The Ins and Outs of Critiquing

David McSherry[1] **and David W. Aha**[2]
[1]School of Computing and Information Engineering
University of Ulster, Coleraine BT52 1SA, Northern Ireland
dmg.mcsherry@ulster.ac.uk
[2]Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5515, Washington DC 20375, USA
david.aha@nrl.navy.mil

Abstract

Eliminating previously recommended products in critiquing limits the choices available to users when they attempt to navigate back to products they critiqued earlier in the dialogue (e.g., in search of cheaper alternatives). In the worst case, a user may find that the *only* product she is prepared to accept (e.g., having ruled out cheaper alternatives) has been eliminated. However, an equally serious problem if previous recommendations are *not* eliminated is that products that satisfy the user's requirements, if any, may be *unreachable* by any sequence of critiques. We present a new version of progressive critiquing that leaves open the option of repeating a previous recommendation while also addressing the *unreachability* problem. Our empirical results show that the approach is most effective when users refrain from *over-critiquing* attributes whose current values are acceptable.

1 Introduction

Critiquing in recommender systems is based on the idea that it is easier for users to critique an example product than to construct formal queries [e.g., Burke *et al*., 1997; Hammond *et al*., 1996; Linden *et al*., 1997]. An early example is *Entrée*, a restaurant recommender that supports both *directional* critiques (e.g., *Like this but cheaper*) and *replacement* critiques (e.g., *Like this with French food*) [Burke *et al*., 1997]. In approaches that combine critiquing with case-based reasoning (CBR), an initially recommended product may be retrieved on the basis of its similarity to an initial user query [Burke, 2002]. The product recommended in response to a critique is typically one that satisfies the critique and is maximally similar to the critiqued product.

Concern that basic critiquing algorithms often lead to protracted recommendation dialogues has prompted significant research interest in increasing the efficiency of critiquing [e.g., McCarthy *et al*., 2005; McSherry and Aha, 2006; Reilly *et al*., 2005]. However, approaches in which previously recommended products are eliminated implicitly assume that critiquing a recommended product amounts to *rejection* of the product by the user. We argue that this is not a realistic assumption and show that preventing users from navigating back to products they critiqued earlier in the dialogue may result in recommendation failure.

As we show in Section 2, an equally serious problem if previously recommended products are *not* eliminated is that products that satisfy the user's requirements, if any, may be *unreachable* by any sequence of critiques. The effectiveness of existing critiquing algorithms may thus be open to question *whether or not* previously recommended products are eliminated. To address this issue, we present a new version of progressive critiquing [McSherry and Aha, 2006] that leaves open the option of repeating a previous recommendation while also addressing the *unreachability* problem.

In Section 2, we use examples to illustrate the problems of *diminishing choices* and *unreachability* in critiquing on which we focus in this paper. In Section 3, we describe how these problems are addressed in progressive critiquing. In Section 4, we evaluate our approach on a well known case base and investigate the effects of user critiquing choices on dialogue outcomes and efficiency. Our conclusions are presented in Section 5.

2 Problems in Critiquing

The challenge of critiquing is to recommend an acceptable product based on information about the user's requirements that is both incomplete and imprecise. In a recommender system for personal computers, for example, a critique such as *Like this with a bigger screen* does not determine precisely the preferred screen size or the minimum size that the user might be prepared to accept. Another problem is that improvements in one attribute can often be gained only at the expense of another [Burke, 2002]. A related issue is the trade-off between preserving similarity to a critiqued product and recommending one that is sufficiently different in a single dimension to satisfy the user's requirements [Bridge and Ferguson, 2002; Burke, 2002; Salamó *et al*., 2005]. Avoiding long and fruitless dialogues when none of the available products are acceptable to the user is another important issue in critiquing [McSherry and Aha, 2006].

In any recommender system, the user's requirements may include constraints that must be satisfied and others that the user is prepared to relax if necessary. Below we describe how the user's critiques and constraints are modeled in our formal analysis of critiquing.

