Conversational Recommendation:
Theoretical Model and Complexity Analysis

Tommaso Di Noia[a], Francesco Donini[b,*], Dietmar Jannach[c], Fedelucio Narducci[a], Claudio Pomo[a]

[a]*Politecnico di Bari, Italy*
[b]*Università degli Studi della Tuscia, Italy*
[c]*University of Klagenfurt, Austria*

Abstract

Recommender systems are software applications that help users find items of interest in situations of information overload in a personalized way, using knowledge about the needs and preferences of individual users. In conversational recommendation approaches, these needs and preferences are acquired by the system in an interactive, multi-turn dialog. A common approach in the literature to drive such dialogs is to incrementally ask users about their preferences regarding desired and undesired item features or regarding individual items. A central research goal in this context is efficiency, evaluated with respect to the number of required interactions until a satisfying item is found. This is usually accomplished by making inferences about the best next question to ask to the user.

Today, research on dialog efficiency is almost entirely empirical, aiming to demonstrate, for example, that one strategy for selecting questions is better than another one in a given application. With this work, we complement empirical research with a theoretical, domain-independent model of conversational recommendation. This model, which is designed to cover a range of application scenarios, allows us to investigate the efficiency of conversational approaches in a formal way, in particular with respect to the computational

---

*Corresponding author.

Email addresses: `tommaso.dinoia@poliba.it` (Tommaso Di Noia), `donini@unitus.it` (Francesco Donini), `dietmar.jannach@aau.at` (Dietmar Jannach), `fedelucio.narducci@poliba.it` (Fedelucio Narducci), `claudio.pomo@poliba.it` (Claudio Pomo)

complexity of devising optimal interaction strategies. Through such a theoretical analysis we show that finding an efficient conversational strategy is NP-hard, and in PSPACE in general, but for particular kinds of catalogs the upper bound lowers to POLYLOGSPACE. From a practical point of view, this result implies that catalog characteristics can strongly influence the efficiency of individual conversational strategies and should therefore be considered when designing new strategies. A preliminary empirical analysis on datasets derived from a real-world one aligns with our findings.

1. Introduction

System-generated recommendations have become a common feature of modern online services such as e-commerce sites, media streaming platforms and social networks. In many cases, the suggestions made by the underlying recommender systems are individualized according to the stated or assumed needs and preferences by the user. In the most prominent applications of recommender systems, *e.g.,* on Amazon.com, Netflix or YouTube, the user preferences are estimated based on the observed past user behavior. There are, however, also a number of application domains where no past interactions logs are available or where the user's needs and preferences might be different each time the user interacts with the service. Consider, for example, someone seeking a recommendation for a restaurant this evening for a party of four, where the requirements include that the location is nearby, that the prices are modest, and that there is a vegetarian option. In such a situation, the user's current needs and preferences have to be interactively acquired by the system to make a suitable recommendation.

The class of systems that support such interactions are called *Conversational Recommender Systems* (CRS). In these systems, the recommendation process consists of an interactive, multi-turn dialog, where the system's goal is to learn about the user preferences to the extent that appropriate recommendations can be made. The corresponding preference elicitation process can be implemented in different ways, ranging from predefined fill-out forms to natural language interfaces—see [1] for an overview. In particular for this latter class of interfaces we observed substantial progress in terms of voice recognition and natural language understanding in recent years, leading to the development of voice-controlled devices like Apple's Siri or Amazon's

Alexa, and continuously improved chat-bot systems.

Independent of the interaction modality, a common approach implemented in many CRS is to ask users about desired or undesired features in sought items, or to ask them about individual items they like or dislike. This applies both for traditional critiquing approaches [2], interactive advisory systems [3], and recent learning-based approaches [4, 5]. In that context, a typical goal when designing a CRS is to minimize the effort for users by asking as few questions as possible, *i.e.,* to increase the *efficiency* of the dialog. The underlying assumption is that a lower number of required interactions leads to a better usable system. Different proposals were made in the literature to increase the efficiency of the recommendation process. These include the selection of predefined dialog paths according to the user's expertise [3], the automatic creation of compound critiques based on different strategies [6, 7], or the selection of the best next question to ask based on reinforcement learning strategies [8, 4].

Today, research in this area is almost entirely empirical. Typical research designs are based on simulations or user studies, in which two or more interaction strategies are compared in one or two application domains based on real or synthetic datasets. The corresponding efficiency measures are, for example, the number of required user interactions or the perceived difficulty and effort of the recommendation dialogs. Such empirical studies are certainly important and insightful. Little is known, however, about the theoretical aspects of the underlying interactive recommendation processes. In many cases, the recommendation process is defined in an informal or semi-formal way, or only given implicitly in terms of an algorithm that selects the next questions to ask. Theoretical questions regarding, *e.g.,* the computational complexity of determining a good or the best interaction strategy can, however, not be answered without a formal characterization of the underlying problem.

With this work, we address this research gap and provide a theoretical model of conversational recommendation. The model is designed in a domain-independent way and aims to cover a wide range of realistic application scenarios. A conversational recommendation process is modeled as a sequence of states, where state transitions correspond to common *conversational moves* [9] that can be found in the literature, including in particular preference statements regarding item features or individual items and the relaxation or revision of previously stated preferences. Furthermore, since the model is agnostic about the algorithm that is used to select and rank

3

the objects for recommendation—*i.e.,* the recommendation algorithm—this model can be applied in various application domains. As a result, the model serves as a basis to analyze important theoretical properties of conversational recommendation processes. We particularly focus on the problem of the computational complexity of finding an efficient conversational strategy. Our theoretical analysis reveals that this problem is NP-hard, but in PSPACE in general, and we furthermore derive which factors in the catalog influence the complexity of the problem; in particular, we show that for a special class of catalogs, the upper bound lowers to POLYLOGSPACE. From a practical perspective, this analysis specifically shows that the efficiency of individual conversation strategies is tied to the characteristics of the available item catalog. Observations from an empirical analyses on real-world datasets support these theoretical considerations.

The paper is organized as follows. Next, in Section 2, we discuss existing works and outline our research goals. We provide a model formalization in Section 3, then our theoretical results in Section 4, and the outcomes of our empirical analyses in Section 5.

2. Previous Work and Research Goals

Dialog efficiency is one of the main dimensions in which CRS are evaluated in research literature [1]. Here, efficiency usually refers to the time or effort that is needed by a user to find a suitable item. The corresponding underlying assumption is that users will find a system more useful if it requires less effort for them. Likewise, if a recommendation dialog takes too many steps, users might quit the conversation or, even worse, abandon the system as a whole.

2.1. Increasing Dialog Efficiency

Efficiency can in general be increased by designing the system in a way that guides users more directly to the items that match their preferences. In a non-interactive recommender system, efficiency is achieved through accurate predictions of what is relevant for users given their long-term preferences. In CRS, where the preferences are interactively elicited, one can *in addition* try to find better conversation *strategies*. By "conversation strategy" we mean the choices that CRS can make during the dialog. After a few interaction steps, a system might, for example, either decide to make a recommendation or to ask more questions in order to obtain a more complete picture of user

preferences. While asking more questions means more interaction cycles in the short term, having more information might reduce the risk of providing irrelevant recommendations and more user effort in later phases of the dialog. Another decision point exists after the system has decided to ask more questions. In this case, the system can usually select between several options regarding what to ask next.

In the literature, different technical approaches were proposed to increase the efficiency of the recommendation process though improved conversation strategies. Here, we give a few examples of different types of such approaches. Note, however, that with this work our aim is not to propose a new strategy, but to understand, on a theoretical level, how complex it is to make optimal choices.

The conceptually most simple approach is to build a CRS that is based on a static set of application-specific rules about how to continue the dialog. In the interactive sales advisory system Advisor Suite [3, 10], for example, the possible dialog paths are predefined in terms of a state-machine. How the system can proceed in the dialog is thus determined through the transition graph. The actual choice of the system is then determined based on manually-defined decision rules. For example, the system could adapt the way the questions are asked depending on the user's self-reported expertise in the domain or based on previous user answers. To what extent such hand-crafted decision rules actually increase the efficiency of the conversations was unfortunately not evaluated in the mentioned papers.

Various more elaborate approaches were proposed in the context of CRS that are based on *critiquing*. In critiquing-based systems [2], the idea is that the CRS provides a recommendation relatively early in the dialog or even starts with an initial recommendation. Users can then apply critiques on a given recommendation—*e.g.,* in the form "cheaper" or "lighter" for a camera recommender system—after which the system makes a new recommendation. The system therefore interactively refines the preference model until a recommendation is accepted (or no recommendation is found to be suitable). While such an approach in its basic form is intuitive and conceptually simple, it might turn out to be not very efficient in particular when there are many item attributes the user can apply critiques on. Depending on the implementation, a critiquing system might offer all attributes for critiquing at once or incrementally ask for desired feature values, also known as "slot filling". In case of incremental slot filling, a naive implementation might end up asking too many questions about features that are either *(i)* not relevant for most

of the users or *(ii)* not particularly well-suited to narrow down the range of remaining options.

To deal with such problems, *dynamic* and *compound* critiquing approaches were proposed [11, 7, 12, 13, 14]. In the compound critiquing approach [7], for example, the idea is not only to present critiques to the user that concern more than one feature (*e.g.,* "less expensive *and* lighter"), but to determine possible compound critiques based on the properties of the item catalog. During an ongoing session, it might for example be most helpful to propose compound critiques that, if applied, would rule out a larger fraction of the available options.

The dynamic and data-based selection of the next question to ask or, more generally, the next conversational move, is however not limited to critiquing-based approaches. In the context of such slot-filling strategies—where the system questions mostly relate to item properties (or: facets)—various approaches were put forward to determine the best order of the questions. Often, such approaches are *entropy-based* and consider the potential effects of individual questions (and their answers) on the remaining space of options [15, 16, 17, 18]. In some cases, feature popularity information is considered as well [15]. Several alternative technical approaches to decide on the next action are based on machine learning, using, for example, reinforcement learning techniques [19, 20, 8] or recurrent neural networks [21].

Instead of asking for desired item features, some approaches are based on eliciting feedback on items considered as a whole. This could be done by asking for like or dislike statements for individual items or by asking for the relative preference regarding two or more items [22]. One main question in that context is how to select these items or sets of items to maximize the efficiency of the process. Such a selection can for example be based on item popularity or based on diversity considerations [23, 24, 25]. In some approaches, the system can also learn to decide if an absolute rating should be elicited or pairs of items should be presented to the user [4, 26].

2.2. Measuring Dialog Efficiency

The most common approach to measure dialog efficiency in the literature is to count the number of required interaction cycles until a recommendation is accepted by the user. Such an approach is followed in many works that are based on critiquing, but also in chatbot-like applications and in more recent learning based systems [27, 28, 29, 30, 8, 6, 31, 32, 33, 34, 23, 35, 36, 19]. In most of these works, the interaction between a user and the *CRS* is

simulated. In such simulations a software agent is developed, that has a certain preference profile and answers requests made by the system based on the underlying preferences. The common assumptions here are that the agent knows all preferences from the beginning, behaves rationally and truthfully, and does not change its mind during the conversation. In reality, not all assumptions might hold, and we include this aspect in our simulations (see later sections), by having users revise their preferences, *e.g.,* in case no item matches the initial preferences.

Alternative efficiency measures that are often used in the context of user studies include the *task completion time* [8, 35, 37]. We may in general assume that shorter task completion times usually means that the recommendation process was more efficient. However, a longer task completion time can, depending on the domain, also mean that the users found more interesting options to explore [38].

In addition to these objective measures—number of interaction turns and task completion time—some researchers also rely on *subjective* measures in the context of studies that involve humans. In such studies, the participants are usually asked after the main experiment task how they perceived the effort that was needed, *e.g.,* to find a suitable item. In most cases, efficiency is only one of several variables that are assessed in such post-task questionnaires. Examples of works that use efficiency-related measures are [39, 40, 22, 41, 8].

2.3. Research Goals

Research Gaps and Goals. Existing research on the efficiency of different conversational recommendation strategies, as mentioned above, is entirely empirical. Either the research is based on studies with simulated users, or it is based on user studies in which participants usually interact with a prototype system. In either case, such studies are based on one or a few particular experimental configurations. In particular, often only one specific item catalog is considered when the efficiency of a newly proposed conversation strategy is demonstrated. However, as the theoretical analysis shows later in this paper, the characteristics of the catalog, in particular, in terms of the number of item features, and the number of feature values that are shared by several items, may have an impact on the efficiency of a given strategy. Depending on such characteristics, for example, in one case it might be much more efficient to ask the user about a feature preference than asking for an item preference, whereas in another situation, the advantage is negligible.

