Dynamic Critiquing

James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth

Adaptive Information Cluster*, Smart Media Institute,
Department of Computer Science, University College Dublin (UCD), Ireland.
{james.d.reilly, kevin.mccarthy, lorraine.mcginty, barry.smyth}@ucd.ie

Abstract. Critiquing is a powerful style of feedback for case-based recommender systems. Instead of providing detailed feature values, users indicate a directional preference for a feature. For example, a user might ask for a 'less expensive' restaurant in a restaurant recommender; 'less expensive' is a critique over the *price* feature. The value of critiquing is that it is generally applicable over a wide range of domains and it is an effective means of focusing search. To date critiquing approaches have usually been limited to single-feature critiques, and this ultimately limits the degree to which a given critique can eliminate unsuitable cases. In this paper we propose extending the critiquing concept to cater for the possibility of *compound critiques*—critiques over multiple case features. We describe a technique for automatically generating useful compound critiques and demonstrate how this can significantly improve the performance of a conversational recommender system. We also argue that this generalised form of critiquing offers explanatory benefits by helping the user to better understand the structure of the recommendation space.

1 Introduction

Conversational recommender systems are a response to the fact that in many information seeking scenarios it is unlikely that the user will provide enough requirements information to uniquely identify what it is that she is looking for. As such, conversational recommender systems assume that a user's initial query is merely a starting point for search, perhaps even an unreliable starting point, and the job of the recommender system is to help the user to refine her initial query as part of an extended system-user interaction (see for example, [1, 9, 16, 18]). Thus, a typical session with a conversational recommender system usually takes the form of a series of *recommend-review-revise* cycles: the user is presented with one or more item recommendations; she provides some form of feedback regarding the suitability of these items; and the recommender updates its user-model according to this feedback and proceeds to the next recommendation cycle. The hope of course is that after a small number of cycles the recommender will have sufficient information to recommend a suitable item to the user.

* This material is based on works supported by Science Foundation Ireland under Grant No. 03/IN.3/I361

Research in the area of recommender systems to date has focused on various aspects of the conversational recommender system architecture, with a number of researchers exploring different options when it comes to the recommend, review and revise stages of each cycle. Briefly, researchers have explored a variety of ways to select items for recommendation, recently moving beyond straightforward similarity-based approaches towards more sophisticated techniques that incorporate factors such as item diversity into the retrieval process [2, 8, 12, 16, 18]. In turn, there are many ways in which a user can review a set of recommendations and a variety of different forms of feedback have been proposed [17]. Finally, other recent work has looked at how user feedback can be exploited to update the recommender's model of the user's requirements [9].

In this paper we focus on the use of a popular form of feedback in conversational recommender systems: *critiquing*. The basic idea was first introduced by Burke *et al.* [3–5] and proposes a form of feedback that expresses what might be termed a *directional preference* over a particular item feature. Entree is the quintessential recommender system that employs critiquing (also sometimes referred to as *tweaking*). Entree is a restaurant recommender, and each time a restaurant is suggested it allows the user to provide feedback in the form of a *critique* or *tweak*. Each tweak is a constraint over the value-space of a particular restaurant feature. For example, the user might indicate that they are looking for a *less expensive* restaurant or a *more formal* setting. These are two individual tweaks: the former on the *price* feature and the latter on the *setting* feature. The advantage of critiquing is that it is a fairly lightweight form of feedback, in the sense that the user does not need to provide specific value information for a feature, while at the same time helping the recommender to narrow its search focus quite significantly [10, 11].

This standard form of critiquing *normally* operates at the level of individual features and this limits its ability to narrow the search focus. At the same time it seems that users often think more naturally about combinations of features. For example, in the PC domain a user might be looking for a PC that is similar to the one shown but that has more memory at a lower price; this is an example of a *compound critique* over the *memory* and *price* features of a PC case. Intuitively compound critiques appear to have the ability to improve the efficiency of conversational recommender systems, by focusing search in the direction of multiple, simultaneous feature constraints; of course they are not limited to two features and in theory compound critiques could be created to operate over all of the features of a case, although this might impact on a user's ability to understand the critique.