Definition 1. *We denote by Matches(Q) the possibly empty set of cases that satisfy a given set of constraints Q.*

It can be seen that:

$$Matches(Q) = \bigcap_{q \in Q} matches(q)$$

where for each $q \in Q$, $matches(q)$ is the set of cases that satisfy q.

In a critiquing system, the user is not required to express her requirements in a formal query. Instead, by critiquing the system's recommendations, the user provides *feedback* that the system uses to guide the search for an acceptable product.

Definition 2. *We denote by matches(r, C) the set of cases that satisfy a critique r when applied to a given case C.*

Definition 3. *We denote by critiques(C) the set of available critiques that are applicable to a given case C.*

A critique cannot be applied to a case that already satisfies the critique. That is, $C \notin matches(r, C)$ for any case C and $r \in critiques(C)$. If previously recommended cases are eliminated, a *critiquing failure* occurs if $matches(r, C) - E = \emptyset$, where E is the set of eliminated cases. If previously recommended cases are not eliminated, a critiquing failure can occur only if $matches(r, C) = \emptyset$. For example, a sequence of repeated *Like this but cheaper* critiques, if allowed by the system, must eventually result in a critiquing failure.

The *expressiveness* of a query language is an important issue in the retrieval of recommended products based on user queries [Bridge and Ferguson, 2002]. An equally important issue in critiquing is whether the user's constraints are expressible using the available critiques.

Definition 4. *We say that a constraint q is expressible using the available critiques if for any case $C \notin matches(q)$ there exists $r \in critiques(C)$ such that $matches(q) \subseteq matches(r, C)$.*

For example, *Between 3 and 5 bedrooms* is an expressible constraint in a property recommender that supports *more* and *less* critiques on bedrooms. An upper price limit (e.g., *price* ≤ £400) is expressible in a holiday recommender with a *Like this but cheaper* critique, while an equality constraint such as *month* = May is expressible using a replacement critique on *month*.

2.1 The Diminishing Choices Problem

It does not seem realistic to assume, as in critiquing algorithms that never repeat a previous recommendation, that critiquing a recommended product amounts to *rejection* of the product by the user. Shoppers often consider alternatives to a recommended product before deciding that it is the best available option. It is equally natural for a recommender system user to critique an acceptable product (e.g., in search of cheaper alternatives). But on attempting to retrace her steps (e.g., because cheaper alternatives do not meet her requirements), such a user may find her choices very limited if previously recommended products have been eliminated.

In the worst case, the user may discover that the *only* product she is prepared to accept has been eliminated. Table 1 shows an example case base in the residential property domain that we use to illustrate this problem of *diminishing choices* in critiquing. The case attributes are location (A or B), bedrooms (3 or 4), and property type (detached or semi-detached). The attributes are equally weighted and the similarity between two cases is the number of matching features.

	Loc	**Beds**	**Type**
Case 1	A	3	det
Case 2	B	4	det
Case 3	A	3	sem

Table 1. Example case base in the property domain.

For a critiquing algorithm in which previously recommended cases are eliminated, Figure 1 shows an example critiquing dialogue in which the user is looking for a 4 bedroom detached property in location A but is prepared to compromise on bedrooms. Her initial query is loc = A, and the initially recommended case is Case 1. The user can see from the response to her first critique that no case satisfies all her requirements (A 4 det). But when she tries to navigate back to Case 1 by critiquing Case 2 on location, the case now recommended is Case 3, even though it is less similar to Case 2 than Case 1. In this situation, it is difficult to see how the elimination of Case 1 (i.e., the only acceptable case) can be justified.

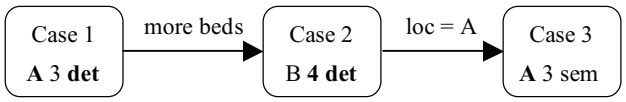

Figure 1. Example critiquing dialogue in which the user is unable to navigate back to the only acceptable case.