Moreover, existing research has not looked yet at the computational complexity of choosing an optimal strategy in a conversation. Our research aims at closing this gap and provides a formal definition of the conversational recommendation problem, based on which such complexity analyses are performed. Furthermore, we complement our theoretical analysis with results obtained from simulation-based experiments with data elicited from a real dataset (MovieLens).

Research Scope. A multitude of CRS have been proposed in the past, focusing on a variety of different problem domains. Depending on the target domain, such CRS are often diverse in terms of what functionality they offer. Specifically, existing CRS vary largely in terms of the supported user intents. In their recent work, Cai and Chen [42] developed a comprehensive list of user intents and corresponding system actions (or: conversational moves) in CRS.

In our work, we focus on the most common CRS in the literature: those that base their conversations on desired or undesired values for item features ("slot filling") and/or the acquisition of preferences of users regarding an item as a whole. Our formalization therefore includes actions that, for example, relate to the initial acquisition of preferred feature values, but also considers additional user actions in such applications, such as the relaxation or revision of constraints [43]. The resulting formalization therefore represents an abstraction from various existing approaches from the literature, and the resulting insights therefore apply to a larger range of CRS implementations. In this context, note that our formalization considers the actual item ranking task as a black box. This means that our theoretical framework is independent of the specific type of algorithm that is used for determining the order of the recommendations. Thus any type of approach, *e.g.,* collaborative, content-based, or hybrid, can be used internally.

Regarding the measure for efficiency, our discussion showed that while a number of alternative measures were applied in the literature, counting the number of required interaction steps is the predominant choice of researchers. Therefore, we also use this measure in our theoretical analysis and in our simulations.

3. Model Formalization

On a conceptual level, a CRS—as considered in our study—works as follows. The system main task in the interaction with the users is to elicit their

preferences[1] regarding item features and items. To that purpose, the system maintains information about these preference statements in a user model. These preference statements determine which of the items of a given catalog qualify to be recommended. Therefore, in our theoretical framework we assume a retrieval-based item filtering approach, which is commonly used in critiquing-based and constraint-based approaches to recommendation [44]. In analogy to database-oriented approaches, we therefore use the term "query" when referring to the user's preferences, as these preferences lead to the retrieval of matching items from the catalog. The retrieved items are then ranked according to their assumed utility for the user. Such a ranking can be based on any type of information, *e.g.,* the popularity of certain items. In order to carry a general analysis, in our approach we abstract from the details of this ranking, as mentioned above.

To model the conversational recommendation process, we rely on the notion of *state* of a conversation, and what transformations this state can be subject to, depending on the interaction. Each expressed user preference, for example, leads to a change of the state of the conversation and may also imply that the set of suitable item recommendation changes. This formalization through conversation states ultimately serves us as a basis to study the efficiency of conversational strategies, in the sense that the most efficient conversational strategies will be the ones minimizing the number of states the conversation must pass through, to reach an end.

3.1. Basic components

We summarize below the components and symbols of our formalization, and we explain them in detail in the rest of the section. In particular, we have:

1. a *Catalog* of items $\mathcal{C}$;

2. the user model, which is a 4-vector $U = \langle Q, \mathcal{K}, \mathcal{P}, \mathcal{N} \rangle$, where the *query* Q denotes the user's positive preferences regarding item features (*desiderata*), $\mathcal{K}$ are constraints, in the form of undesired values for a feature, $\mathcal{P}$ is a set of positively-rated *preferred* items, and $\mathcal{N}$ is the set of items that are negatively rated by the user;

---

[1]In this paper, we use the words "preference", "desire", "needs" as synonyms.

3. the *Recommender* system R (a black box in this paper), that given U ranks items in $\mathcal{C}$, and returns the top-k items, which we abbreviate as $R(U)$; we stress our being agnostic in modeling R, which could be content-based, or based on collaborative filtering, etc. Following the intuition, we assume that $R(U) \cap \mathcal{N} = \emptyset$—that is, no negatively-rated item is ever proposed;

4. the *state* of the conversation as the 5-vector $\langle Q, \mathcal{K}, \mathcal{P}, \mathcal{N}, R(U) \rangle$ which we denote[2] also as $U + R(U)$, *i.e.,* the state is composed by all of the above elements but for the catalog, which we consider constant during the conversation;

5. the interactions between the user and the recommender as transformations of U, denoted by τ.

This formalization provides us with the necessary notation to study *strategies*, *i.e.,* algorithms that, given a state of the conversation, suggest the next move in order to proceed towards a successful recommendation.

To ease reading, we adopt the following syntax conventions: atomic elements are denoted by lowercase letters, *e.g.,* x, t, v; sets and vectors of atomic elements are denoted by uppercase letters, *e.g.,* C, D, I, Q; and sets and vectors of the previous sets and vectors are denoted by calligraphic letters, *e.g.,* $\mathcal{C}, \mathcal{K}$. We now delve into each element in detail.

Features. Let $\{f_1, f_2, \ldots, f_p\}$ be a set of *features* for items and queries, where each feature f_i can take values in a *domain* D_i. In the rest of this paper, we assume that features are only single-valued, *i.e.,* we do not consider set-valued features. When making examples, we often characterize the feature with a name instead of the index, as in $f_{cuisine}, f_{director}$. We highlight the *Boolean domain* $B = \{false, true\}$ as a special one. While any domain D could be translated into a number of $|D|$ Boolean domains via one-hot encoding, we prefer not to use such an encoding because of its exponential-space blowup[3] that would hamper our subsequent computational study. We denote the union of all domains as $V = (\bigcup_{i=1}^{p} D_i)$.

---

[2]Inspired by Python, we use "+" for vector concatenation, *e.g.,* $\langle A, B, C \rangle + \langle D, E \rangle = \langle A, B, C, D, E \rangle$.

[3]The values of a feature need $\lceil \log_2(|D|) \rceil$ bits to be encoded, while their one-hot encoding needs $|D|$ bits.

Variables. In addition to domain values, let $X = \{x_1, x_2, \ldots\}$ be a set of variables. Intuitively, a variable is a placeholder for a query feature which the user has not yet expressed a preference on. As an abstraction of both variables and values, we define terms. A *term t* can be either a value in a domain, or a variable; so the set of terms is $T = V \cup X$. Terms are needed to model user queries, which we introduce next.

Queries. The user's stated preferences are represented by a *query vector* of terms $Q = \langle t_1, t_2, \ldots, t_p \rangle$, where each position i (called *slot*) in the vector is associated with a feature f_i; features act as names for vector positions—similar to field names in relational databases. We say that "the feature f_i contains the term t_i in Q" when the i-th position in Q holds the term t_i—written $f_i(Q) = t_i$. Intuitively, when a feature f contains a variable x, it means that we have no knowledge about the user's preferences in this respect. Remember that one goal of the interactive process is to fill in such features/slots.

Constraints. A query represents the *positive* part of a user's desires; to represent the negative part—what the user definitely does not want—we define the *constraints* on Q as another vector $\mathcal{K} = \langle C_1, C_2, \ldots, C_p \rangle$, where each constraint $C_i \subset D_i$ is a set of values the user dislikes for feature f_i. Such a constraint represents the negative part of the history of the conversation, namely, the negative answers given by the user during the interaction—*e.g.,* "I don't want horror movies tonight", which is formalized as $horror \in C_{genre}$. Observe that for such negative constraints, we limit this formalization to the most common case implemented in CRS, that is, we consider here only single values.

Items. We represent items using the same feature space as the queries, but for variables. Given a catalog $\mathcal{C} = \{I_1, I_2, \ldots\}$ of items, we let an item $I = \langle v_1, v_2, \ldots, v_p \rangle$ be a vector where each feature f_i in position i contains value v_i—written $f_i(I) = v_i$. In this study we consider only *complete knowledge, i.e.,* all feature values are known. [4]

Substitutions. During the interactive recommendation process, we continuously learn more about the user's preferences, which leads to changes in the user model U. We call such changes *transformations.* We define the simplest

---

[4]Extending the formalization to incomplete item knowledge is left for future work.

form of transformation as slot filling, which in our setting means substituting a variable with a value. A *simple substitution* $\sigma : X \to V$ is a function mapping a variable x to a value v. Apart from such a change, the substitution leaves the rest untouched—more formally, the substitution σ is the identity function on every term but x, which is changed to v. Recall that we are interested in substitutions in the user model U, and not in the items, whose descriptions do not contain variables.

We extend σ to vectors position-wise, *i.e.,* we have $\sigma(\langle t_1, t_2, \ldots, t_p \rangle) = \langle \sigma(t_1), \sigma(t_2), \ldots, \sigma(t_p) \rangle$. In this way, we can use the notation $\sigma(Q)$, knowing that σ will change only the feature containing x. Clearly, if a user already stated that s/he dislikes a specific feature value, such a value should never be proposed again during the conversation. Formalizing such an intuition, we say that a substitution is *coherent with* $\mathcal{K}$ if it does not substitute a variable with a value forbidden by the constraints, in formulas: for all $i = 1, \ldots, p$ it holds that $\sigma(t_i) \notin C_i$. From now on, we consider only substitutions coherent with $\mathcal{K}$.

We let substitutions $\sigma_1, \sigma_2, \ldots, \sigma_h$ be repeatedly applied to a term as $\sigma(t) = \sigma_1(\sigma_2(\ldots(\sigma_h(t)\ldots)))$, and also such nested substitutions σ can be applied to vectors position-wise, written as $\sigma(Q)^5$. Moreover, the sequence of such nested substitutions preserves the order (in reverse) in which the subsequent slot-filling interactions change the state of the conversation.

An item I *matches* a query Q if there is a substitution σ coherent with $\mathcal{K}$, such that $\sigma(Q) = I$. In practice, σ formalizes all the slot fillings that would lead from Q to the proposal of (exactly) I. We say that a substitution σ, coherent with constraints $\mathcal{K}$, *satisfies a query Q in $\mathcal{C}$* if there is an item $I \in \mathcal{C}$ such that $\sigma(Q) = I$. Given a catalog $\mathcal{C}$, we write $Q(\mathcal{C})$ for the set of items matching Q, in formulas: $Q(\mathcal{C}) = \{I \in \mathcal{C} \mid \exists \sigma : \sigma(Q) = I\}$.

We note that when a feature f contains a variable—*i.e.,* when the user specifies no preference for f—we do not know whether the user *does not care* about f—so that the user would accept any value for f—or the user *does not know* that there are some alternatives that fit her preferences while others do not. For instance, suppose a restaurant recommender describes items with some features, among which there is a Boolean feature regarding dress code;

---

[5]Even if each substitution of the sequence changes one variable with a value, such a compositional extension allows us to write simply $\sigma(Q)$ instead of specifying which simple substitution changes which feature, as it would be in heavier notations like $\langle \sigma_1(t_1), \sigma_2(t_2), \ldots, \sigma_p(t_p) \rangle$.

and suppose that when starting the conversation, the user did not mention preferences about dress code. When facing the proposal of a restaurant whose dress code feature is *true*, the user might say "*no, I don't want dress code restaurants; sorry I didn't mention it before*". Formally, this would mean that the conversation starts with a query Q_0 in which $f_{dressCode}(Q_0) = x$, the recommender (after some interaction) proposes a restaurant I for which $f_{dressCode}(I) = true$ (so I matches Q_0 with $\sigma(x) = true$), the user rejects it with the above dialog, and the conversation proceeds with the updated constraints $\mathcal{K}'$ in which $C_{dressCode} = \{true\}$ (the disliked value).

Positively- and negatively-rated items.. Both Q and $\mathcal{K}$ formalize the "analytic" part of a user's preferences, *i.e.,* what s/he thinks about particular values of item features. However, sometimes the user may want to express a holistic opinion on an item as a whole, *e.g.,* "*I don't want to go to restaurant Y anymore, I had a bad impression last time I went*" (dislike), or "*I liked movie Z very much*" (preference). We model such statements as two sets of items, $\mathcal{P} \subset \mathcal{C}$ (positive rating) and $\mathcal{N} \subset \mathcal{C}$ (negative rating). $\mathcal{P}$ and $\mathcal{N}$ are part of the user model U as mentioned above, so that our formalization is as general as possible. We exclude from the formalization of the conversation the trivial solution that directly proposes items in $\mathcal{P}$; such items are not proposed (*e.g.,* in the movie domain, they might be movies already seen; in the restaurants domain, they could be appreciated restaurants far away from the user's current position).