This is not to say that the idea of compound critiques is novel. In fact, the seminal work of [4] refers to critiques for manipulating multiple features. They give the example of the 'sportier' critique, in a car recommender, which increases engine size and acceleration and allows for a greater price. This is a compound critique but it has been fixed by the system designer. We believe that a more flexible approach is required because the appropriateness of a particular compound critique will very much depend on the remaining cases that are available.

For example, if, during the course of the recommender session, a car-buyer has focused on large family cars then it is unlikely that many 'sporty' cars will exist in the remaining cases and the 'sportier' critique may no longer be valid. However, a new compound critique—lower engine size, greater fuel economy and a higher eco-rating—might now be useful. The point is that compound critiques should be generated on the fly with reference to the remaining cases and this is the starting point for our work. In this paper we will describe and evaluate different approaches to automatically creating and prioritising compound critiques. Specifically, in Section 2.2 we will describe how the Apriori algorithm[7, 15] can be used to generate and grade candidate compound critiques, and in Section 3 we will demonstrate how the availability of these selected critiques can reduce the average number of recommendation cycles needed by up to 40%.

2 Dynamic Compound Critiquing

In this work we will assume a conversational recommender system in the image of Entree. Each recommendation session will be commenced by an initial user query and this will result in the retrieval of the most similar case available for the first recommendation cycle. The user will have the opportunity to accept this case, thereby ending the recommendation session, or to critique this case. When she critiques the case, the critique in question acts as a filter over the remaining cases, and the case chosen for the next cycle is that case which is compatible with the critique and which is maximally similar to the previously recommended case.

To critique a case the user will be presented with a range of single-feature (*unit*) critiques plus a set of compound critiques that have been chosen because of their ability to carve-up the remaining cases. In this section we will describe in detail how these compound critiques are generated and selected during each cycle by looking for frequently occurring patterns of critiques within the remaining cases; we refer to our approach as *dynamic critiquing*.

2.1 Critique Patterns - A Precursor to Discovery

Let us assume that our recommender system is currently engaged in a recommendation session with a user, and that a new case has been returned as part of the current cycle. Each case that remains in the case-base can be compared to this new case to generate a so-called *critique pattern*. This pattern essentially recasts each case in the case-base in terms of the unit critiques that apply to each of its features when compared to the current case.

Figure 1 illustrates what we mean with the aid of an example. It shows the current case that has been selected for recommendation to the user as part of the current cycle and also a case from the case-base. The current case describes a 1.4GHz, desktop PC with 512Mb of RAM, a 14" monitor and a 40Gb hard-drive, all for 1500 euro. The comparison case, from the case-base, describes a 900MHz, desktop with 512MB or RAM, a 12" monitor and a 30Gb hard-drive for 3000

	Current Case	Case c from CB	Critique Pattern
Manufacturer	Compaq	Sony	!=
Monitor (inches)	14″	12″	<
Memory (MB)	512	512	=
Hard-Disk (GB)	40	30	<
Processor	Pentium 3	Pentium 3	=
Speed (Mhz)	1400	900	<
Type	Desktop	Desktop	=
Price (Euro)	1500	3000	>

Fig. 1. Illustrating how a critique pattern is generated.

euro. The resulting critique pattern reflects the differences between these two cases in terms of individual feature critiques. For example, the critique pattern shown includes a "<" critique for processor speed— we will refer to this as $[Speed <]$—because the comparison case has a slower processor than the current recommended case. Similarly, the pattern includes the critique $[Price >]$ because the comparison case is more expensive than the current case. So, prior to the discovery process, and after a case has been selected for the current cycle, it is necessary to generate a critique pattern for every case in the case-base relative to the current case. These patterns serve as the source of compound critiques.