Thus eliminating previously recommended products is bound to result in recommendation failure if the user critiques the only acceptable product (or products). Even in a product case base of realistic size, it is not unusual for only a small number of acceptable products to be available if the user has several requirements that must be satisfied [McSherry, 2003]. The diminishing choices problem is thus a potentially serious drawback of critiquing approaches in which previous recommendations are eliminated.

However, Theorem 1 shows that when previously recommended cases are eliminated, a case that satisfies a given set of expressible constraints can always be reached, without critiquing failures, if such a case exists. In practice, whether such a case is reached without critiquing failures may of course depend on the user's critiquing choices. It is also possible in practice for a case that satisfies all the user's constraints to be reached without any critiques (i.e., the initially recommended case may satisfy all her constraints).

Theorem 1. *Eliminating previously recommended cases in critiquing ensures that a case that satisfies a given set Q of expressible constraints can always be reached without critiquing failures if such a case exists.*

Proof. Let $C^* \in Matches(Q)$ be any case that satisfies all the constraints in Q. If $C_1 \notin Matches(Q)$, where C_1 is the initially recommended case, then there exists $q_1 \in Q$ such that $C_1 \notin matches(q_1)$. As all the constraints in Q are expressible using the available critiques, there exists $r_1 \in critiques(C_1)$ such that $matches(q_1) \subseteq matches(r_1, C_1)$. As $C_1 \notin Matches(Q)$, $C^* \in Matches(Q) = Matches(Q) - \{C_1\} \subseteq matches(q_1) - \{C_1\} \subseteq matches(r_1, C_1) - \{C_1\}$.

If $C_2 \notin Matches(Q)$, where C_2 is the case recommended in response to r_1, then there exists $q_2 \in Q$ such that $C_2 \notin matches(q_2)$ and $r_2 \in critiques(C_2)$ such that $matches(q_2) \subseteq matches(r_2, C_2)$. As $C_1, C_2 \notin Matches(Q)$, $C^* \in Matches(Q) = Matches(Q) - \{C_1, C_2\} \subseteq matches(q_2) - \{C_1, C_2\} \subseteq matches(r_2, C_2) - \{C_1, C_2\}$. It follows that $matches(r_2, C_2) - \{C_1, C_2\} \neq \emptyset$, and so r_2 is bound to succeed despite the elimination of C_1 and C_2.

If $C_3 \notin Matches(Q)$, where C_3 is the case recommended in response to r_2, we can continue as long as necessary to construct a sequence of recommended cases $C_1, C_2, ..., C_k$ and successful critiques $r_1, r_2, ..., r_k$ such that $C_1, C_2, ..., C_k \notin Matches(Q)$. As previously recommended cases are eliminated, $C_1, C_2, ..., C_k, C_{k+1}$ are distinct cases, where C_{k+1} is the case recommended in response to r_k. As the supply of distinct cases $C \notin Matches(Q)$ must eventually be exhausted, it must eventually be true that $C_{k+1} \in Matches(Q)$. □

2.2 The Unreachability Problem

Table 2 shows a second example case base in the property domain that we use to illustrate the unreachability problem in critiquing. In the example dialogue that we now present, the initially recommended case is Case 1 (A 3 sem 2) and the user is looking for a 4 bedroom detached property in location A with 3 reception rooms (A 4 det 3). If all the user's requirements must be satisfied, then Case 5 is the only acceptable case. Attributes in the case base are equally weighted and the similarity between two cases is the number of matching features. We assume a basic critiquing algorithm in which:

- The available critiques are more and less critiques for beds and reception rooms (RRs) and replacement critiques for loc and type
- Each critique is applied only to a single attribute
- The single case recommended in response to a successful critique is one that is maximally similar among those that satisfy the critique (i.e., no account is taken of any previous critiques or initial query)
- Previously recommended cases are not eliminated

For such a critiquing algorithm, Figure 2 shows the cases recommended in a sequence of four critiques. After three critiques, the user has made no progress towards her goal, and her fourth critique brings her back to where she started. The user can do no better by critiquing Case 1 on *beds* or *RRs* or by choosing different critiques on Cases 2-4. She cannot reach Case 5 even by resorting to critiques that are inconsistent with her requirements, as none of the *negative* critiques she might choose (e.g., *less beds* than Case 3) are satisfied by Case 5.