3.2. States and Transformations

A *state* of a *CRS* is the 5-vector $U + R(U)$. The conversation starts with $U_0 = \langle Q_0, \mathcal{K}_0, \mathcal{P}_0, \mathcal{N}_0 \rangle$ as the initial information about the user, and $U_0 + R(U_0)$ as the initial state. If there are no past item ratings available from previous interactions, we face a 'cold start' situation where $\mathcal{P}_0 = \mathcal{N}_0 = \emptyset$. When a new conversational session starts, we let $Q_0 = \langle x_1, x_2, \ldots, x_p \rangle$ (all variables) and $\mathcal{K}_0 = \langle \emptyset, \emptyset, \ldots, \emptyset \rangle$ (all empty sets), *i.e.,* no preference on individual features was expressed yet. Recall that we assume the catalog $\mathcal{C}$ as constant in time—*i.e.,* no items enter or exit the catalog during the conversation.

In our model, we are formalizing system-driven conversations, where a conversation consists of a sequence of interactions. Given a state $U + R(U)$, the interactions initiated by the system can be one of the following.

1. the system asks the user to *fill in* (provide) a value for a particular feature under-specified so far (a variable in Q);

2. the system asks the user to enlarge a too narrow choice for a feature value in Q;

3. the systems asks for changing a feature value in Q;

4. upon rejection of one or more items, the system can ask the user whether there is a specific feature of these item(s) she dislikes.

The user can react to the above system prompts with one of the following interactions:

(a) given one or more recommendations $R(U)$, the user can accept one of them, or reject them;

(b) the user can state that she dislikes every item whose feature f is filled with a particular value v (*e.g.,* "*I don't like green cellphones*")

 While this set of possible interactions might seem limited, remember that it covers a wide range of conversation strategies that were described in the literature, as discussed in Section 2. Regarding the specific user reactions, Interaction (a) amounts to adding the rejected item(s) to $\mathcal{N}$, while Interaction (b) amounts to adding the value v to a set C_i in the constraints vector $\mathcal{K}$. Regarding system prompts, in what follows we show how such interactions can be expressed in terms of *transformations* on U, which extend substitutions as follows: a transformation $\tau : T \to T$ can map any term to another term. In this way, substitutions are just a particular case of transformations. We need transformations to model system-user interactions in a more general way than just turning a variable into a value (as is the case for slot filling).

3.2.1. Interaction 1: Slot filling
 Suppose the user issued a very general query Q to start with, *e.g.,* "*I would like to go to a restaurant tonight*". The system has so many answers to this query that presenting only the top-k ranked items $R(U)$ might miss a satisfying experience, and given that $f_{cuisine}(Q) = x$, it asks in turn: "*Which kind of cuisine do you prefer?*" and provides a set of possible feature values of the available items. When this set is too large to be presented to the user, another variant could be that the system actively suggests some value,

e.g., "*What about French cuisine?*" given that there are plenty of French restaurants in the catalog. The formal counterpart of this dialog is a simple substitution σ such that $\sigma(x) = French$, expressing the user's choice, and after this simple transformation is performed, the new state is $U' + R(U')$, where $U' = \langle \sigma(Q), \mathcal{K}, \mathcal{P}, \mathcal{N} \rangle$, and $f_{cuisine}(\sigma(Q)) = French$.

3.2.2. Interaction 2: Slot unfilling

Here transformations more general than substitutions come into play. Imagine a conversation between a restaurant recommender and a user, in which after some questions (slot fillings) regarding cuisine, location, price level, etc. the recommender asks the user: "*What about Restaurant I_5? Japanese cuisine, midtown, upper-level price, no dress code?*"; yet getting a "Dislike" from the user. Suppose there are no more restaurants with the features set so far. Then the recommender may say "*May I propose you other types of cuisine?*". Formally, $\mathcal{N}' = \mathcal{N} \cup \{I_5\}$ (since Item I_5 is rejected), and the query Q in which $f_{cuisine}(Q) = Japanese$, is transformed by $\tau(Japanese) = x$ (a fresh variable), yielding a new state $U' + R(U')$ with $U' = \langle \tau(Q), \mathcal{K}, \mathcal{P}, \mathcal{N}' \rangle$, so that $f_{cuisine}(\tau(Q)) = x$. This interaction is a first convenient way of driving a conversation out of a dead end, without committing on a new choice.[6] A second way follows below.

3.2.3. Interaction 3: Slot Change

This interaction is similar to the previous case, but now the system proposes a specific change to the user: "*I have no other Japanese restaurants in midtown, upper-level, with no dress code; but downtown, there are a few. Are you willing to change midtown for downtown? Other alternatives are: north outskirts, or riverside*"[7] Of course, if the user accepts an alternative, the feature should change value. The formalization is also similar to the previous case: the conversation is in a state $U + \{I_5\}$, and feature $f_{location}(I_5)$ contains the value *midtown*. Again, since the user dislikes I_5, it is now $\mathcal{N}' = \mathcal{N} \cup \{I_5\}$. We let $\tau(midtown) = downtown$ (or whatever other value is chosen by the user), stepping into the new state $U' + R(U')$ with $U' = \langle \tau(Q), \mathcal{K}, \mathcal{P}, \mathcal{N}' \rangle$ so that $R(U') \neq \emptyset$ (since there are recommendable Japanese restaurants downtown).

---

[6]This kind of transformation has also been called *query relaxation* [45, 43].
[7]Algorithms for computing such alternatives are proposed, *e.g.,* in [46, 47].

We note that in the area of Knowledge Representation, slot unfilling could be considered a special case of knowledge *contraction* [48], while slot change is a form of *revision* [49]. We occasionally use such names instead of slot unfilling and slot change in the rest of the paper.

The formalization described so far can be distilled into a definition of conversation as follows.

Definition 1. *A Conversation is a sequence of transformations $\tau_1, \tau_2, \ldots, \tau_z$ that starting from an initial state $U_0 + R(U_0)$ ends in a final state of success (the user chooses an item), or failure.*

We observe that failure may occur because *(i)* either a maximum number of interactions has been reached, or *(ii)* the user gives up after a number of interactions, or *(iii)* the conversation reaches a state $U + \emptyset$ in which it happens that both the CRS has no more items satisfying the user's preferences, and the user does not want to reconsider any of the preferences stated in U.

4. Devising Optimal Strategies

In the development of CRS with system-driven conversations, a cornerstone is the method the CRS uses to decide how to proceed in the conversation; we call such a method a *strategy*. Our goal in this section is to study the problem of developing efficient conversation strategies for CRS. For our setting, a CRS *strategy* is an algorithm that, in every possible state of a conversation, suggests the system its next action, among the ones formalized above. As discussed in Section 2, the efficiency of a CRS strategy can be studied in terms of the number of interactions the user needs to reach an item I. One can consider the worst case, *i.e.,* the maximum number of interactions (for some "difficult" item), or the average case, with respect to a probability estimating how likely item I will be the chosen one. In this paper we study the minimization of the maximum number of interactions, leaving the probabilistic setting for future work.

To study conversation strategies, we need some reasonable assumptions about how the user answers the system prompts. In our work, we assume that there is a set $\mathcal{J}$ of *ideal* items the user would be satisfied with. Without loss of generality, since we are studying efficiency, we assume that $\mathcal{J} \cap \mathcal{C} \neq \emptyset$, meaning that there always exist at least one item in the catalog that would satisfy the user, and the only point is how many interactions are needed

to find it[8]. Observe also that we do not set $\mathcal{J} \subset \mathcal{C}$, *i.e.,* we assume that there are "ideal" items the user would like, but which are not available in the catalog. They are exactly such items that make search difficult, since the user might drive the search (in the conversation) towards such unavailable items, heading towards available ones only later on.

We assume that the user, during the conversation, adopts (even unintentionally) the following *truthful attitude* when interacting with the CRS:

- the user chooses a slot filling $\sigma(x) = v$ for a feature f_i, *if and only if* there is at least one ideal item $I \in \mathcal{J}$ whose feature f_i is actually filled by v—in symbols, $\exists I \in \mathcal{J} : f_i(I) = v$. Note that this is a double implication, *i.e.,* if there is no such ideal item, the user does not choose (or accepts as a suggestion) to fill the slot f_i with value v;

- similarly for slot change: the user chooses (or accepts the suggestion) the change of a slot value $\tau(v) = w$ for a feature f_i, *if and only if* there is at least one ideal item $I \in \mathcal{J}$ such that $f_i(I) = w$;

- user's like/dislike statements about whole items are always coherent with ideal items $\mathcal{J}$, *i.e.,* when the user states that she likes an item I, it is $I \in \mathcal{J}$, and when the user states that she dislikes an item I, then $I \notin \mathcal{J}$.

4.1. Lower Bounds

We first show that our problem of devising efficient strategies for CRS includes Binary Decision Trees (BDT) minimization as a special case. We recall the definition of a BDT [50] for our purposes. A BDTs is a procedural representation of a *decision table* defined as a tabular representation of a function whose domain is contained in B^p and whose range is the set of decisions. Given integers p, and $q \leq 2^p$, a Decision Table is a $q \times p$ Boolean matrix[9] in which the p columns represent conditions/tests that can be *true* or *false*, and the q rows represent the combinations of such conditions—one

---

[8]This means that we do not consider here failure cases; in particular, we consider neither the case where there is a maximum number of interactions (Case *(i)* above), nor the case where the user gives up (Case *(ii)* above). Moreover, we consider no cutoff by the system (which would hide worst cases beyond that threshold).

[9]The use of the symbol p (the same symbol used for the number of features in our formalization) is intentional here, as it will be clear later on.

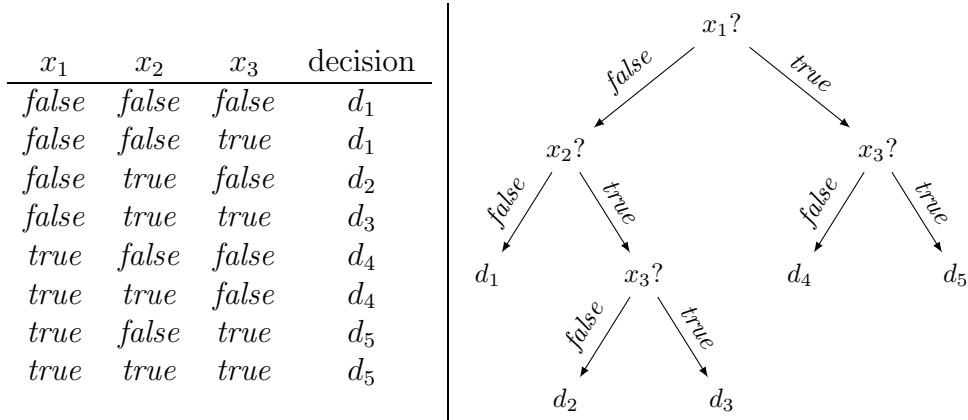| x_1	x_2	x_3	decision
false	*false*	*false*	d_1
false	*false*	*true*	d_1
false	*true*	*false*	d_2
false	*true*	*true*	d_3
true	*false*	*false*	d_4
true	*true*	*false*	d_4
true	*false*	*true*	d_5
true	*true*	*true*	d_5

Table 1: A decision table (left) and a possible, optimized BDT representing it (right), whose depth is 3

of which has to be assessed to take the appropriate decision. Each decision could be associated to one or more rows—all the combinations of conditions in which this decision has to be taken—and rows corresponding to impossible combinations are usually omitted. We give an example in Table 1 (left).

A BDT is a strategy for testing the conditions that leads to a decision in every possible situation. In particular, the tree contains a test in each internal node, a label *true/false* in each arc, and a decision in every leaf (possibly repeated). An example of BDT is given in Table 1 (right). The procedural interpretation of the BDT is: starting from the root, test the condition in the node, and proceed along the arc labeled with the result of the test. Repeat until a leaf is reached, which represents the decision to be taken. The *cost* of the decision taken is the number of tests performed, which coincides with the length of the path from the root to that leaf. For a given table, there can be several BDT representing it, and BDT minimization is the problem of finding a BDT whose decision costs are all within a maximum, which coincides with the *depth* of the tree (*i.e.,* the maximum path length from the root to a leaf in the tree).