2.2 Discovering Compound Critiques

The key to exploiting compound critiques relies on our ability to recognise useful recurring subsets of critiques within the potentially large collection of critique patterns (the *pattern-base*). Our intuition is that certain subsets will tend to recur throughout the pattern-base. For example, we might find that 50% of the remaining cases have a smaller screen-size but a larger hard-disk size than the current case; that is, 50% of the critique patterns contain the sub-pattern $\{[Monitor <],[Hard-Disk >]\}$. If this critique is applicable to the user—if she is in fact looking for smaller screens and larger hard-disks—then its application will immediately filter out half of the remaining cases, thus better focusing the search for a suitable case during the next cycle. Presumably, neither of the individual critiques that make up this compound critique would wield the same discriminatory power on their own.

The problem at hand then is how to recognise and collate these recurring critique patterns within the pattern-base. This is similar to the so-called *market basket analysis*, which aims to find regularities in the shopping behaviour of customers [7]: each critique pattern corresponds to the shopping basket for a single customer, and the individual critiques correspond to the items in this basket. Many data-mining algorithms try to find sets of items that are frequently purchased together, and so our proposal is to use similar techniques to find sets of critiques that frequently occur together. Ordinarily this is a challenging problem, largely because of the combinatorics involved: a typical supermarket will have

several thousand different products and this can lead to a combinatoric explosion in the number of possible groups of recurring items. This problem is not so acute in our critiquing scenario because there are only a limited number of possible critiques. For instance, each numeric feature can have a "<"or a ">" critique and each nominal feature can have a "=" or a "! =" critique, so there are only $2n$ possible critiques in a case-base where the cases are made up of n individual features.

In addition, efficient algorithms do exist for restricting the search-space of possibilities so that only a subset of all of the possible compound critiques needs to be checked. One such algorithm is the well-known Apriori algorithm [7, 15], which characterises these recurring item subsets as association rules of the form $A \rightarrow B$ —from the presence of a certain set of critiques (A) one can infer the presence of certain other critiques (B). For example, one might learn that from the presence of the critique, $[Monitor <]$, we can infer the presence of $[Hard - Disk >]$ with a high degree of probability; in other words the pattern $\{[Monitor <],[Hard - Disk >]\}$ is commonplace.

Apriori measures the importance of a rule in terms of its *support* and *confidence*. The support of a rule, $A \rightarrow B$, is the percentage of patterns for which the rule is correct; that is, the number of patterns that contain both A and B divided by the total number of patterns. Confidence, on the other hand, is a measure of the number of patterns in which the rule is correct relative to the number of patterns in which the rule is applicable; that is, the number of patterns that contain both A and B divided by the number of patterns containing A. For instance, we would find that the rule $[Monitor <] \rightarrow [Hard - Disk >]$ has a support of 0.1 if there are a total of 100 critique patterns but only 10 of them contain $[Monitor <]$ and $[Hard - Disk >]$. Likewise, the confidence of this rule would be 0.4 if 25 of the critique patterns contain only $[Monitor <]$. Unfortunately a detailed account of Apriori is beyond the scope of this paper but, very briefly, Apriori is a multi-pass algorithm, where in the k^{th} pass all large itemsets of cardinality k are computed. Initially *frequent itemsets* are determined. These are sets of items that have at least a predefined minimum support. Then, during each new pass those itemsets that exceed the minimum support threshold are extended. Apriori is efficient because it exploits the simple observation that no superset of an infrequent itemset can be frequent to prune away candidate itemsets.

Our specific proposal is to use Apriori, during each recommendation cycle, to generate a collection of compound critiques (frequent itemsets over the pattern-base), and to then select a subset of these compound critiques so that they may be presented to the user as alternative critiquing options.

2.3 Grading Compound Critiques

During any particular cycle a large number of compound critiques, of varying sizes, may be discovered. Of course it is not feasible to present all of these to the user so we must look to choose a select subset. Which subset we choose is likely to have a significant bearing on the degree to which the compound critiques may

prove to be successful at reducing session length. There are two main criteria in this regard:

- We would like to present compound critiques that are likely to be applicable to the user, in the sense that they are likely to constrain the remaining cases in the direction of their target case. In this way there is a good chance that these compound critiques will be selected over any of the unit critiques.
- We would like to present compound critiques that will filter out large numbers of cases so that there is a greater chance that the target case will be retrieved in the next cycle.