	Loc	Beds	Type	RRs
Case 1	A	3	sem	2
Case 2	B	3	det	2
Case 3	B	4	sem	2
Case 4	B	3	sem	3
Case 5	A	4	det	3

Table 2. Example case base in which an acceptable case may be unreachable from any other case.

To confirm that Case 5 is unreachable with any of Cases 1-4 as the initially recommended case, it suffices to observe that Case 5 cannot be reached in a single step from any of the other cases. So the best the system can do with any of Cases 1-4 as the initially recommended case is to recommend a case that satisfies only *one* of the user's requirements.

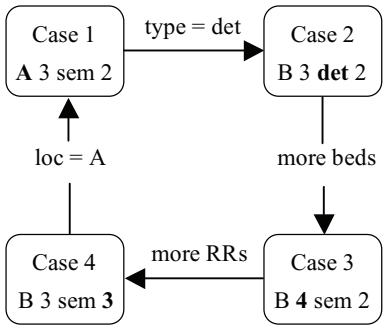

Figure 2. Example critiquing dialogue in which the only acceptable case cannot be reached.

At the expense of susceptibility to the diminishing choices problem (Section 2.1), eliminating previously recommended products in the example dialogue would result in

Case 5 being recommended in response to the user's fourth critique. Another problem highlighted by our example is that recommended cases in a critiquing dialogue may involve *unnecessary* compromises with respect to previous critiques that may frustrate or even mislead the user, and delay progress towards her goal. Given the response to her second critique, for example, the user may begin to doubt the availability of a detached property with 4 bedrooms.

3 Progressive Critiquing

Progressive critiquing differs from other critiquing algorithms in that a recommended product must, if possible, satisfy all previous critiques as well as the current critique [McSherry and Aha, 2006]. This strategy plays an important role in the algorithm's ability to address the unreachability problem when, as in the version we now present, previously recommended products are not eliminated.

3.1 Taking Account of Previous Critiques

While improvements in one attribute can often be gained only at the expense of another [Burke, 2002], we have seen in Section 2 that recommended cases may involve *unnecessary* compromises with respect to previous critiques. In progressive critiquing, the case recommended in response to the user's current critique is required, if possible, to satisfy all previous critiques as well as the current critique. That is, if r_1, ..., r_k is a sequence of critiques applied to recommended cases C_1, ..., C_k then the case C_{k+1} recommended in response to r_k is one that is maximally similar among the available cases C, if any, such that:

$$C \in \bigcap_{1 \le i \le k} matches(r_i, C_i).$$

The non-existence of a case that satisfies all the user's critiques is recognized as a *progression* failure. In the event of such a failure, the recommended case is one that is maximally similar among those that satisfy the user's current critique. Such a case must exist as the user may select a critique in progressive critiquing only if it is satisfied by at least one case.

As previous recommendations are not eliminated in progressive critiquing, the case recommended in this situation may be one that was previously critiqued by the user. As we show in Theorem 2, however, a previous recommendation can be repeated in progressive critiquing only when there is no case that satisfies all the user's critiques.

Theorem 2. *A previous recommendation can be repeated in progressive critiquing only when a progression failure has occurred.*

Proof. Let C_1 be the initially recommended case and let r_1, ..., r_k be any sequence of critiques that has not resulted in a progression failure. For $1 \le i \le k$, let C_{i+1} be the case recommended in response to r_i. As a progression failure has not occurred,

$$C_{k+1} \in \bigcap_{1 \le i \le k} matches(r_i, C_i).$$

As $C_i \notin matches(r_i, C_i)$ for $1 \le i \le k$, it follows that $C_{k+1} \notin \{C_1, ..., C_k\}$. □

By allowing previous recommendations to be repeated in the event of a progression failure, progressive critiquing avoids the *diminishing choices* problem (Section 2.1). Moreover, as we now show, the reachability of cases that satisfy a given set of expressible constraints, if any, does not depend on the elimination of previously recommended cases in progressive critiquing.