. The cost could also be averaged over all leaves, yielding a search for the minimum *expected* cost.

The decision problem related to such optimization problem is: "Given a decision table and an integer M, is there a BDT representation of the table

whose decision cost is $\leq M$?" This problem was proved NP-complete[10] for the minimum expected cost by Hyafil and Rivest [51], and then extended to minimum depth by Chikalov *et al.* [52, Prop. 13].

To show that BDT minimization is a particular form of the problem of devising optimal CRS strategies, given a decision table $q \times p$ we fix a catalog $\mathcal{C}$ of q items, each item with p Boolean features. Decisions are one-one with items, so the number of decisions is $|\mathcal{C}|$, and conditions/tests are one-one with Boolean features. The CRS begins the conversation from a *cold-start state* in which no preference is known yet. More precisely,

- the query contains only variables: $Q_0 = \langle x_1, x_2, \ldots, x_p \rangle$;

- the constraints vector is a p-vector of empty sets $\mathcal{K}_0 = \langle \emptyset, \emptyset, \ldots, \emptyset \rangle$;

- $\mathcal{P}_0 = \mathcal{N}_0 = \emptyset$.

Moreover, for our lower bound we suppose that the answers given by the user always lead to an item that exists in the catalog, that is, $\mathcal{J} \subseteq \mathcal{C}$, so that once the system eventually proposes an item (after a conversation in which the user has the truthful attitude explained above), the user immediately accepts it. In this situation, the minimal interaction sequences are the ones in which only slot filling is performed, going from the root of the tree to a chosen item in a leaf. Such conversations composed solely by slot fillings on Boolean features are equivalent to the paths of a BDT, hence a lower bound on the efficiency of such conversations is also a lower bound for minimizing general conversational strategies.

Borrowing from results about the complexity of finding optimal BDTs, we prove below that the problem of finding an efficient CRS strategy is NP-hard in general, since it includes a decision problem which is NP-complete.

Theorem 1. *The following problem is NP-hard: "given an integer M, a catalog $\mathcal{C}$, an initial cold-start state $U_0 + R(U_0)$, where $U_0 = \langle Q_0, \mathcal{K}_0, \mathcal{P}_0, \mathcal{N}_0 \rangle$, and $Q_0 = \langle x_1, x_2, \ldots, x_p \rangle$, the constraints $\mathcal{K}_0$ are a p-vector of empty sets, and $\mathcal{P}_0 = \mathcal{N}_0 = \emptyset$, is there a conversation strategy made only of slot filling questions, such that the maximum number of interactions with the user is less than or equal to M?"*

---

[10]In the general case each test may have its own cost, and each combination of conditions may have a probability attached to it, but the problem is NP-complete even with uniform costs and probabilities, which is the case we are describing here.

Proof. We give a polynomial reduction from BDT minimization, that we precisely recall here: given a set of objects $O_1, \ldots, O_c$, and a set of tests $T_1, \ldots, T_p$, such that each test returns *true* for exactly three objects[11], decide whether there is a BDT whose internal nodes are the tests, whose leaves are the objects, and whose depth is less than or equal to a given length M.

Given an instance of the source problem as above, we construct the target problem in conversational recommendation. Let the catalog $\mathcal{C}$ be made by c items: for each object O, there is an item I_O.

We let the Boolean features be one-one with tests *i.e.,* $f_i = T_i$, for $i = 1, \ldots, p$ while the items are defined as $I_O = \langle T_1(O), \ldots, T_p(O) \rangle$ *i.e.,* each feature f_i of an item I_O has the same value as the one returned by Test T_i on Object O.

We now prove that there exists a BDT solving an instance of the source problem, whose depth is less than or equal to M if and only if there is an efficient strategy for the target CRS, whose maximum number of slot filling interactions is less than or equal to M. Without loss of generality, we represent the strategy of the CRS as a binary tree, too.

"$\Rightarrow$" Suppose there exists such a BDT. Then define an efficient strategy as follows: take the BDT T, and substitute test labels T_i in the internal nodes with feature labels f_i whose filling should be asked to the user, leaving edge connections untouched (*true/false* test results in BDT are one-one with *true/false* user filling in the CRS strategy). Then substitute each leaf node of the BDT, marked with object O, with a leaf marked by item I_O.

"$\Leftarrow$" Suppose now that there exists an efficient CRS strategy for the target problem defined above—with p Boolean features and c items—whose maximum number of slot filling questions is within M. Such a strategy defines a BDT whose maximum path length is M as follows: transform the binary tree representing the strategy by substituting each internal-node feature label f_i with test T_i, and each leaf-node item label I_O with object O. Clearly such a BDT is isomorphic to the strategy, hence its depth is M. $\square$

The above theorem highlights the difficulty in optimizing CRS efficiency in the case in which the only transformation is slot filling; one may wonder how representative of the general case such "simple" case is, and whether

---

[11]Such a requirement comes from the original reduction [51] from the NP-complete problem EXACT-3-COVER. The proof extends to tests selecting more than three objects, but we keep the original special case for the discussion in Section 4.1.1.

interactions with slot unfilling/change could be more efficient than slot filling, after all. The following theorem shows that the other transformations are just "deroutes" from the shortest interaction paths, showing that conversations containing slot unfilling/change can always be transformed into conversations containing only slot filling, whose length is no worse than the initial one.

Theorem 2. *For every sequence of interactions from $U_0 + R(U_0)$ to a success state $U_n + \{I\}$, where I is accepted by the user, there is another sequence of interactions from a state $U'_0 + R(U'_0)$ to a state $U'_m + \{I\}$, whose transformations are only substitutions (slot fillings), and whose number of interactions m is not larger than the initial one, i.e., $m \leq n$.*

Proof. If the interaction is already composed by substitutions only, the claim is trivially true; suppose then that there are also slot unfilling and slot change interactions. For every contraction $\sigma(v) = x$, we focus on the state in which the feature contained v. If $f_i(Q_0) = v$, *i.e.,* the value was in the initial query Q_0, let $Q'_0 = \tau(Q_0)$ where $\tau(v) = x$, that is, we start with the variable unfilled from the very beginning. If instead the filling happened in an intermediate interaction, we just remove the interaction in which f_i was (erroneously) filled by v. Of course, we eliminate also the subsequent slot unfilling interaction. In both cases, the new sequence of interactions is shorter (by 1 or by 2) than n. We proceed in a similar way also for slot changing, and repeat the steps until no more such steps are performed. In the end, we obtain an interaction sequence that starts from $U'_0 + R(U'_0)$ (with U_0 possibly different from U_0), ends with the same satisfying item at $U'_m + \{I\}$, where $m \leq n$. $\square$

The above theorem tells us that the shortest possible conversations include the ones in which only slot filling is performed. Such conversations are the lucky ones, because there is no later revision of the slot fillings made. The goal of the CRS in this case is just to ask the least number of slot fillings leading to one item to propose, and stop. Hence, the above theorem evidences the significance of our NP-hardness reduction for CRS in general. We now make some points about this result.

4.1.1. Discussion on the lower bound

We stress that we are not interested in the above proof by itself, but in how it enriches the understanding of the computational problem of efficiency in CRS, thanks to the following observations:

1. The problem is NP-hard for the case in which each *true* feature selects three items. This means that NP-hardness does not depend on *feature entropy*, that would be maximal when a feature selects half the catalog. Moreover, the strategy that selects the feature with maximum entropy is only a heuristic, which does not guarantee optimality. Note also that since the original reduction [51] is from EXACT-3-COVER, and the more general problem EXACT-COVER is NP-hard too, our problem is NP-hard when a feature selects *at least* three items.

2. Hardness is instead entangled with catalog *incompleteness*, that is, the fact that not all combinations of feature values correspond to an available item; for (although unrealistic) complete catalogs—catalogs in which there is an item for every possible combination of feature values—optimal decision trees can be constructed in time polynomial in the catalog size [53].

3. The proof does not mention how the tree is represented, only the decision whether there exists such a tree or not. Hence, NP-hardness holds also in the case in which the tree is represented implicitly, as a circuit (or a neural network) which, given as input a state of the conversation, suggests the next feature to fill (usually called a *succinct* representation [54] of a tree).

4. Observe that there are at most $\binom{c}{3}$ possible different tests, hence $p \in O(n^3)$—*i.e.,* the number of features is bounded by a (low) polynomial in c. This means that NP-hardness shows up even when the number of features p and the number of items are polynomially related.

5. The proof does not rely on the existence of more than one user; hence the problem is NP-hard even if we want to devise an efficient strategy only for one particular user (or, equivalently, for non-personalized CRS).

6. The complexity analysis confirms the intuition that cold start is a worst case for CRS. When some features in Q contain a value, or $\mathcal{P}$ and $\mathcal{N}$ are not empty, this prunes some parts of the decision tree. Of course, the problem remains NP-hard in the size of the remaining features and items. Many solutions have been proposed to alleviate such a cold-start problem, however, as in many applications new entries are very

common (new-user and new-item problems), cold-start situations seem unavoidable, and algorithms that implement efficient strategies should cope with such situations.

7. Theorems 1 and 2 together show that, although the conversation is composed by actions of slot filling, slot unfilling and slot change, *the problem of devising an efficient strategy is not a problem of optimal planning* of the order of such actions. Slot unfilling and slot change are fundamental for another dimension of CRS evaluation, namely, *effectiveness*: when slot filling comes to a dead end (an item rejection), only the other two actions may steer the conversation out of that dead end, and lead to an acceptable alternative recommendation.

4.2. Upper Bounds

We now study the upper bound of the complexity of devising efficient strategies for CRS. Since generally feature domains contain several values, we represent slot filling strategies by (general) Decision Trees (DT), which are trees whose internal nodes can have more than two outgoing edges (one edge for each possible feature value). Rephrasing the definition of BDTs of the previous section, a DT for driving the conversation of a CRS contains a feature f in each internal node, a label with a value v filling f in each arc outgoing from the node, and an item in every leaf. An example of a catalog and a possible slot filling strategy for the conversation (represented as a DT) are given in Table 2.

director	starring	genre	movie
Spielberg	*Hanks*	*historical*	*Forrest Gump*
Spielberg	*Dreyfuss*	*action*	*Jaws*
Eastwood	*Hanks*	*action*	*Sully*

Table 2: A catalog (left) and a possible, optimized, conversational strategy (right) represented as a DT. Note that two features (out of three) are sufficient to distinguish all items.

23

The strategy represented by the DT is: starting from the root, ask the user to fill the feature in the node with a preferred value, and proceed along the arc labeled with the user's answer. Repeat until a leaf is reached, which identifies the item to propose. The *maximum cost* of such a part of the conversation is the maximum number of slot fillings asked to the user—*i.e.,* the depth of the tree. We recall that the problem of finding an optimal decision tree (in terms of minimum depth) is trivially NP-complete: it is NP-hard, because it contains BDT optimization (see previous section) as a special case, and it belongs to NP because the DT representing the strategy is polynomial in the size of the catalog. Hence a nondeterministic Turing Machine can compute such an optimal DT in polynomial time, making nondeterministic choices about which feature labels each internal node.

Observe that at any given point of the conversation, the focus is restricted to a set of **recommendable items** $\mathcal{S} \subseteq \mathcal{C}$ compatible with the answers given by the user so far; for example, if a user tells the system that one of her most favorite directors is *Spielberg*, the system concentrates on Spielberg's movies (the set $\mathcal{S}$) excluding (for the moment) the others. In such a set of items, $f_{director} = Spielberg$, and each other feature f_i may contain only a subset of all possible values in its domain D_i—only values compatible with Spielberg's movies. Continuing the example, for the items in $\mathcal{S}$ the feature $f_{starring}$ may contain a value among *Hanks, Duvall, Cruise, etc.*, but it never contains *De Sica*, because there are no movies starring *De Sica* whose director is Spielberg. We call such compatible values *Active Values* for f_i in $\mathcal{S}$, written as $AV(\mathcal{S}, f_i)$. More formally, $AV(\mathcal{S}, f_i) = \{v \in D_i \mid \exists I \in \mathcal{S} : v = f_i(I)\}$. In the previous example, once the user's answers restrict the movies to the set $\mathcal{S}$ of the movies directed by Spielberg, $AV(\mathcal{S}, f_{starring}) = \{Hanks, Duvall, Cruise, \ldots\}$. Active values will be useful both in the rest of this theoretical section, and in the next section regarding experiments.

