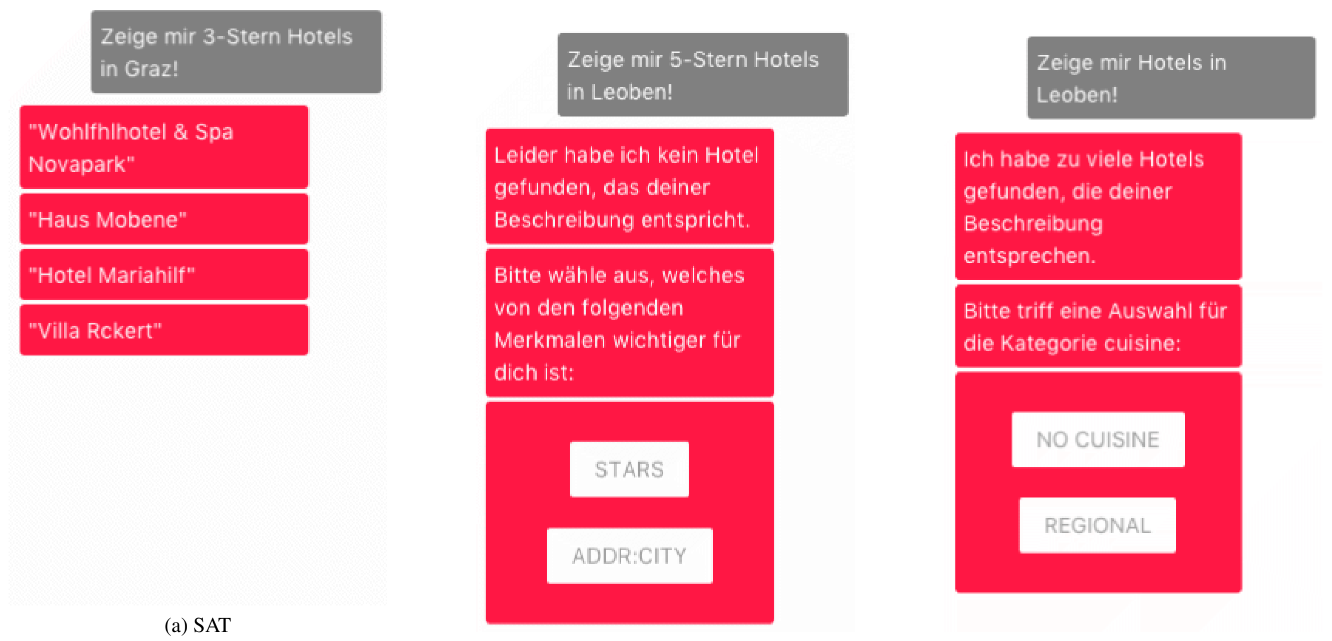
- We could get too many results to present in a meaningful way. In this case, the function $GetBestEntrAttr$ is called.
- We could get no results at all. In this scenario, we call the function $MiREQ$.

3.2.1 $GetBestEntrAttr$

This part of the algorithm is called, when the query is satisfiable, but the result does not lie within [n]. Therefore, we have to add further constraints to our model. Because we want to occupy as little of the user's time as possible, we have to efficiently select additional constraints. This is done by choosing a category out of the domain which best splits the current subset S of the data set. The criterion for our selection is Shannon's information entropy [22]:

$$H(X) = -\sum_{i} P(x_i) log(P(x_i))$$

To apply entropy as a measure, we have to restructure the data slightly. A_S represents this restructured version of S, which maps every attribute in the domain to all values of its occurrences in S. This is realized with python dictionaries of the form: $"attribute_1" : ["value_1", "value_2"], "attribute_2" : ["value_1"], \ldots$. After computing the number of occurrences of every value for a


Zeige mir 3-Stern Hotels in Graz!

"Wohlfhlhotel & Spa Novapark"

"Haus Mobene"

"Hotel Mariahilf"

"Villa Rckert"

Zeige mir 5-Stern Hotels in Leoben!

Leider habe ich kein Hotel gefunden, das deiner Beschreibung entspricht.

Bitte wähle aus, welches von den folgenden Merkmalen wichtiger für dich ist:

Zeige mir Hotels in Leoben!