Theorem 3. *A case that satisfies a given set Q of expressible constraints can always be reached in progressive critiquing without critiquing or progression failures if such a case exists.*

Proof. If $C_1 \notin Matches(Q)$, where C_1 is the initially recommended case, then there exists $q_1 \in Q$ such that $C_1 \notin matches(q_1)$. As all the constraints in Q are expressible using the available critiques, there exists $r_1 \in critiques(C_1)$ such that $matches(q_1) \subseteq matches(r_1, C_1)$. If $C_2 \notin Matches(Q)$, where C_2 is the case recommended in response to r_1, then there exists $q_2 \in Q$ such that $C_2 \notin matches(q_2)$ and $r_2 \in critiques(C_2)$ such that $matches(q_2) \subseteq matches(r_2, C_2)$. In progressive critiquing, the case C_3 recommended in response to r_2 must also satisfy r_1 if such a case exists. But $Matches(Q) \subseteq matches(q_1) \cap matches(q_2) \subseteq matches(r_1, C_1) \cap matches(r_2, C_2)$, so the existence of at least one case that satisfies all the constraints in Q ensures the existence of a case that satisfies both critiques. As a progression failure has not occurred, C_1, C_2, and C_3 must be distinct cases by Theorem 2.

If $C_3 \notin Matches(Q)$, we can continue as long as necessary to construct a sequence of recommended cases C_1, C_2, ..., C_k and successful critiques r_1, r_2, ..., r_k such that C_1, C_2, ..., $C_k \notin Matches(Q)$ and:

$$C_{k+1} \in \bigcap_{1 \le i \le k} matches(r_i, C_i)$$

where C_{k+1} is the case recommended in response to r_k. Again by Theorem 2, C_1, C_2, ..., C_{k+1} must be distinct cases. As the supply of distinct cases $C \notin Matches(Q)$ must eventually be exhausted, it must eventually be true that $C_{k+1} \in Matches(Q)$. □

3.2 Recovery from Progression Failures

A mechanism for recovery from progression failures is needed to ensure that progress can again be made if the user is prepared to compromise. Our solution is to maintain a list of *active* critiques that a recommended case must, if possible, satisfy as well as the user's current critique. If no such case exists, a maximally similar case among those that satisfy the current critique is recommended, and any active critiques that are not satisfied by this case are deleted from the list of active critiques.

Figure 3 shows an example critiquing dialogue in the personal computer (PC) domain that we use to illustrate our approach. Values of the critiqued attributes PC type, screen size (in inches), and price (in UK pounds) are shown for each recommended case. A progression failure has occurred

as the case recommended in response to the user's third critique satisfies only two of her three critiques.

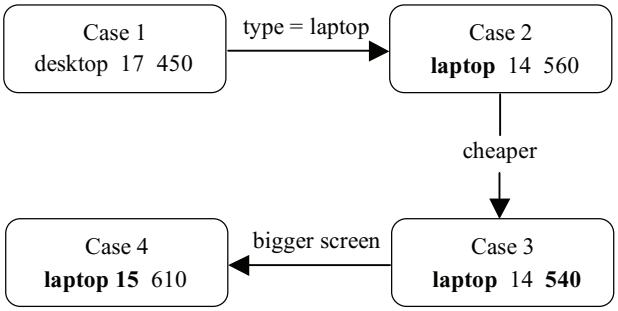

Figure 3. Example dialogue in progressive critiquing.

If the user now critiques Case 4, her critiques on Case 1 and Case 3, but not on Case 2, remain in force. For example, a PC recommended in response to a *faster* critique on Case 4 must also be a laptop with *screen* > 14 if such a case exists. If not prepared to compromise on price, the user may instead choose to navigate back in the direction of less expensive options by critiquing Case 4 on price. Such a critique may bring her directly back to Case 2 or Case 3, but only if there is no laptop cheaper than Case 4 with *screen* > 14.