An algorithm computing a DT for the catalog $\mathcal{C}$ could be used in CRS for a strategy using *only* slot filling. However, a single DT would be an incomplete strategy for CRS, since it does not consider the possibility of a reject action by the user, and consequently, slot unfilling and slot change. On the other hand, observe that the possible actions of the CRS cannot be mixed in any possible way: infinite loops—such as a slot filling followed immediately by an unfilling of the same slot—must be avoided. Hence, in what follows we make a reasonable restriction on possible strategies: every slot unfilling (contraction) or slot change (revision) can be performed only if, immediately before them, at least one item has been rejected by the user.

Such an item can therefore be added to the set $\mathcal{N}$ the user dislikes, and will not be proposed again. Recalling that the CRS can propose only items in $\mathcal{C} - \mathcal{N}$, this incrementality in $\mathcal{N}$ mirrors a reduction of the items to propose, and leads to a trivial upper bound on the possible strategies: each item could be proposed at most once. After all possible items in the catalog have been proposed and rejected, the conversation must stop. We call such strategies *well founded*, in analogy with well-founded sets, and we restrict ourselves to them in the analysis below.

Intuitively, deciding whether or not there exists a well-founded strategy for a catalog $\mathcal{C}$, which ends in at most M interactions, is a problem that can be solved by exploring recursively all possible transformations, and all active values for a feature in a slot filling or slot changing transformation. However complex, such an exhaustive exploration needs only polynomial space (the size of the catalog times the size of feature domains), and would prove that the problem of devising an optimal strategy is in PSPACE. We prove this intuition by first exhibiting Algorithm 1 below, and then analyzing its space complexity.

Theorem 3 (Correctness). *Given as input a catalog $\mathcal{C}$ of items, an integer M, and an initial state $U_0 + R(U_0)$, Algorithm 1 returns true if and only if there exists a well-founded strategy for the CRS which, starting from $U_0 + R(U_0)$, ends in at most M interactions.*

Proof. First of all, observe that Algorithm 1 terminates, since in every recursive call either the number M decreases, or the set $\mathcal{N}$ of disliked items enlarges, or both, leading necessarily to a base case. Once termination is proved, correctness of Algorithm 1 can be proved by induction on M. For the base cases $M \leq 0$ (Line 1), $|\mathcal{C} - \mathcal{N}| \leq M$ (Line 3), and user acceptance (Line 7), the result is obviously correct (see also comments aside base cases). Suppose now that the algorithm is correct for any state and interaction bound less than M.

To ease the proof presentation, note that the recursive part of Algorithm 1 can be divided in two branches: first one spans Lines 6–37, while second one spans Lines 38–48. We refer to these two branches in the rest of the proof.

"$\Rightarrow$" Suppose the algorithm terminates with *true* on input $U + R(U)$ and M. Either this result comes from the first branch, or from the second one; for both branches, in the lines marked with ($\exists$), the algorithm chose a feature f— and possibly a transformation in case of the first branch—such that for any

possible value chosen by the user (in the lines marked with ($\forall$)), all recursive calls[12] terminate with *true* on input $U' + R(U')$ (where U' depends on the value chosen by the user) and $m \in \{M-1, M-2\}$. By inductive hypothesis, each recursive call ensures the existence of a well-founded strategy for the CRS which, given $U' + R(U')$, ends in at most m interactions. Then, the overall strategy on input $U + R(U), M$ is the one that makes exactly the choice made in ($\exists$), followed by the strategies defined by the recursive calls.

"$\Leftarrow$" Suppose the algorithm terminates with *false*. This means that in either branch, for any possible choice (feature, transformation) made in ($\exists$)-lines, there is at least one possible choice (feature value v) the user can make in ($\forall$)-lines that makes at least one recursive call end with *false* on input $U' + R(U'), m \in \{M-1, M-2\}$. By inductive hypothesis, this means that when the user chooses v, there is no strategy starting from $U' + R(U')$ that ends in at most m interactions. Hence there is no strategy starting from $U + R(U)$ and ending in M interactions, either. $\qquad\square$

Theorem 4. *Given as input a catalog $\mathcal{C}$ of items—with p features and at most K values in each feature domain—an integer M, and an initial cold-start state $U_0 + R(U_0)$, Algorithm 1 uses polynomial space.*

Proof. As in the previous proof, we refer to Lines 6–37 as "first branch", while "second branch" refers to Lines 38–48.

First of all, a single recursive call needs a counter of $\lceil \log p \rceil$ bits for enumerating features (either Line 18 or Line 39, depending on which branch) and a counter of $\lceil \log K \rceil$ bits for enumerating values in the chosen feature (lines immediately below the above ones). Hence, the space of a single recursive call is $O((\log p) \cdot (\log K))$.

A bound on the height of the call stack can be computed as follows: at most p nested calls can be made in the second branch—filling all p features in Q—after which a single item has been isolated, and the first branch must be taken. Each time the first branch is taken, at least one item is added to $\mathcal{N}$ (in the worst case), reducing the set of items $\mathcal{C} - \mathcal{N}$ that can be proposed. Clearly, at most $|\mathcal{C}|$ calls can recur into the first branch. Overall, the recursion

---

[12]Strictly speaking, when the transformation is a slot unfilling, there is no choice by the user, and just one recursive call returning *true*. In this case the inductive hypothesis on $U' + R(U'), M - 1$ directly proves that there is a strategy also for $U + R(U), M$.

stack height is bounded by $p \cdot |\mathcal{C}|$, so the total space occupied by the recursion stack is $O(p \cdot |\mathcal{C}| \cdot (\log p) \cdot (\log K))$.

Observe now that an item with p features (each feature containing one value among K possible ones), needs $p \cdot \lceil \log K \rceil$ bits of encoding, hence the input of a catalog of $|\mathcal{C}|$ items has size $n = |\mathcal{C}| \cdot p \cdot \lceil \log K \rceil$. With respect to such an input size, the stack space is $O(n \cdot \log p)$. Now from the product expressing n, it follows that $\log p < \log n$, hence $O(n \cdot \log p) \subseteq O(n \cdot \log n)$. This proves that Algorithm 1 uses polynomial space. $\qquad\square$

Combining Theorems 3 and 4, one obtains that the problem of deciding whether there exists a strategy for CRS terminating in a given number of interactions can be solved in Nondeterministic Polynomial Space (NPSPACE), and since NPSPACE = PSPACE [55], the problem belongs to PSPACE, too. Recall that $PSPACE \subseteq EXPTIME$, and in fact, it can be easily seen that Algorithm 1 runs in exponential time in the worst case.

4.3. Improving the upper bound

The above lower and upper bounds restrict the complexity of finding efficient strategies for CRS between NP (lower bound) and PSPACE (upper bound). Since the aim of this paper is to bridge CRS practical efficiency with theoretical results, we leave investigation on how to close this gap to future (more theoretically-oriented) research. On the practical side, our PSPACE upper bound ensures that usual techniques for solving NP-complete problems, and in particular, finding optimal DT—*e.g.,* Mixed-Integer Optimization [56], branch-and-bound search [57], SAT and MAXSAT encoding [58]— can be used for devising optimal conversational strategies, but only—as far as our result proves—as steps inside an overall recursive procedure, while our NP lower bound ensures that employing such techniques is not an overshoot.

To improve the upper bound, remark that when the user rejects a particular feature value—*e.g.,* "*I don't like green phones*", as taken into account in Line 11—this fact rules out all items sharing the same feature value, thus improving the worst case of the strategy. Let us call Protocol P1 the protocol in which the CRS does not ask for disliked values when an item is rejected (*i.e.,* Algorithm 1 never enters Line 11), and call Protocol P2 the one in which the CRS *always* asks for a disliked value when the user rejects an item, the user provides such a value, and the CRS discards all items sharing the same feature value.

Algorithm 1 ExploreStrategies($\mathcal{C}, U, M$)

Input: catalog $\mathcal{C}$ (fixed), conversation state $U = \langle Q, \mathcal{K}, \mathcal{P}, \mathcal{N} \rangle$, integer M

Output: $\begin{cases} true & \text{if there is a strategy whose number of interactions is} \leq M \\ false & \text{otherwise} \end{cases}$

1: **if** $M \leq 0$ **then return** $false$ ▷ Base case 1: there is no strategy with 0 interactions
2: **else**
3:      **if** $|\mathcal{C} - \mathcal{N}| \leq M$ **then return** $true$ ▷ Base case 2: trivial strategy, propose each item
4:      **else**$[|\mathcal{C} - \mathcal{N}| > M > 0]$
5:          **if** $Q(\mathcal{C} - \mathcal{N}) = \{I\}$ **then**   ▷ Q selects a singleton
6:              propose I to the user
7:              **if** I is accepted **then return** $true$ ▷ item found within M interactions
8:              **else**$[I$ is rejected$]$
9:                  let $\mathcal{N}' \leftarrow \mathcal{N} \cup \{I\}$
10:                  *"the user rejects I for a specific reason (value)?"*
11:                  **if** $\exists f_i$ s.t. $f_i(I) = v$ **and** the user dislikes v **then**
12:                      let $C_i' \leftarrow C_i \cup \{v\}$ ▷ v is added to disliked values
13:                      let $\mathcal{K}' \leftarrow \langle C_1, \ldots, C_i', \ldots, C_p \rangle$
14:                      $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{I' \mid v = f_i(I')\}$ ▷ add items with v to disliked items
15:                  **else**
16:                      $\mathcal{K}' \leftarrow \mathcal{K}$
17:                  **end if**   ▷ no more items to propose at this point, change Q
18:                  $(\exists)$ choose a feature f_i s.t. $f_i(Q) = v$ (f_i was filled in Q)
19:                  $(\exists)$ choose $\tau \in \{$ Slot Unfilling (SU), Slot Change (SC)$\}$
20:                  **if** τ is SU **then**
21:                      let $\tau(v) = x$ where $x \in X$ is a new variable
22:                      **return** ExploreStrategies($\mathcal{C}, \langle \tau(Q), \mathcal{K}', \mathcal{P}, \mathcal{N}' \rangle, M - 1$)
23:   ▷ -1 interaction: the rejection
24:                  **else**$[\tau$ is SC$]$ ▷ try all values but v and the disliked ones
25:                      $(\forall)$ let $tryAllSlotChanges \leftarrow true$
26:                      **for all** $v' \in D_i$ s.t. $v' \notin C_i \cup \{v\}$ **do**
27:                          let $\tau(v) = v'$
28:                          **if** $\tau(Q)(\mathcal{C} - \mathcal{N}) \neq \emptyset$ **then**   ▷ try only changes selecting some item
29:                              **if** ExploreStrategies($\mathcal{C}, \langle \tau(Q), \mathcal{K}', \mathcal{P}, \mathcal{N}' \rangle, M - 2$) **then**   ▷ -2 interactions:
rejection, and change value)
30:                                  returns $false$
31:                                  $tryAllSlotChanges \leftarrow false$ ▷ user might choose a value leading to $> M$
interactions
32:                              **end if**
33:                          **end if**
34:                      **end for**
35:                      **return** $tryAllSlotChanges$ ▷ $true$ only if any user's change leads to $\leq M$ interactions
36:                  **end if**
37:              **end if**
38:          **else**$[Q(\mathcal{C})$ selects more than one item$]$ ▷ ask the user to fill a slot that distinguishes items
39:              $(\exists)$ choose a feature f_i s.t. $f_i(Q) = x \in X$
40:              $(\forall)$ let $tryAllSlotFillers \leftarrow true$
41:              **for all** $v \in AV(Q(\mathcal{C}), f_i)$ **do**   ▷ try all active values occurring in items of $Q(\mathcal{C})$
42:                  let $\tau(x) = v$
43:                  **if** ExploreStrategies($\mathcal{C}, \langle \tau(Q), \mathcal{K}, \mathcal{P}, \mathcal{N} \rangle, M - 1$) **then** ▷ one interaction less (slot filling)
                     returns $false$
44:                      $tryAllSlotFillers \leftarrow false$ ▷ user might fill a value leading to $> M$ interactions
45:                  **end if**
46:              **end for**
47:              **return** $tryAllSlotFillers$ ▷ $true$ only if any user's fill leads to $\leq M$ interactions
48:          **end if**
49:      **end if**
50: **end if**

Intuitively, it seems that P2 is always more efficient than P1. We prove in what follows that, in fact, the difference between P1 and P2 carries over to an improvement in the upper bound of the efficiency problem, backing up intuition, but highlighting when such intuition fails. For this part, we recall θ-notation: for a function $f(n)$, we write $f(n) \in \theta(g(n))$ if both $f(n) \in O(g(n))$, and $f(n) \in \Omega(g(n))$, *i.e.*, $f(n)$ grows no faster than $g(n)$, but also there are infinitely many (worst) cases in which $f(n)$ grows as fast as $g(n)$. Recall that the relation "$\in \theta$" between two functions is symmetric and transitive[13].