The first of these criteria is difficult to cater for since it is unlikely to be at all clear exactly what target case the user is seeking. That said, it is a good bet that certain features of their target case may be inferred from the feedback provided during previous cycles. For example, if the user reliably looks for cheaper PCs then compound critiques that contain $[Price <]$ may be good candidates. The second criterion is more straightforward to address. The support of a compound critique is a direct measure of its ability to filter out few or many cases. A compound critique with a low support value means that it is present in a small proportion of critique patterns and thus it is only applicable to a few remaining cases. If applied the critique will therefore eliminate many cases from consideration.

It is worth noting that there is a tension between the use of support as a grading metric for compound critiques and the way that it will influence the above criteria. While low-support critiques will eliminate many cases, these critiques seem less likely to lead to the target case, all things being equal. Conversely, preferring high-support critiques will increase the chance that the critiques will lead to the target case, but these critiques will fail to eliminate many cases from consideration.

3 Evaluation

At this point we have described how our dynamic critiquing technique applies Apriori within a standard critiquing-based, conversational recommender system in order to identify sets of compound critiques that may be presented to the user during each cycle. Ultimately we are doing this in the hope that compound critiques will be selected in favor of unit critiques and that they will help to reduce the number of cycles needed to satisfy the user. In this section we will evaluate this in the context of a PC recommender and we will look at a number of key issues including: the number of compound critiques that are generated during a typical cycle; the size of these critiques and the size of the critiques that are selected for presentation to the user; the application frequency of compound critiques during a typical cycle; and finally, the impact of dynamic critiquing on session length.

3.1 Setup

Algorithmic Variations. The basic dynamic critiquing approach is fixed: compound critiques are generated during each cycle (with a minimum support threshold of 0.25). However, we will examine three variations on this theme, each distinguished according to how the set of top 5 critiques is chosen for presentation to the user during the current cycle: (1) LS - the top 5 critiques with the lowest support are chosen; (2) HS - the top 5 critiques with the highest support are chosen; (3) RAND - A random set of 5 critiques are chosen. So the above provide three system variations based on dynamic critiquing and in addition we also include a standard, unit critiquing approach (STD) as a benchmark.

Dataset. The well-known PC dataset is used as a source of case and query data [9, 13]. This dataset consists of 120 PC cases each described in terms of 8 features including *type, manufacturer, processor, memory,* etc. This dataset is available for download at *www.cs.ucd.ie/staff/lmcginty/PCdataset.zip*

Methodology. We adopt a similar leave-one-out methodology to [9–11]. Specifically, each case (*base*) in the case-base is temporarily removed and used in two ways. First it serves as the basis for a set of queries constructed by taking random subsets of its features. We focus on subsets of 1, 3 and 5 features to allow us to distinguish between hard, moderate and easy queries, respectively. Second, we select the case that is most similar to the original base. These cases serve as the recommendation *targets* for the experiments. Thus, the base represents the ideal query for a 'user', the generated query is the initial query that the 'user' provides to the recommender, and the target is the best available case for the 'user', based on their ideal. Each generated query is a test problem for the recommender, and in each recommendation cycle the 'user' picks a critique that is compatible with the known target case; that is, a critique that, when applied to the remaining cases, results in the target case being left in the filtered set of cases. In a typical cycle there may be a number of critiques (unit and compound) that satisfy this condition and the actual one chosen depends on the system being used: LS picks the critique with the lowest support with ties broken by a random choice, HS picks the one with the highest support with ties broken by a random choice, and RAND picks a random critique. Each leave-one-out pass through the case-base is repeated 30 times and recommendation sessions terminate when the target case is returned.