3.3 Related Work

Taking account of previous critiques is a feature that progressive critiquing shares with incremental critiquing [Reilly *et al*., 2005]. In the latter approach, a case's similarity to the critiqued case is combined with the number of previous critiques it satisfies in a single *compatibility* measure. However, a highly similar case may dominate other cases that satisfy more critiques, with the result that the case recommended in response to a critique may not satisfy all the user's critiques even if such a case exists. Also in contrast to progressive critiquing, previously recommended cases are eliminated in incremental critiquing. This feature also distinguishes incremental critiquing from the approaches used in FindMe systems like *Entrée* and *Car Navigator* [Burke *et al*., 1997; Hammond *et al*., 1996].

Assessment of similarity with respect to *less-is-better* (LIB) attributes and *more-is-better* (MIB) attributes in progressive critiquing is based on assumed preferences with respect to such attributes [McSherry and Aha, 2006]. That is, the preferred value of a LIB attribute (e.g., *price*) is assumed to be the *lowest* value in the product case base, while the preferred value of a MIB attribute is assumed to be the *highest* available value. One advantage is that no updating of user preferences is needed in response to critiques on MIB or LIB attributes. Instead, only the user's *constraints* are updated, thus avoiding problems such as how to adjust a preferred price in response to a *Like this but cheaper* critique [Bridge and Ferguson, 2002].

4 Empirical Results

Our evaluation focuses on the effects of user critiquing behavior on dialogue outcomes and efficiency when previously recommended cases are not eliminated in progressive critiquing. We expect the best results to be achieved when the user refrains from *over-critiquing* attributes whose current values are acceptable. For example, a *Like this but cheaper* critique on a product whose price is acceptable may cause an unnecessary progression failure that hinders progress towards the user's goal. Over-critiquing may also result in a *critiquing loop* in which the user keeps coming back to an unacceptable case she has already critiqued. This is an important issue, as a critiquing loop may result in recommendation failure if the user decides to terminate the dialogue.

Performance measures of interest in our evaluation are the average length of critiquing dialogues and the percentage of dialogues in which a *target* case is reached. Our experiments are based on a leave-one-in approach in which each case in the PC case base [McGinty and Smyth, 2002] is used to represent the user's requirements in a simulated critiquing dialogue. The value of a LIB attribute in a left-in case is treated as an upper limit (e.g., *price* ≤ 700), and the value of a MIB attribute as a lower limit (e.g., *speed* ≥ 2.2). The values of other attributes (e.g., *make*, *type*, *screen size*) are treated as equality constraints. The left-in case also serves as a target case in the critiquing dialogue. If other cases also satisfy all the user's constraints, as in 14% of dialogues, there may be more than one target case.

The case that is *least* similar to the left-in case is presented as the initially recommended case in a simulated critiquing dialogue that continues until a target case has been reached or the user has tried all possible critiques on a recommended case without reaching a target case. The available critiques are *less* critiques on LIB attributes, *more* critiques on MIB attributes, *more* and *less* critiques on *screen size* and replacement critiques on nominal attributes.

We experiment with simulated users in two categories:

Class 1: Choose only critiques on attributes that fail to satisfy their constraints

Class 2: Choose only critiques on attributes that fail to satisfy their constraints or LIB/MIB attributes that already satisfy their constraints

Table 3 shows the percentages of critiquing dialogues that were successful (i.e., a target case was reached) in Classes 1 and 2. The fact that all Class 1 dialogues were successful provides empirical confirmation that a case that satisfies a given set of expressible constraints can always be reached in progressive critiquing if one exists (Theorem 3). In Class 2, however, over-critiquing resulted in 13% of dialogues being terminated by the simulated user without reaching a target case.

For all Class 1 dialogues and successful (+ve) Class 2 dialogues, Figure 4 shows the minimum, average, and maximum numbers of critiques required to reach a target case. Lengths of unsuccessful (-ve) dialogues in Class 2 are

also shown. A striking feature of the results for Class 1 is that only 4 critiques are required on average to reach a target case that satisfies all the user's constraints.