Corollary 1. *Let the given data as in Theorem 4, and additionally, suppose the number of values of all feature domains be bounded by a constant K, and $|\mathcal{C}| \in \theta(K^p)$. Then deciding whether there is a well-founded strategy always terminating within M interactions, that applies Protocol P2, is a problem that belongs to POLYLOGSPACE.*

Proof. We prove membership in POLYLOGSPACE using the same Algorithm 1 above, plus the fact that in every iteration of the algorithm the condition in Line 11 is satisfied, as required by Protocol P2.

Observe that $|\mathcal{C}| \leq K^p$ since there cannot be more items than all possible p combinations of feature values. After $K - 1$ rejections in Line 11, each ruling out a value for a feature, the number of remaining items $|\mathcal{C} - \mathcal{N}|$ is bounded by $K^{(p-1)}$ (all but one possible values for a feature have been eliminated, hence that feature does not count in the exponent). When the values ruled out $v_{i_1}, v_{i_2}, \ldots$ belong each one to a different feature $f_{i_1}, f_{i_2}, \ldots$, the bound is a more complex product[14] than $K^{(p-1)}$, but in any case, after at most $p \cdot (K - 1)$ rejections and disliked values, all values but one have been ruled out for each feature domain. At that point, either the user accepts the last item—the one whose features contain the last remaining value in each domain—or the user rejects it, and in any case the conversation concludes. This means that now the stack height of Algorithm 1 is no more bounded by $p \cdot |\mathcal{C}|$ (where the factor $|\mathcal{C}|$ was the bound on possible rejections) as in the proof of Theorem 4, but is bounded by $p \cdot p(K - 1) = p^2(K - 1)$, and the

---

[13]Simmetry: $f(n) \in \theta(g(n)) \iff g(n) \in \theta(f(n))$. Transitivity: $f(n) \in \theta(g(n)), g(n) \in \theta(h(n)) \Rightarrow f(n) \in \theta(h(n))$

[14]For example, if the disliked values were 2 for the first feature, 5 for the third one, and 3 for the last feature, the expression would be the product of p factors $(K - 2) \cdot K \cdot (K - 5) \cdot K \cdots K \cdot (K - 3)$.

space needed by the call stack is $O(p^2(K-1) \cdot (\log p) \cdot (\log K)) = O(p^2 \log p)$ for constant K. We now prove that the above expression implies that space requirements drop to a polynomial in the *logarithm* of the input.

Observe that $|\mathcal{C}| \in \theta(K^p)$ implies by symmetry $K^p \in \theta(|\mathcal{C}|)$, that is, $p \in \theta(\log |\mathcal{C}|)$ when K is a constant. Hence, by transitivity the number of nested recursive calls is in $\theta(\log^2 |\mathcal{C}|)$, and the space needed by Algorithm 1 drops to $\theta((\log^2 |\mathcal{C}|)(\log \log |\mathcal{C}|)) \subset O(\log^3 |\mathcal{C}|)$. The size of the input is $n = |\mathcal{C}| \cdot p \cdot \lceil \log K \rceil$, which belongs to $\theta(|\mathcal{C}| \cdot \log |\mathcal{C}|)$ in the hypotheses of this theorem, hence $\log n \in \theta(\log |\mathcal{C}| + \log \log |\mathcal{C}|) = \theta(\log |\mathcal{C}|)$. Summing up, the space needed by nondeterministic Algorithm 1 is in $O(\log^3 n)$, which, by Savitch's theorem [55], implies that the problem complexity now drops to $DSPACE(\log^6 n)$ (problems solvable by a deterministic algorithm that uses $O(\log^6 n)$ space), which is strictly included in POLYLOGSPACE. $\square$

However complex the class $DSPACE(\log^6 n)$ might seem, observe that the upper bound was exponentially reduced with respect to PSPACE. Moreover, from the well-known relation $POLYLOGSPACE \subseteq DTIME(2^{\lceil \log n \rceil^{O(1)}})$, (problems solvable by a deterministic algorithm whose time is bounded by a subexponential function, a class which is also known as *Quasi-Polynomial time (QP)*), we note that also time upper bounds provably improve over the previous EXPTIME ones[15].

Hence, the above result backs up theoretically CRS implementations whose strategies enforce Protocol P2. However, observe that the above result holds only in the situation in which the input catalog has a number of items that grows as fast as K^p—i.e., items are "dense" in the space of features—and K is a constant, hence the input increases because more features $f_{p+1}, f_{p+2}, \ldots$ are added—i.e., more domains $D_{p+1}, D_{p+2}, \ldots$ are added— not because each domain D_i of each feature value increases its cardinality. If, instead of considering K as a constant, we would consider p as a constant, the improvement on the upper bound due to Protocol P2 would not theoretically show up. In other words, when the number p of features is fixed, and the catalog increases because more possible values are added to each feature, asking to rule out such *values* one by one (as in Protocol P2) is not provably more efficient than asking to rule out *items* one by one (as in Protocol P1).

Hence, our theoretical analysis points out which characteristics of the cat-

---

[15]Observe that the general problem remains NP-hard, though, because the additional hypotheses of this corollary are not met by the reduction in Theorem 1.

alog must be evaluated when implementing efficient strategies for a CRS: if the catalog is composed by items with few features, and lots of possible values in each feature, there is not yet any theoretical evidence that Protocol P2 yields a tangible increase in efficiency with respect to Protocol P1; if instead the catalog consists of items described by lots of features, each feature with few values, then P2 improves efficiency over P1 in a complexity-theoretical provable way. In the next section, we provide an evidence of such a difference in a practical setting. Namely, we set up two catalogs IS1, IS2—both derived from real data on movie recommendation—with the characteristics described above, and simulate conversations with users—whose preferences are taken from real user preferences—testing both protocols. The results confirm the theoretical expectations, namely, that the additional steps of Protocol P2 really pay off only for the second catalog, IS2.

5. Experimental Evaluation

The goal of our experimental evaluation is to demonstrate that the two alternatives in Algorithm 1—that is, entering or not Line 11—lead to different results in terms of the efficiency of a CRS, depending on the characteristics of the catalog. More specifically, a user can take two distinct actions when she rejects a recommendation, and each action corresponds to one of the above-defined protocols:

P1 - the user rejects the recommendation and the CRS does not ask the user to provide a specific reason, *i.e.,* a reason that refers to a disliked feature value. Examples of such more unspecific feedback—if any feedback is given at all—could be, "*I don't want to go to the Blue Smoke restaurant*" (maybe because a friend of mine reported privately to me a bad impression), or "*I don't want to see the movie American Beauty*" (because the title sounds silly to me, but I cannot explain this to a system);

P2 - the user rejects the recommendation and the CRS asks for a specific item characteristic (*i.e.,* feature value) she does not like at all. For example, green color for cellphones, sea-view restaurants, a particular movie director, etc. We assume that a user will truthfully answer such a question when asked.

We will describe the details and assumptions of how we simulate the conversations below.

5.1. Research Hypotheses

We formulate two hypotheses for our experiment aiming to confirm the theoretical analysis provided at the end of Section 4.3.

H1 We do *not* expect a strong difference in terms of efficiency between P1 and P2 when the items in the catalog have few features with a large number of distinct values.

H2 We *do* expect a strong difference in terms of efficiency between P1 and P2 when the items in the catalog have several features with few distinct values (*i.e.,* IS2).

Basically, the only difference between H1 and H2 is the item catalog, which is the independent variable in our experiment.

5.2. Experimental setting

We designed an offline experiment that simulates the two above-mentioned interaction strategies (*i.e.,* protocols) within a *system-driven* model: the *CRS* asks, the user answers. Examples of corresponding dialogs are outlined in Section 5.3.

5.2.1. Evaluation Metric

In order to evaluate interaction efficiency we count the number of questions (*NQ)* the *CRS* asks before the user accepts a recommendation. A smaller number of required questions indicates higher interaction efficiency. This metric, as mentioned above, is commonly used in the CRS literature [37].

5.2.2. Dataset and Catalog Description

In our experiments, we rely on the MovieLens-1M dataset, which is widely used in recommender-system research, and which contains movie ratings by a larger user community[16]. The statistics of the dataset are given in Table 3. Unfortunately, in this dataset the items do not have textual features associated to them. Therefore, we mapped the MovieLens dataset with DBpedia resources to obtain side information for each item in the catalog. To close this gap we follow the approach of Anelli *et al.* [59].

---

[16]https://grouplens.org/datasets/movielens/

Table 3: Dataset characteristics

Dataset	users	items	ratings	content features
Movielens-1M	6,040	3,706	1,000,209	-
Movielens-1MC (with content)	6,040	3,308	511,260	279

After this processing, since not all the items have a mapping to the knowledge graph, the number of items became 3,308, and the total number of ratings after the mapping is 511,260. Through the mapping, 279 features were introduced into the dataset. Each item has one or more of these features expressed by a triple: ⟨*Item, Feature, Value*⟩. As an example, consider the movie *The Hateful Eight* directed by *Quentin Tarantino*. For this movie, we have the following triple: ⟨*The Hateful Eight, director, Quentin Tarantino*⟩. We denoted this data set as *MovieLens-1MC - with content* in Table 3.

In order to prove our hypotheses, we built two different itemsets from MovieLens-1MC:

- *Itemset1 (IS1)* has only a few features, but with a larger number of distinct values and is designed to prove H1 (we do not expect a strong difference in terms of efficiency between P1 and P2 *when the items in the catalog have few features with a large number of distinct values*).

- *Itemset2 (IS2)*, in contrast, has a larger number of features, but each of them only has a few distinct values and is designed to prove H2 (we expect a strong difference in terms of efficiency between P1 and P2 *when the items in the catalog have several features with few distinct values*).

Specifically, to make the datasets sufficiently different and that reflect the characteristics described in our research hypotheses, IS1 and IS2 have 4 and 10 features respectively (selected from the 279 content features of Movielens-1MC) for each item. For each feature of IS1 there are from 1,500 to 2,500 distinct values, whereas there are about 100 for IS2. The characteristics of the two itemsets are reported in Table 4.

In order to focus on the main goal of our experiment—to provide evidence that the efficiency of different strategies depends on itemset characteristics—we make the same assumptions as in Section 3 that *(i)* values for all features exist and *(ii)* that all features have singleton values. Accordingly, we replaced

Table 4: Itemset characteristics. Each value indicates the number of distinct value belonging to each features

	director	starring	producer	writer	distributor	musicComposer	language	narrator	basedOn	country
Itemset1	1542	2293	1587	1981	-	-	-	-	-	-
Itemset2	100	100	100	100	99	100	99	100	100	65

`null` values with a value randomly chosen from the set of possible values for that feature, and set-valued features (*e.g.,* the movie cast) with a single value randomly chosen among them. Note that for the purpose of our experiment, this replacement does not effect the final result.

5.2.3. Experimental Protocol—Details

In our experiment, we simulate conversations between a user and a CRS where this latter drives the interaction by asking questions to the user about preferred item features and by making recommendations for items. We assume that the simulated user has certain pre-existing preferences regarding item features, and truthfully responds to the system's questions about these preferences. When provided with a recommendation, the user either rejects it, which means that the dialog continues, or accepts it and the dialog ends. The CRS in our simulation implements one of the described conversation strategies, P1 or P2. Remember that—depending on the catalog characteristics—we expect to see differences in terms of how many interaction steps are necessary before the dialog successfully ends.

For the sake of the simulation, we assume that *in each conversation* there is exactly one distinguished item $\hat{I}$ in the catalog C that the user will accept, which we call *test* item or *ideal* item. This item $\hat{I}$ matches all preferences of the user (even if it is not the only item in C that matches the pre-existing general user preferences). Remember here the pre-existing user preferences consist of a number of desired value for features. With respect to the *genre* feature, a user might for example have a preference for *action* and *romantic* movies. When asked by the system for a particular genre preference, the user might therefore state one of the two values. During the conversation, the recommender will only consider such a *stated preference* when retrieving suitable items $\mathcal{S}$ (as defined in Section 4.2).