Ich habe zu viele Hotels gefunden, die deiner Beschreibung entsprechen.

Bitte triff eine Auswahl für die Kategorie cuisine:

(a) SAT (b) MiREQ (c) GetBestEntrAttr

Figure 4: User Interface

specific attribute, we now calculate the entropy, for every attribute. The attribute with the highest entropy splits the data set S most effectively. As mentioned above, some attributes may lead to unwanted recommendations. One reason for this is the appearance of seemingly random values when looked at them lacking context. This could be attributes internally used within OSM, for instance. Fields like ”source” often have seemingly random values like ”survey” or ”Kleine Zeitung”. The same pseudo-randomness can be observed with attributes like ”housenumber” which can take an arbitrary integer value. Those values appear in many data points and, when observed isolated, do not contribute any information with regard to splitting the data set. When computing the information entropy based on attributes lacking spatial ordering or clustering properties, we are not dividing the data set strategically but randomly. This is why we chose to exclude them from our knowledge base beforehand. If several attributes occur with equal entropies in the data set, we can randomly select one of these categories, as all of them split the data equally well. In the last step, the user is asked for selecting one value for this category. The user's selection is added to the set of requirements and $EntRecom$ gets called again.

3.2.2 $MiREQ$

This part of the algorithm is called, when the query is not satisfiable for our knowledge base KB with the user-defined set of requirements REQ. Following [17], we state the Inconsistent Requirements Problem. As a counter measurement to the Inconsistent Requirement Problem, we have to soften the query to get results. To achieve this, we have to find inconsistencies in REQ, being a subset Γ thereof. $\forall$ inconsistent subsets Γ, $KB \cup \{\neg Ab_R \mid R \in REQ \setminus \Gamma\} \cup \{Ab_R | R \in \Gamma\}$ is satisfiable, with AB being a Boolean variable for selecting and deselecting a requirement R to be considered [18] This, we implemented by checking on models with combinatoric variations of subsets of REQ. This means, that we evaluate a constraint model consisting of our knowledge base and requirements with varying values for AB. Given the cardinality of Γ, we iterate over all possible distributions of AB with $|REQ| - |\Gamma|$ considered requirements. Because we want to preserve as much of the user's initial query as possible, we want to find a minimal set of inconsistent constraints. As proposed in [17], to find such a minimal set of inconsistencies, we have to obtain a constraint model that is satisfiable for the smallest cardinality of Γ possible. For this, we repeat the process of checking on combinatoric variations of subsets, with increasing numbers of assumed inconsistencies, starting with one. This we do until we reach $|REQ|$, in which case there are no consistent requirements. When a satisfiable constraint model is found, we are able to retrieve all inconsistent requirements in the form of all unconsidered requirements. With this subset of REQ, we return from $MiREQ$. After the $MiREQ$-function returns, we ask the user for his or her preference for dropping one of his or her previously defined requirements within the minimal set of inconsistencies. While the current implementations assure us to find minimal sets of inconsistent requirements, in future implementations, we hope to be able to make use of the unsatcore functionality of Z3 for this task. This is expected to significantly improve performance.

3.3 Evaluation

We developed our tests concentrating on the algorithm itself and the data structures needed for execution. The data preprocessing was not in the focus of this test. For our tests with synthetic test data this is especially true, as they use a adapted version of EntRecom without user interaction, as depicted in Figure 5. The goal of our experiments is to show potentials for optimizations of the implementation, as well as proofing the versatility of the algorithm proposed in [17].

In our tests, we used both, synthetic and real-world data. For benchmarking and basic experiments, we mainly used synthesized data, while real-world data, specifically from Open Street Maps is

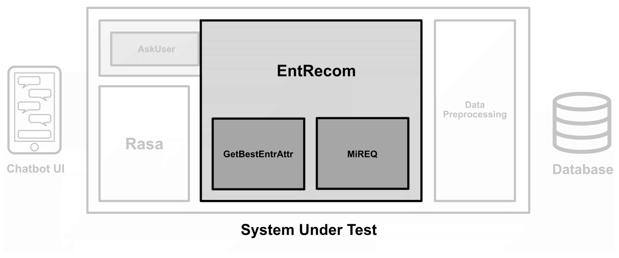


Figure 5: Focus of Tests with Synthetic Data

important to ensure the flexibility and robustness of the implementation in real-world scenarios.