3.2 How Many Compound Critiques?

Perhaps the first question to explore concerns the number of compound critiques that can be generated during a recommendation cycle. Obviously this will depend on a number of factors including: the current recommended case; the number of cases that are available in the cycle; and the variation that exists among these cases.

To form a general picture of how critique generation was influenced by the number of cases we recorded the number of compound critiques generated during each cycle along with the number of cases that were available during that cycle. The results are presented in Figure 2 as a graph of the average number of critiques versus the different numbers of cases. They indicate a general trend towards fewer compound critiques as the number of available cases decreases. Initially, for large numbers of cases (between 120 and 110), the number of generated critiques falls off quickly from a high of over 200, and then is seen to stabalise around the 100 level. Obviously the actual number of compound critiques generated per cycle is significant and typically there are more critiques than cases due to the combinatorics of critique generation. Remember we select only 5 of these critiques per cycle for presentation to the user so it will be interesting to investigate how the different policies (LS, HS and RAND) vary in terms of their ability to influence recommendation efficiency.

Fig. 2. Graph of the average number of critiques generated versus the different numbers of cases available.

3.3 Critique Size

Early on in this work we were concerned that the size of compound critiques would be an important issue: that compound critiques containing many features would be routinely generated, and that these critiques would be too complex to present to any user. There are 8 features per PC case and thus a compound critique can have a size of between 2 and 8, but presenting compound critiques with, for example, 5 or 6 separate features would make for a user-interface nightmare.

To understand the size-range of the critiques being generated we first counted the number of critiques generated for each of the 8 different sizes across an entire leave-one-out pass through the case-base. The results are presented in Figure 3 and they show that the majority of compound critiques are actually quite short.

For instance, 66% of critiques contain either 2 or 3 features and less than 2.5% contain more than 5 features. This indicates that there is a significant degree of variation between the PC cases; if these cases were more homogeneous then larger critiques would have been commonplace.

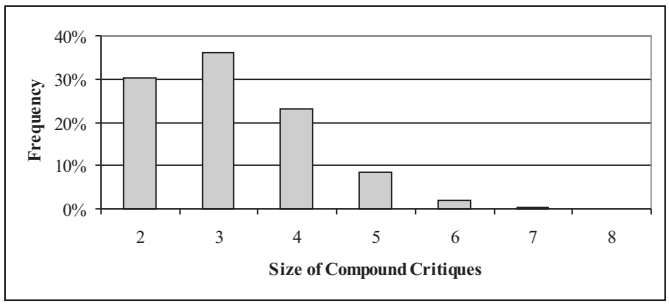

Fig. 3. Results showing the frequency of sizes of compound critiques.

As a follow-up experiment on the issue of critique size, we also looked at the average size of the compound critiques that are being presented to the user as part of the recommendation cycle, for each of the different critiquing strategies. The results are shown in Figure 4 as a graph of the average compound critique size presented during a cycle versus the maximum generation threshold, t; terminating Apriori at the $t + 1^{th}$ iteration will limit all itemsets (compound critiques) to be no greater than t in size. The results show that the compound critiques chosen by the HS strategy are unaffected by the maximum generation threshold; the top 5 HS critiques have an average size of about 2.12 regardless of the generation threshold. Of course this is to be expected since critiques with a high support value are likely to be small; high support means that the critique is common across the available cases and this is more likely if the compound critique contains few features. Thus, preferring compound critiques with high support is tantamount to preferring small compound critiques. Conversely, the average size of the compound critiques selected by the LS strategy are larger by up to 76% (an average size of 3.7) when the maximum generation threshold is set to its maximum value of 8. When we look at the average size of compound critiques that are actually selected by the 'user' in our experiments we find that they too follow the pattern presented in Figure 4 and are virtually identical to the sizes shown.