In our experiment, we aim to simulate a realistic situation where users do not know before the recommendation process which item (movie) they want *exactly*. Translated to our setting, this means that the user does not know about $\hat{I}$ or its features *in advance*, but is able to tell that she accepts

it once it is presented to her based on the stated preferences. As a result, the conversation with the system might often reach a dead end, *i.e.,* where the system does not recommend $\hat{I}$ and the user therefore has to revise their *stated* preferences. In our example, the user might have stated to prefer *action* movies when asked by the CRS, even if the ideal item $\hat{I}$ actually is a romantic movie. Hence, the user has to revise the stated preference at some stage so that the ideal item can be recommended by the CRS.[17] Please note that in our experiment we simulate a user cold-start situation, *i.e.,* we are not taking any long-term user profile into account the during the dialog. Without loss of generality, to avoid the introduction of further notation, in this section we will use f_i to refer to the feature selected at the i-th step of the dialog. We also remember that during the dialog, features are selected randomly.

In order to determine the pre-existing user preferences and the ideal items based, we select a set of positively-rated items (PRI) for each user. This set consists of those items that the user has rated with a value that is greater or equal to their average rating. We use this set PRI for two purposes. First, we simulate a dialog for each element I of PRI as an ideal item. Second, we use the items in PRI to determine the pre-existing preferences of a user and simulate their answers to the questions posed by the system. The user preferences on the feature f_i are then encoded in the set $UP_i = \bigcup\{v = f_i(I) \mid I \in \text{PRI}\}$ (for *U*ser *P*references). Therefore, if the user previously liked *action* and *romantic* movies, the set of pre-existing preferences for that feature contain exactly these two values. Remember that since we want the dialog to be successful only in case it ends with the acceptance of $\hat{I}$, for each conversation the catalog consists of $\mathcal{C}' = (\mathcal{C} - \text{PRI}) \cup \hat{I}$. We remove all the items in PRI but $\hat{I}$ from the catalog $\mathcal{C}$ since we do not want to recommend items the user has already seen before.

When the simulated dialog with a defined ideal item $\hat{I}$ starts, the system will ask a question on a feature *e.g.,* "*What is your favorite genre?*". The simulated user will then respond by choosing a value from the set $UP_i \cap AV(\mathcal{S}, f_i)$, where AV is the set of *Active Values* (see Section 4.2). In our example, $f_i = genre$. By definition, this set of values therefore changes

---

[17]Note that the chosen protocol will by design lead to a high number of required interactions, in particular as make the specific assumption for our simulation that only one item is acceptable in each dialog.

during the interaction as it is obtained through the intersection between all the pre-existing preferences for that feature (UP_i) and the possible values in the set of recommendable items $\mathcal{S}$. As a consequence, in our simulation the user cannot answer with a value that is not present in any recommendable object in $\mathcal{S}$.

After each user answer, the set of recommendable items $\mathcal{S}$ is updated by the CRS. If the user, for example, answered to prefer action movies, all movies with a different genre are not recommendable anymore at the next stage of the dialog and are then removed from $\mathcal{S}$. We remember that the user does not know about the ideal item I in this session nor its features. Therefore, the choice of the user answer $v \in UP_i \cap AV(\mathcal{S}, f_i)$ is done in a randomized manner in our experiment. This also avoids the introduction of any bias or any optimization.

A recommendation is shown when the system has no more questions to ask. This may happen in two cases:

a) The system has asked the user p questions, with p being the number of features;

b) After the user's answer to the q-th question, with $q < p$, we have $|\mathcal{S}| = 1$. In this case, the system recommends the only item in $\mathcal{S}$ without asking any explicit preference for the features f_i —with $q < i \le p$— to the user. In other words, the systems does not try to elicit preferences for the remaining features that have not been asked yet.

The user rejects the recommendation when I is not present in the list of recommended items. If the recommendation is rejected, the recommended items are removed from the catalog $\mathcal{C}'$ and the system starts again posing questions to the user. It is noteworthy that, since we may have more than one item in $\mathcal{C}'$ described by the feature values selected by the user during the dialog, the final recommendation may contain also a set of items.

In case of rejection P1 and P2 behave differently. In case of P1, the system starts again from a first randomly selected feature and discards all the values previously selected for each f_i by the user during the previous interactions. In protocol P2, in contrast, the user declares one of the feature values she *dislikes* in the recommended items. Once the user declares she dislikes a value v for a feature f_i, the system keeps all the previous choices in the sequence of features $f_1, \ldots, f_{i-1}$ for the current dialog and starts asking questions for f_i and the following ones $f_{i+1}, \ldots, f_p$. We cannot keep the value previously

selected in the sequence of features $f_{i+1}, \ldots, f_p$ for the current dialog since after the user selects a new value $v' \in UP_i \cap AV(\mathcal{S}, f_i)$ for f_i, the set $\mathcal{S}$ changes and, as a consequence, the same happens to $UP_{i+1} \cap AV(\mathcal{S}, f_{i+1})$ for f_{i+1} (and the remaining features). Regarding feature f_i, the user is allowed to select values from $UP_i \cap AV(\mathcal{S}, f_i) - v$. We observe that protocol P2 always enters Line 11 in Algorithm 1.

On the other side, when the recommendation succeeds —when the ideal item I is in the list of recommendations— the dialog is successfully ended and the simulation continues with a new dialog for another user and/or target item. The simulation ends when a dialog was simulated for each element in PRI for every user.

5.3. Running Examples

We show the main differences between the two protocols defined above using two fictitious dialogs.

Example 1 (Protocol 1 - The user rejects a recommendation and the item is discarded from the dataset.)**.**

System: *Tell me a film score composer you like, I have these options: Ennio Morricone, Alessandro Alessandroni, Paul Sawtell, Frank Skinner.*

User: *I like Ennio Morricone.*

System: *Who is one of your most liked actors, in your opinion among Leonardo Di Caprio, Clint Eastwood, Gian Maria Volonté?*

User: *I like Ennio Morricone.*

System: *Do you like Quentin Tarantino as film director?*

User: *Yes, I like Quentin Tarantino.*

System: *Based on your answers, I recommend you Django Unchained. What do you think about it?*

User: *Sorry, I don't want it—a friend of mine spoiled the plot for me.*

The recommendation failed. *Django Unchained* is removed from the set of recommendable items and the conversation goes on. After several failures the conversation may continue as in the following conversation.

System: *Who is one of your most loved film score composers among Piero Umiliani, Armando Trovajoli, Piero Piccioni?*

User: *I like Piero Umiliani.*

System: *Do you like Mario Monicelli as film director?*

User: *Yes, I do. I like Mario Monicelli.*

System: *For the above choices, I have movies of the following genres: Comedy, Neorealistic, Historical. Which one would you prefer?*

User: *Let's try comedy movies.*

System: *Which actor/actress would you prefer most among Vittorio Gassman, Vittorio De Sica, Sophia Loren?*

User: *I like Vittorio Gassman.*

System: *Based on your answers, I recommend Big Deal on Madonna Street. What do you think?*

User: *Great, I'll try this one!*

The recommendation is successful. The dialog is over.   □

Example 2 (Protocol 2 - The user rejects a recommendation and the CRS asks to declare a disliked feature.)**.**

System: *What kind of genre do you prefer among Western, Historical and Love Story?*

User: *I would like to explore Western movies.*

System: *Who is one of your favorite film score composers? I have these options: Ennio Morricone, Alessandro Alessandroni, Paul Sawtell, Frank Skinner.*

User: *I like Ennio Morricone.*

System: *Who is one of the best actor among Leonardo Di Caprio, Clint Eastwood, Gian Maria Volonté and Tim Roth in your opinion?*

User: *I like Leonardo Di Caprio.*

System: *Do you like Quentin Tarantino as film director?*

User: *I like Leonardo Di Caprio.*

System: *Based on your answers I recommend Django Unchained. Do you think you'll like it?*

User: *No, I don't.*

System: *With the choices you gave me, I have no more matching movies. Would you mind reconsidering one of your stated preferences?*

User: *Ok, let's forget about Leonardo Di Caprio. I also like Tim Roth.*

At this point the system discards movies by *Leonardo Di Caprio* and concentrates on movies starring *Tim Roth*.

System: *Do you like Robert Richardson as director of photography?*

User: *Sure, I know Robert Richardson as a good director of photography.*

System: *Would you prefer movies that have won an Oscar?*

User: *Yes, I do.*

System: *Based on your answers, I recommend The Hateful Eight. Will you try it out?*

User: *Sounds good, I will.*

The recommendation is successful. The dialog is over. □

5.4. Results and Discussion

In this section we show the experimental results. For each itemset (*i.e., I1* and *I2*) we compared the two protocols (*i.e., P1* and *P2*) and we counted the Number of Questions (NQ) required for reaching the test item $\hat{I}$. Figures 1a and 1b report the average number of questions required to reach a test item (the average is computed on all the configurations), and the overall maximum number of questions for each protocol and itemset.

(a) Average number of Questions per Configuration

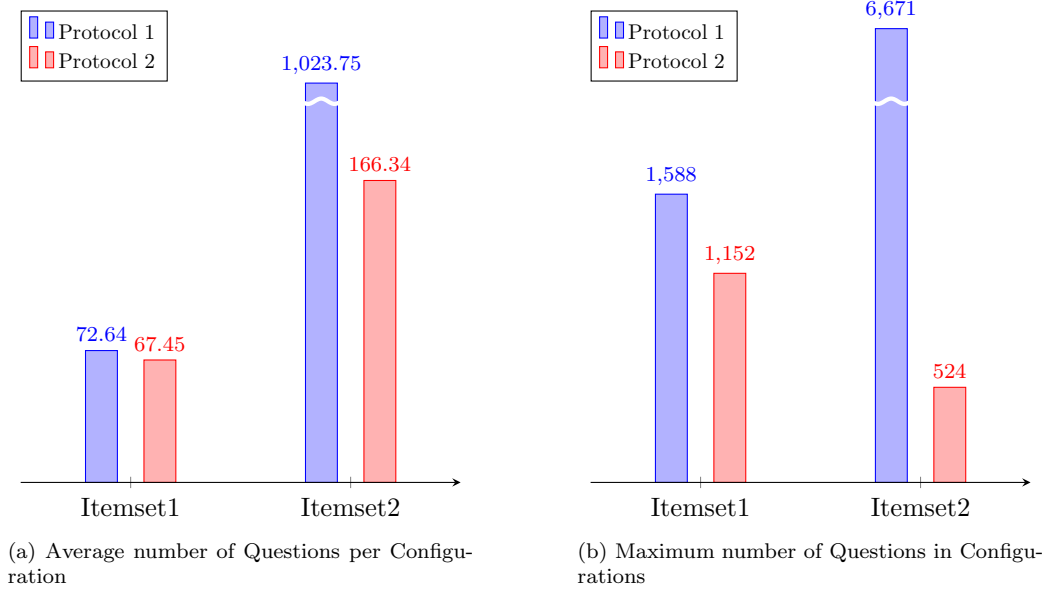(b) Maximum number of Questions in Configurations

Figure 1

As expected from the theoretical analysis, with IS1 we observe minor differences between the two protocols in terms of number of questions made by the system before reaching and recommending the test item. More specifically, P1 needs 72.64 as average number of questions with IS1, whereas P2 needs 67.45 questions. The difference is however huge for IS2 where P1 needs more than 1,000 questions on average to reach the test item, while P2 needs around 166 questions. Hence, we can confirm that when the items in the catalog have several features with few distinct values, the efficiency of P2 grows drastically compared to P1. Also *maximum* number of questions—reported in Figure 1b for each pair of protocol and itemset—confirms this different efficiency for IS1 and IS2.

We now discuss the fact that NQ is very large—indeed, also in the best combination (P2 with IS1), the number of questions is close to 70, which sounds unrealistic for any real user. It is worth noting that in this experiment we implemented the very worst-case scenario. Our experiment used an unrealistic setting on purpose. In our particular scenario the recommendation task is deliberately very difficult:

- for each dialog, there is only one test item (true positive) to be found in the catalog and we do not consider the usual possibility that the user

would accept several items;

- the CRS works in cold-start condition without any user profile, thus several questions are needed for acquiring the user preferences;

- the CRS does not implement a cut-off on the number of questions to ask the user.