3.3.1 Real-World Test Data

As described before, we use data from Open Street Maps for our tests. First, we want to observe, how the algorithm copes on real-world data and which degree of data preprocessing and filtering of the data is necessary. Furthermore, we developed the rules for generating our synthetic test data based on the exported data from OSM.

For this part of our evaluation, we empirically tested the algorithm for usable results as well as automated tests for exhaustive testing on the data.

3.3.2 Synthetic Test Data

Our modus operandi for generating and conducting tests with synthetic data can be described as follows:

1.) Definition of attribute classes and generation of data sets
2.) Definition of requirement sets and classifying them within a testing matrix
3.) Performing speed tests following the testing matrix

To evaluate our results for the performance test, we order tests within a three-dimensional testing matrix. The first axis of this matrix represents the size of the test set. The second axis represents the number of attributes and the last axis represents the number of allowed results n. The value at every position in the matrix represents the called subfunction within our recommender algorithm.

We randomly generated data sets of different sizes and with different numbers of attributes for our performance tests. Because data within the domains "hotels" and "cars", have several distinct properties, we set up our test data following certain rules. To achieve this, we categorized the attributes dependent on the data type of their corresponding values. Furthermore, we added complementary classes based on certain characteristics of the domain.

This results in the following basic classes of attributes based on their data type:

a) Boolean values
b) Integer values
c) Float values
d) String values

Additionally, we added the following supplementary classes:

e) Often reappearing parts in string values (e.g. names)
f) Frequently recurring string values (e.g. city, manufacturer)
g) Restricted numbers (e.g. star-rating, number of seats)
h) Dates (e.g. registration date)
h) Ranges (e.g. distance-to-the-center, price-category)

ad a) Boolean values appear often in real-world data and are easy to process. This class represents attributes like internet-access or payment-credit-card in the domain tourism and esp or abs in the automotive domain.
ad b) Arbitrary integer values appear often as counter variables like visitors, likes or 5-star-reviews
ad c) Floating-point numbers occur within both domains, tourism, and cars in various forms. This class covers all attributes in the context of distance, like distance-to-the-center for a hotel or mileage for a car. It also stands for location attributes given in coordinates (longitude, latitude) and mean values like user-rating.
ad d) Strings are very versatile and therefore used in many ways in different sources. In many cases, strings contain informational value beyond what a number may cover. But often strings are used in places, where the informative content is not higher than number-valued attributes and could be represented with Boolean values or numbers instead. This is true for several classes of attributes like stars with values of "4-star".
ad e) The name attribute is always a string and does not have to but is likely to include a domain-specific term in its value. For hotels, this would be "Hotel" or a synonym thereof. Because of these recurring terms and the omnipresence over all domain-specific data sources, we treat this attribute separately from the more general string class. When generating data sets, we introduce these terms into our samples regularly.
ad f) Fields like city or manufacturer contain frequently reappearing values, which makes it special regarding information entropy and therefore interesting to us.
ad g) Stars of a hotel can be represented with 1 to 5, which makes a reappearance in this category very probable.
ad h) As we want to simplify the selection process, we reduce the date to the year. This results in an integer value often constrained by an upper boundary being the current year and a case dependent lower boundary. Depending on the specific attribute, these values may recur frequently.
ad i) Ranges are special because they consist not only of one, but two boundary values. For simplification reasons, we categorize ranges within the relative classes: low, med and high. Of course, these range classes are also very likely to reoccur.

Our synthetic data is randomly generated with respect to those classes of attributes and with varying occurrences of the different types.

3.4 Results

The results are splited in two parts. First, we discuss the results of the synthetic data, followed by the proof of concept results with real-world data.