In summary then, our initial concerns about the generation of large compound critiques, and the problems that this might cause, appear to be unfounded. First of all, it is always possible to limit the size of the critiques, at generation-time, by prematurely halting Apriori after an appropriate number of iterations. Secondly, even without imposing this limit, large critiques are unlikely to occur

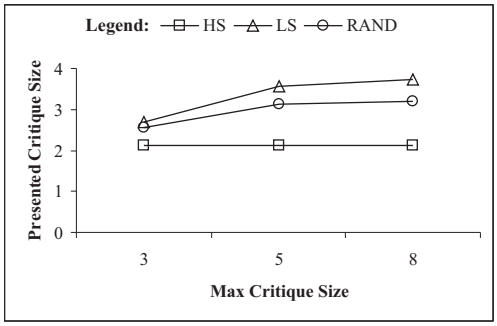

Fig. 4. Graph of the average compound critique size presented during a cycle versus the maximum generation threshold.

unless the available cases are unusually homogeneous. And even the LS critique selection strategy, which naturally favours large critiques still only selects critiques with at most 3 or 4 features.

3.4 Application Frequency

So far we have seen that, in general, our dynamic critiquing technique tends to produce large numbers of compound critiques in a typical cycle. We have also found most of these critiques to contain between 2 and 4 features and the HS selection strategy tends to prefer small critiques while the LS strategy prefers the larger critiques. Given that in our experiment we are returning 5 compound critiques per cycle to the user the next logical question is whether these critiques tend to be chosen.

Figure 5 presents a graph of the probability that a compound critique will be chosen by the 'user' at each cycle in a recommendation session. We compute the probability that a compound critique will be chosen in cycle k by calculating the proportion of times that a compound critique was selected in a k^{th} cycle throughout our experiment. Figure 5 plots these probability distributions for each dynamic critiquing strategy for up to the 20^{th} recommendation cycle. The results are as predicted in Section 2.3 where we suggested that compound critiques would be more likely to be chosen under the HS strategy than under the LS strategy. Figure 5 indicates that compound critiques are chosen under the HS strategy between 55% and 86% of the time. Under the LS strategy they are chosen between only 33% and 40% and under RAND they are chosen about 15% of the time.

3.5 Recommendation Efficiency

Our basic assumption is that the application of a compound critique should help to focus search more efficiently than the application of a unit critique and

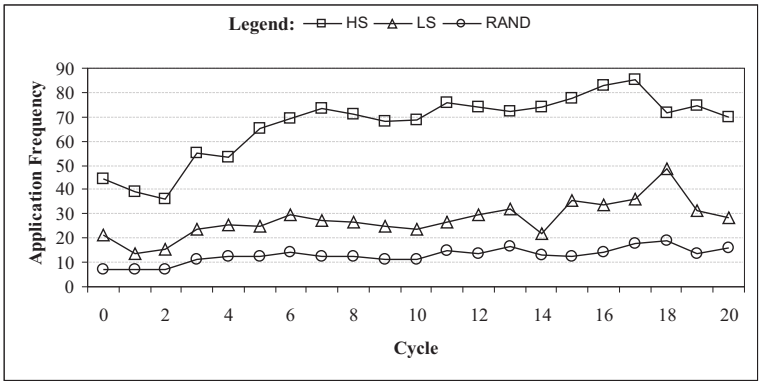

Fig. 5. Results showing the probability that a compound critique will be selected for each of the strategies evaluated.

therefore reduce the length of recommendation sessions. We now know that the HS strategy leads to the frequent application of small compound critiques whereas the LS strategy leads to the less frequent application of large critiques. Which, if either, of these strategies leads to a reduction in session length remains to be seen. To test this we compare the performance of each of our four strategies on the PC case-base according to the leave-one-out test methodology described above.