In conclusion, the results of our experimental evaluation are aligned with our theoretical findings and we can thus confirm both the research hypotheses H1 and H2.

In other words, our simulation confirmed what was foreseen by the theoretical analysis, namely that the difference between protocol P1 and protocol P2 shows up clearly only in a dataset in which there are many features with a small set of different values.

6. Summary and Outlook

While conversational approaches to recommendation have been explored since the late 1990s, we observe an increasing interest in these types of system in recent years in particular due to recent developments in natural language processing. Independent of the interaction modality, efficiency in terms of required dialog turns has commonly been a main target of research. So far, however, questions of efficiency were almost exclusively investigated in an empirical manner. With this work, we contribute to a better understanding of theoretical properties of conversational recommendation problems and we specifically address questions related to the computational complexity of finding efficient dialog strategies. One main insight of our theoretical analysis—which was also confirmed by a computational experiment—is that when designing an efficient conversation strategy, we must always consider the characteristics of the item catalog. On a more general level, we hope that our work helps to stimulate more theory-oriented research in this area which leads us to a better understanding of foundational properties of an important class of interactive AI-based systems.

Regarding potential future works, remember that the problem formalization chosen in our work targets at the predominant class of CRS in the literature—systems that interactively elicit user preferences regarding item features—and covers a set of common interaction types in these types of

systems. For the purpose of the current study, we however limited our formalization to two basic types of providing positive or negative preferences: *(a)* by providing information about individual feature values, and *(b)* by providing feedback for individual items as a whole. In our future work, we plan to extend the formalization to allow users to state preferences on combination of feature values, *e.g., "I don't like open-air Japanese restaurants"*, or *"I like ocean-view, romantic Italian restaurants"*. Other simplifications in our current work that we will address in the future include the consideration of multi-valued features, both in the queries and the item catalog, and situations where individual item features might be unknown.

Finally, we see another and potentially more far-reaching direction for future research in the explicit consideration of individual long-term user preferences in the interactive recommendation process. Technically, this may for example be implemented by the introduction of a function that predicts, based on past behavior of a user, if she will be satisfied by a certain item I in the ongoing conversation. The corresponding goal of CRS development would then be to devise efficient strategies that drive the user towards those items that the system assumes she will most probably choose, again with the aim of optimizing the efficiency of the process.

References

[1] D. Jannach, A. Manzoor, W. Cai, L. Chen, A survey on conversational recommender systems, ACM Comput. Surv. (2021).

[2] L. Chen, P. Pu, Critiquing-based recommenders: survey and emerging trends, User Modeling and User-Adapted Interaction 22 (1-2) (2012) 125–150.

[3] D. Jannach, Advisor Suite – A knowledge-based sales advisory system, in: ECAI '04, 2004, pp. 720–724.

[4] K. Christakopoulou, F. Radlinski, K. Hofmann, Towards conversational recommender systems, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 815–824.

[5] Q. Chen, J. Lin, Y. Zhang, M. Ding, Y. Cen, H. Yang, J. Tang, Towards knowledge-based recommender dialog system, in: EMNLP-IJCNLP '19, 2019, pp. 1803–1813.

[6] J. Zhang, P. Pu, A comparative study of compound critique generation in conversational recommender systems, in: AH '02, 2006, pp. 234–243.

[7] J. Reilly, K. McCarthy, L. McGinty, B. Smyth, Dynamic critiquing, in: ECCBR 04, 2004, pp. 763–777.

[8] T. Mahmood, F. Ricci, Improving recommender systems with adaptive conversational strategies, in: HT '09, 2009, pp. 73–82.

[9] R. Reichman, Getting computers to talk like you and me, MIT Press, Cambridge, Massachusetts, 1985.

[10] D. Jannach, G. Kreutler, et al., Rapid Development of Knowledge-based Conversational Recommender Applications with Advisor Suite, J. Web Eng. 6 (2) (2007) 165–192.

[11] K. McCarthy, J. Reilly, L. McGinty, B. Smyth, Thinking positively-explanatory feedback for conversational recommender systems, in: Proceedings of the European Conference on Case-Based Reasoning (ECCBR-04) Explanation Workshop, 2004, pp. 115–124.

[12] L. Chen, P. Pu, Preference-based organization interfaces: aiding user critiques in recommender systems, in: International Conference on User Modeling, Springer, 2007, pp. 77–86.

[13] P. Viappiani, P. Pu, B. Faltings, Conversational recommenders with adaptive suggestions, in: RecSys '07, 2007, pp. 89–96.

[14] P. Pu, P. Viappiani, B. Faltings, Increasing user decision accuracy using suggestions, in: CHI '06, 2006, p. 121–130.

[15] N. Mirzadeh, F. Ricci, M. Bansal, Feature selection methods for conversational recommender systems, in: EEE '05, 2005, pp. 772–777.

[16] H. Shimazu, ExpertClerk: A conversational case-based reasoning tool for developing salesclerk agents in E-Commerce webshops, Artificial Intelligence Review 18 (3-4) (2002) 223–244.

[17] S. R. Chakraborty, M. Anagha, K. Vats, K. Baradia, T. Khan, S. Sarkar, S. Roychowdhury, Recommendence and fashionsence: Online fashion advisor for offline experience, in: CoDS-COMAD '19, 2019.

[18] Z. Yin, K.-h. Chang, R. Zhang, DeepProbe: Information directed sequence understanding and chatbot design via recurrent neural networks, in: KDD '17, 2017, pp. 2131–2139.

[19] Y. Sun, Y. Zhang, Conversational recommender system, in: SIGIR '18, 2018, pp. 235–244.

[20] D. Tsumita, T. Takagi, Dialogue based recommender system that flexibly mixes utterances and recommendations, in: WI '19, 2019, pp. 51–58.

[21] K. Christakopoulou, A. Beutel, R. Li, S. Jain, E. H. Chi, Q&R: A two-stage approach toward interactive recommendation, in: KDD '18, 2018, pp. 139–148.

[22] B. Loepp, T. Hussein, J. Ziegler, Choice-based preference elicitation for collaborative filtering recommender systems, in: CHI '14, 2014, pp. 3085–3094.

[23] F. Narducci, M. de Gemmis, P. Lops, G. Semeraro, Improving the user experience with a conversational recommender system, in: AI*IA '18, 2018, pp. 528–538.

[24] G. Carenini, J. Smith, D. Poole, Towards more conversational and collaborative recommender systems, in: IUI '03, 2003, pp. 12–18.

[25] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, J. Riedl, Getting to know you: Learning new user preferences in recommender systems, in: IUI '02, 2002, p. 127–134.

[26] X. Zhao, W. Zhang, J. Wang, Interactive collaborative filtering, in: CIKM '13, 2013, pp. 1411–1420.

[27] K. McCarthy, J. Reilly, L. McGinty, B. Smyth, On the dynamic generation of compound critiques in conversational recommender systems, in: AH '04, 2004, pp. 176–184.

[28] K. McCarthy, M. Salamó, L. Coyle, L. McGinty, B. Smyth, P. Nixon, Group recommender systems: A critiquing based approach, in: IUI '06, 2006, pp. 267–269.

[29] G. Adomavicius, Y. Kwon, New Recommendation Techniques for Multicriteria Rating Systems, IEEE Intelligent Systems 22 (3) (2007) 48–55. `doi:10.1109/MIS.2007.58`.

[30] P. Viappiani, C. Boutilier, Regret-based optimal recommendation sets in conversational recommender systems, in: RecSys '11, 2009, pp. 101–108.

[31] J. Reilly, J. Zhang, L. McGinty, P. Pu, B. Smyth, A comparison of two compound critiquing systems, in: IUI '07, 2007, pp. 317–320.

[32] F. Ricci, Q. N. Nguyen, O. Averjanova, Exploiting a map-based interface in conversational recommender systems for mobile travelers, in: Tourism Informatics: Visual Travel Recommender Systems, Social Communities, and User Interface Design, IGI Global, 2010, pp. 73–79.

[33] P. Grasch, A. Felfernig, F. Reinfrank, Recomment: Towards critiquing-based recommendation with speech interaction, in: Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, ACM, New York, NY, USA, 2013, pp. 157–164.

[34] Y. Ikemoto, V. Asawavetvutt, K. Kuwabara, H.-H. Huang, Tuning a conversation strategy for interactive recommendations in a chatbot setting, Journal of Information and Telecommunication 3 (2) (2019) 180–195.

[35] Y. Jin, W. Cai, L. Chen, N. N. Htun, K. Verbert, MusicBot: Evaluating critiquing-based music recommenders with conversational interaction, in: CIKM '19, 2019, pp. 951–960.

[36] P. Wärnestål, User evaluation of a conversational recommender system, in: Proceedings of the 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, 2005, pp. 32–39.

[37] A. Iovine, F. Narducci, G. Semeraro, Conversational recommender systems and natural language:: A study through the converse framework, Decision Support Systems 131 (2020) 113250.

[38] W. Chen, H. Huang, S. T. Chou, Understanding consumer recommendation behavior in a mobile phone service context, in: ECIS '08, 2008, pp. 1022–1033.

[39] W. Wang, I. Benbasat, Research Note—A contingency approach to investigating the effects of user-system interaction modes of online decision aids, Information Systems Research 24 (3) (2013) 861–876.

[40] F. Pecune, S. Murali, V. Tsai, Y. Matsuyama, J. Cassell, A model of social explanations for a conversational movie recommendation system, in: HAI '19, 2019, p. 135–143.

[41] L. Chen, P. Pu, Evaluating critiquing-based recommender agents, in: AAAI '06, 2006, pp. 157–162.

[42] W. Cai, L. Chen, Predicting user intents and satisfaction with dialogue-based conversational recommendations, in: UMAP '20, 2020, p. 33–42.

[43] D. Jannach, Finding preferred query relaxations in content-based recommenders, in: Intelligent techniques and tools for novel system architectures, Springer, 2008, pp. 81–97.

[44] A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, Constraint-based recommender systems, in: Recommender systems handbook, Springer, 2015, pp. 161–190.

[45] N. Mirzadeh, F. Ricci, M. Bansal, Supporting user query relaxation in a recommender system, in: E-Commerce and Web Technologies, 2004, pp. 31–40.

[46] D. Jannach, Preference-based treatment of empty result sets in product finders and knowledge-based recommenders, in: 18th PUK Workshop - New Results in Planning, Scheduling, and Design (PUK 2004), 2004, pp. 55–69.

[47] A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, E. Teppan, Plausible repairs for inconsistent requirements, in: IJCAI 2009, 2009, pp. 791–796.

[48] S. Colucci, T. D. Noia, E. D. Sciascio, F. M. Donini, M. Mongiello, Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace, Electron. Commer. Res. Appl. 4 (4) (2005) 345–361.

[49] P. Gärdenfors, Belief revision and knowledge representation, in: Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge, 1996, p. 117.

[50] B. M. E. Moret, Decision trees and diagrams, ACM Comput. Surv. 14 (4) (1982) 593–623.

[51] L. Hyafil, R. L. Rivest, Constructing optimal binary decision trees is np-complete, Inf. Process. Lett. 5 (1) (1976) 15–17.

[52] I. Chikalov, S. Hussain, M. Moshkov, Totally optimal decision trees for boolean functions, Discrete Applied Mathematics 215 (2016) 1 – 13.

[53] D. Guijarro, V. Lavín, V. Raghavan, Exact learning when irrelevant variables abound, Inf. Process. Lett. 70 (5) (1999) 233–239.

[54] C. H. Papadimitriou, M. Yannakakis, A note on succinct representations of graphs, Inf. Control. 71 (3) (1986) 181–185.

[55] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. Syst. Sci. 4 (2) (1970) 177–192.

[56] D. Bertsimas, J. Dunn, Optimal classification trees, Mach. Learn. 106 (7) (2017) 1039–1082.

[57] G. Aglin, S. Nijssen, P. Schaus, Learning optimal decision trees using caching branch-and-bound search, in: IAAI 2020, 2020, pp. 3146–3153.

[58] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva, Learning optimal decision trees with SAT, in: Proceedings IJCAI 2018, 2018, pp. 1362–1368.

[59] V. W. Anelli, T. D. Noia, E. D. Sciascio, A. Ragone, J. Trotta, How to make latent factors interpretable by feeding factorization machines with knowledge graphs, in: Proceedings ISWC 2019, 2019, pp. 38–56.