Using the synthetic data, we define sets of requirements to test the algorithm on. These sets belong to one of the following classes relative to the knowledge base e.g. the test data and the given n, they are tested with:

i) Satisfiable with n or less results. This leads to a direct return from the $EntRecom$-algorithm, presenting us the results S the constraint solver found.
ii) Satisfiable with more than n results. In this case, we have to call $GetBestEntrAttr$, which calculates entropies for all attributes. Another interaction with the algorithm is needed, as we have to

select a value for the attribute with the highest informational content.

iii) Not satisfiable. In this case, we have to call $MiREQ$ which iteratively chooses subsets of REQ until it finds the largest satisfiable subset. Its complement is the set of inconsistent requirements δs. Again, another interaction through $AskUser$ is necessary. Now we have to select a requirement out of δs we want to keep in REQ the other constraints are dropped.

To classify the requirement sets for every test set according to the three classes from above, we use an adapted version of $EntRecom$, which does not return results or perform any recursive calls. The purpose of this classification is, to be able to identify test results with regard to the sub-functions called within $EntRecom$. The class of REQ - either i, ii, or iii - for a certain test configuration represents the value of the three-dimensional testing matrix.

After classification, we perform our tests on the generated data. A typical test-scenario starts with the users first interaction with the chatbot environment, which, in turn, results in setting up the knowledge base. Regularly the user is asked to select a region of interest. For our tests, we use a data set of a specific size. Then a set of requirements and a maximum for the expected results are chosen. We now perform a test for every class in the testing matrix and get results in the form of execution times.

For a fixed n of 5 and tests performed with one to five requirements in the sets, we plot the function calls of $MiREQ$ and $GetBestEntrAttr$ on increasingly sized test sets. In case, we obtain a result for our query which lies within n, our constraint solver $CSolver$ is the only function call made.

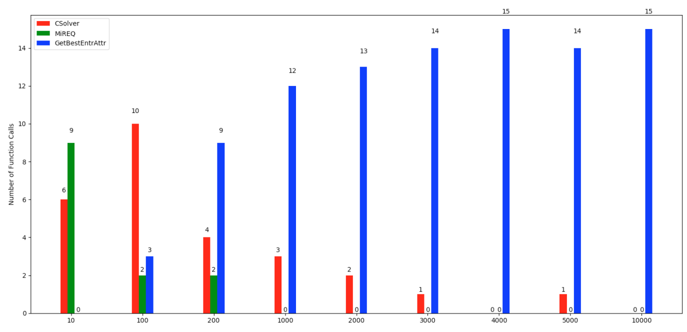


Figure 6: Classification Based on Function Calls within EntRecom

As you see, the function calls of $MiREQ$ decrease, while the calls of $GetBestEntrAttr$ increase for larger test sets. While this basic assumption holds for real data, the impact is not as drastic, as users tend to perform queries with less than five requirements initially.

The algorithm was also used with real-world data. Therefore, we inserted the data from OSM into the knowledge base. We interact with the textual chat-like web interface in the intended way and get the correct results from the algorithm. The algorithm returned a result set within a few iterations and these results fit to the user specification.

These experiments show also that there is a problem of relevance of attributes. The attributes, which have a high entropy to reduce the result set, are not nessesarily relevant for users. Our tests show that i.e. the attribute wheelchair, which is part of accessibility, has a high entropy but will not be interesting for a large amount of users. This issue has to be addressed in an upcoming version of the algorithm.

4 RELATED WORK

The application of model-based reasoning, and especially model-based diagnosis, in the field of recommender systems is not novel. For example, papers like [4, 11, 19] compute the minimal sets of faulty requirements. These requirements should be changed in order to find a solution. In these papers, the authors rely on the existence of minimal conflict sets computing the diagnosis for inconsistent requirements. Felfernig et al. [4] present an algorithm that calculates personalized repairs for inconsistent requirements. The algorithm combines concepts of MBD with a collaborative problem solving approach to improve the quality of repairs in terms of prediction accuracy. In [19], the concept of representative explanations is introduced. This concept follows the idea of generating diversity in alternative diagnoses informally, constraints that occur in conflicts should as well be included in diagnoses presented to the user. Jannach [11] proposes to determine preferred conflicts "on demand" and use a general-purpose and fast conflict detection algorithm for this task, instead of computing all minimal conflicts within the user requirements in advance.