For each recommendation session we record the number of cycles required before the target case is retrieved, and we average these values for each system. The summary results are presented in Figure 6 as a bar-chart with each bar showing the average session length for each of the four test systems measured across all test queries. Clearly, there is an advantage to the dynamic critiquing approach, with all three variations out-performing the standard critiquing system on average. However, the scale of this advantage is very much dependent on the variation of dynamic critiquing that is used. For example, the HS variation provides only a very minor advantage, reducing average session length from 5.89 cycles to 5.68 cycles, a relative reduction of less than 4%. This is to be expected as the HS system prefers compound critiques with high support values and these critiques, by definition, will not serve to filter cases greatly upon their application. The advantage is far more striking when we look at the LS system, which uses a complementary strategy that prefers critiques with low support values, critiques that are likely to constrain the cases for the next cycle. The LS variation reduces session length to 3.8, a reduction of nearly 36% relative to STD. The RAND variation occupies the middle-ground offering relative reductions of almost 16%.

It is also worth investigating how recommendation efficiency is influenced by initial query length, as a measure of query difficulty. To do this we recomputed the session length averages above by separating out the queries of various sizes.

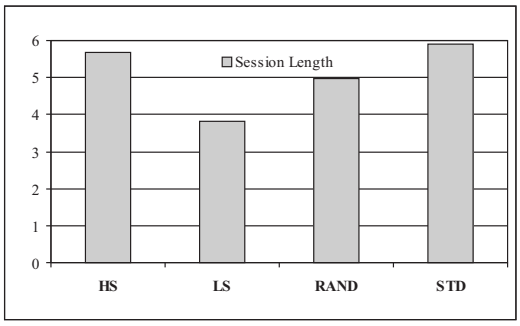

Fig. 6. Evaluation results illustrating the average session lengths recorded for each of the systems compared.

The results are presented in Figure 7. Figure 7(a) shows a graph of average session length against initial query length. Once again the results are very clear, pointing to a significant advantage for LS, but little or no advantage for HS. As expected the average session length falls with increasing initial query length but it is interesting to note that the scale of any advantage due to dynamic critiquing appears to be greater for the more difficult (smaller) initial queries. Figure 7(b) shows the results for the relative improvements realised over STD for all of the strategies evaluated. For example, for difficult queries (with a single specified feature) we find that LS enjoys a session length reduction of just over 40%, relative to STD. This falls to almost 32% for the moderate queries (containing 3 specified features) and falls again to just under 26% for the relatively easy queries with 5 fully specified features. This is to be expected perhaps since the easier queries naturally result in shorter sessions and thus there are fewer opportunities for compound critiques to be chosen, and hence fewer opportunities for their benefit to be felt.

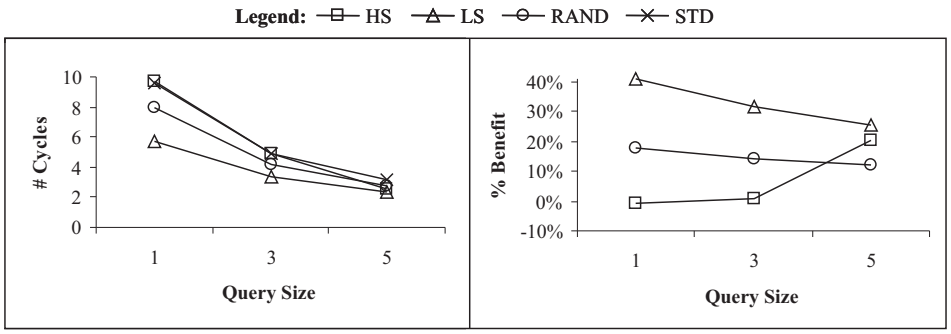

Fig. 7. Recommendation Efficiency Results.

4 Discussion

In summary then, dynamic critiquing (LS strategy) appears to offer significant performance benefits when compared to standard critiquing. We believe that it can be readily incorporated into standard case-based conversational recommender systems. The Apriori algorithm is an efficient approach to compound critique generation and its inclusion does not add appreciably to the overall computational load of the recommender system. For example, in our evaluation system, the compound critiques for a typical cycle are generated in an average of 46ms (on a 2.8 GHz Pentium 4) and, as such, offer no significant computational burden to the recommender. This does not mean that Apriori will necessarily scale well for very large case-bases, but we believe that it will remain computationally tractable for typical case-bases sizes. After all, Apriori is designed for large-scale data-based mining running into the millions of transactions and thousands of unique items and case-bases with hundreds or even thousands of cases and only tens of features do not present a significant challenge.