	Successful	Unsuccessful
Class 1	100%	0%
Class 2	87%	13%

Table 3. Outcomes of progressive critiquing dialogues.

In successful Class 2 dialogues, over-critiquing had a major impact in terms of the average number of critiques (9.5) required to reach a target case. A greater cause for concern, however, is the average length of more than 20 critiques for unsuccessful dialogues in Class 2. These findings support our hypothesis that progressive critiquing is most effective when users refrain from over-critiquing attributes (e.g., *price*) whose current values are acceptable.

Figure 4. Lengths of progressive critiquing dialogues.

5 Conclusions

The *diminishing choices* problem is a potentially serious drawback of critiquing approaches in which previously recommended products are eliminated. On the other hand, this policy avoids another serious problem which we refer to as the *unreachability* problem. Our solution to this apparent dilemma is a new version of progressive critiquing [McSherry and Aha, 2006] that leaves open the option of repeating a previous recommendation while also ensuring that a product which satisfies a given set of expressible constraints can always be reached if any such products exist.

As shown by our empirical results, the approach is most effective when the user refrains from *over-critiquing*, with only four critiques required on average to reach a target case in the PC case base without the benefit of an initial query. The problems caused by over-critiquing could perhaps be avoided by offering users guidelines for more effective critiquing. Other possible solutions to be investigated in future work include eliminating any product that the user critiques for a second time.

References

[Bridge and Ferguson, 2002] Derek Bridge and Alex Ferguson. An expressive query language for product recommender systems. *Artificial Intelligence Review*, 18:269-307, 2002.

[Burke, 2002] Robin Burke. Interactive critiquing for catalog navigation in e-commerce. *Artificial Intelligence Review*, 18:245-267, 2002.

[Burke *et al*., 1997] Robin Burke, Kristian Hammond, and Benjamin Young. The FindMe approach to assisted browsing. *IEEE Expert*, 12:32-40, 1997.

[Hammond *et al*., 1996] Kristian Hammond, Robin Burke, and Kathryn Schmitt. A case-based approach to knowledge navigation. In David Leake, editor, *Case-Based Reasoning: Experiences, Lessons & Future Directions*, pages 125-136. AAAI Press, Cambridge, Massachusetts, 1996.

[Linden *et al*., 1997] Greg Linden, Steve Hanks, and Neal Lesh. Interactive assessment of user preference models: The Automated Travel Assistant. In *Proceedings of the Sixth International Conference on User Modeling*, pages 67-78, Chia Laguna, Sardinia, Italy, 1997. Springer Wien New York.

[McCarthy *et al*., 2005] Kevin McCarthy, James Reilly, Lorraine McGinty, and Barry Smyth. Experiments in dynamic critiquing. In *Proceedings of the Tenth International Conference on Intelligent User Interfaces*, pages 175-182, San Diego, California, 2005. ACM Press.

[McGinty and Smyth, 2002] Lorraine McGinty and Barry Smyth. Comparison-based recommendation. In *Proceedings of the Sixth European Conference on Case-Based Reasoning*, pages 575-589, Aberdeen, Scotland, 2002. Springer-Verlag.

[McSherry, 2003] David McSherry. Similarity and compromise. In *Proceedings of the Fifth International Conference on Case-Based Reasoning*, pages 291-305, Trondheim, Norway, 2003. Springer-Verlag.

[McSherry and Aha, 2006] David McSherry and David W. Aha. Avoiding long and fruitless dialogues in critiquing. In *Proceedings of AI-2006*, Cambridge, UK, 2006. Springer-Verlag.

[Reilly *et al*., 2005] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Incremental critiquing. *Knowledge-Based Systems*, 18:143-151, 2005.

[Salamó *et al*., 2005] Maria Salamó, Barry Smyth, Kevin McCarthy, James Reilly, and Lorraine McGinty. Reducing critique repetition in conversational recommendation. *IJCAI-05 Workshop on Multi-Agent Information Retrieval and Recommender Systems*, pages 55-61, 2005.