Papers that deal with the integration of diagnosis and constraint solving are [3] and [24, 25], who proposed a diagnosis algorithm for tree-structured models. The approach is generally applicable due to the fact that all general constraint models can be converted into an equivalent tree-structured model using decomposition methods, e.g., hyper tree decomposition [7, 8]. [26] provides more details regarding the coupling of decomposition methods and the diagnosis algorithms for tree-structured models. In addition to that, [21] generalized the algorithms of [3] and [24]. In [15] the authors also propose the use of constraints for diagnosis where conflicts are used to drive the computation. In [6], which is maybe the earliest work that describes the use of constraints for diagnosis, the authors introduce the use of constraints for computing conflicts under the correctness assumptions. For this purpose they developed the concept of constraint propagation. Despite of the fact that all of these algorithms use constraints for modeling, they mainly focus on the integration of constraint solving for conflict generation, which is different to our approach. For presenting recommendation tasks as constraint satisfaction problem, we refer to [12].

Human-chatbot communication represents a broad domain. It covers technical aspects as well as psychological and human perspectives. Contributions like [9, 30] show several ways of implementing chatbots in different domains. Wallace [30] demonstrates an artificial intelligence robot based on a natural language interface (A.L.I.C.E.) that extends ELIZA [31], which is based on an experiment of Alan M. Turing in 1950 [29]. This work describes how to create a robot personality using AIML, an artificial intelligence modelling language, to pretend intelligence and self-awareness.

Sun et al. [27] introduced a conversational recommendation system based on unsupervised learning techniques. The bot was trained by successful order conversations between user and real human agents.

Papers like [5, 10, 13, 32] address the topics user acceptance and experience. In [32] a pre-study shows that users infer the authenticity of a chat agent by two different categories of cues: agent-related cues and conversational-related cues. To get an optimal conversational result, the bot should provide a human-like interaction. Questions of conversational UX design raised by [5] and [16] demonstrate the need to rethink user interaction at all.

The topic of recommender systems with conversational interfaces is shown in [14], where an adaptive recommendation strategy was shown based on reinforcement learning methods. In the paper [28], the authors proposed a deep reinforcment learning framework to build personalized conversational recommendation agents. In this work, a recommendation model trained from conversational sessions and rankings is also presented.

5 CONCLUSION AND FUTURE WORK

In this paper, we showed an implementation and its evaluation of EntRecomm, an algorithm using model-based diagnosis and Shanon's information entropy. In our tests, we used both, synthetic and real-world data. For benchmarking and basic experiments, we mainly used synthesized data, while real-world data, specifically from Open Street Maps is important to ensure the flexibility and robustness of the implementation in real-world scenarios. We also showed the performance for different data sets and also revealed open issues, like the relevance of chosen attributes, which is already a starting point for future work. Another important step will be a user study to evalute the acceptance of the algorithm.

ACKNOWLEDGEMENTS

Research presented in this paper was carried out as part of the AS-IT-IC project that is co-financed by the Cooperation Programme Interreg V-A Slovenia-Austria 2014-2020, European Union, European Regional Development Fund.

REFERENCES

[1] Johan de Kleer and Brian C. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).

[2] Rina Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[3] Yousri El Fattah and Rina Dechter, 'Diagnosing tree-decomposable circuits', in *Proceedings* 14^{th} *International Joint Conf. on Artificial Intelligence*, pp. 1742 – 1748, (1995).

[4] Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan, 'Plausible repairs for inconsistent requirements.', in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 791–796, (01 2009).

[5] Asbjørn Følstad and Petter Bae Brandtzæg, 'Chatbots and the new world of hci', *interactions*, **24**(4), 38–42, (June 2017).

[6] Hector Geffner and Judea Pearl, 'An Improved Constraint-Propagation Algorithm for Diagnosis', in *Proceedings* 10^{th} *International Joint Conf. on Artificial Intelligence*, pp. 1105–1111, (1987).

[7] Georg Gottlob, Nicola Leone, and Francesco Scarcello, 'Hypertree Decomposition and Tractable Queries', in *Proc. 18th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS-99)*, pp. 21–32, Philadelphia, PA, (1999).

[8] Georg Gottlob, Nicola Leone, and Francesco Scarcello, 'A comparison of structural CSP decomposition methods', *Artificial Intelligence*, **124**(2), 243–282, (December 2000).

[9] B. Graf, M. Krüger, F. Müller, A. Ruhland, and A. Zech, 'Nombot - simplify food tracking', volume 30-November-2015, pp. 360–363, (2015).

[10] Jennifer Hill, W. Randolph Ford, and Ingrid G. Farreras, 'Real conversations with artificial intelligence: A comparison between human-human online conversations and human-chatbot conversations', *Computers in Human Behavior*, **49**, 245 – 250, (2015).

[11] Dietmar Jannach, 'Finding preferred query relaxations in content-based recommenders', *IEEE Intelligent Systems*, **109**, 81–97, (04 2008).

[12] Dietmar Jannach, Markus Zanker, and Matthias Fuchs, 'Constraint-based recommendation in tourism: A multiperspective case study', *Journal of IT and Tourism*, **11**, 139–155, (2009).

[13] A. Khanna, M. Jain, T. Kumar, D. Singh, B. Pandey, and V. Jha, 'Anatomy and utilities of an artificial intelligence conversational entity', pp. 594–597, (2016).

[14] Tariq Mahmood, Francesco Ricci, and Adriano Venturini, 'Learning adaptive recommendation strategies for online travel planning', *Information and Communication Technologies in Tourism 2009*, 149–160, (2009).

[15] Jakob Mauss and Martin Sachenbacher, 'Conflict-driven diagnosis using relational aggregations', in *Working Papers of the 10th International Workshop on Principles of Diagnosis (DX-99)*, Loch Awe, Scotland, (1999).

[16] R.J. Moore, R. Arar, G.-J. Ren, and M.H. Szymanski, 'Conversational ux design', volume Part F127655, pp. 492–497, (2017).

[17] Iulia Nica, Oliver A. Tazl, and Franz Wotawa, 'Chatbot-based tourist recommendations using model-based reasoning', in *ConfWS*, (2018).

[18] Iulia Nica and Franz Wotawa, 'Condiag - computing minimal diagnoses using a constraint solver', (2012). International Workshop on Principles of Diagnosis ; Conference date: 31-07-2012 Through 03-08-2012.

[19] Barry O'Sullivan, Alexandre Papadopoulos, Boi Faltings, and Pearl Pu, 'Representative explanations for over-constrained problems', **1**, (07 2007).

[20] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[21] Martin Sachenbacher and Brian C. Williams, 'Diagnosis as semiring-based constraint optimization', in *European Conference on Artificial Intelligence*, pp. 873–877, (2004).

[22] C. E. Shannon, 'A Mathematical Theory of Communication', *The Bell System Technical Journal*, **27**(3), 379–423, (1948).

[23] C. E. Shannon, 'A mathematical theory of communication', *Bell System Technical Journal*, **27**, 379–623, (1948).

[24] Markus Stumptner and Franz Wotawa, 'Diagnosing Tree-Structured Systems', in *Proceedings* 15^{th} *International Joint Conf. on Artificial Intelligence*, Nagoya, Japan, (1997).

[25] Markus Stumptner and Franz Wotawa, 'Diagnosing tree-structured systems', *Artificial Intelligence*, **127**(1), 1–29, (2001).

[26] Markus Stumptner and Franz Wotawa, 'Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems', in *Proceedings of the* 18^{th} *International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 388–393, Acapulco, Mexico, (2003).

[27] Y. Sun, Y. Zhang, Y. Chen, and R. Jin, 'Conversational recommendation system with unsupervised learning', pp. 397–398, (2016).

[28] Yueming Sun and Yi Zhang, 'Conversational recommender system', in *The 41st International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '18, pp. 235–244, New York, NY, USA, (2018). ACM.

[29] Alan M. Turing, *Computing Machinery and Intelligence*, 23–65, Springer Netherlands, Dordrecht, 2009.

[30] R.S. Wallace, *The anatomy of A.L.I.C.E.*, 2009.

[31] J. Weizenbaum, 'Eliza-a computer program for the study of natural language communication between man and machine', *Communications of the ACM*, **9**(1), 36–45, (1966).

[32] N.V. Wünderlich and S. Paluch, 'A nice and friendly chat with a bot: User perceptions of ai-based service agents', (2018).