In addition to the performance advantages of dynamic critiquing it is also worth considering what we might term its 'explanatory benefits'. Recently a number of researchers have argued for the need for improved interaction between system and user, with many arguing that recommender systems must provide some form of explanation to the user in order to help her understand the reason behind recommendations [14, 6]. We believe that compound critiques have a role to play in this regard. Unlike unit critiques, compound critiques help users to understand some of the common interactions that exist between groups of features. For example, in the PC domain, the compound critique $[Speed >], [Memory >], [Price >]$ tells the user that faster processors, more memory, and higher prices go hand-in-hand, and by tagging this critique with its support value we can inform the user about the proportion of remaining cases that satisfy this critique. We believe that in many recommender domains, where the user is likely to have incomplete knowledge about the finer details of the feature-space, that compound critiques will help to effectively map out this space. For this reason we believe that users will actually find it easier to work with compound critiques than unit critiques and this may, for example, help the user to make fewer critiquing errors. For instance, with standard critiquing in the PC domain a user might naively select the $[Price <]$ unit critique in the mistaken belief that this may deliver a cheaper PC that satisfies all of their other requirements. However, reducing price in this way may lead to a reduction in memory that the user might not find acceptable and, as a result, she will have to backtrack. This problem is less likely to occur if the compound critique $[Price <], [Memory <]$ is presented because the user will come to understand the implications of a price-drop prior to selecting any critique. Of course all of these findings need to be validated in real-user trials.

A further research issue relates to the cognitive load associated with asking the user to evaluate more that one feature at a time. While we are confident that savings in processing time and session length will outweigh the cognitive

burden placed on the user, supportive evaluations have yet to be carried out. Once again, a real-user study is necessary here.

5 Conclusions

Critiquing is an important mode of user feedback that is ideally suited to many case-based recommendation scenarios. It is straightforward to implement, easy for users to understand and use, and it has been shown to be effective at guiding conversational recommender systems. To date the standard form of critiquing has been largely limited to single-feature critiques, what we have called unit critiques. In this paper we have suggested the use of compound critiques to constrain multiple features simultaneously. We have described a technique called dynamic critiquing, which is capable of automatically and efficiently generating compound critiques during each recommendation cycle. And as part of each cycle a subset of these new critiques is presented to the user, along with the standard unit critiques. We have evaluated our technique using the well-known PC data-set and compared different strategies for selecting a suitable subset of critiques. Our experiments indicate that significant performance improvements are possible: the LS dynamic critiquing strategy, which prefers low-support critiques, has the ability to reduce session length by up to 40% compared to standard critiquing.

In summary then, the idea of using compound critiques is a general one that is likely to be just as applicable as standard critiquing across a wide range of recommendation scenarios. The use of compound critiques clearly has the potential to improve recommendation efficiency and dynamic critiquing provides an efficient approach to generating suitable compound critiques in conversational recommender systems. We further suggest that these compound critiques may confer an explanatory advantage on a recommender system by helping the user to appreciate the dependancies that exist between features and cases. Together these advantages establish compound critiquing as a powerful new interaction modality for case-based conversational recommender systems.

References

1. D.W. Aha and K.M. Gupta. Causal Query Elaboration in Conversational Case-Based Reasoning. In S. Haller and G. Simmons, editors, *Proceedings of the Fifteenth International FLAIRS Conference*, pages 95–100. AAAI Press, 2002. Pensacola Beach, Florida, USA.
2. D. Bridge. Product Recommendation Systems: A New Direction. In D. Aha and I. Watson, editors, *Workshop on CBR in Electronic Commerce at The International Conference on Case-Based Reasoning (ICCBR-01)*, 2001. Vancouver, Canada.
3. R. Burke. Interactive Critiquing for Catalog Navigation in E-Commerce. *Artificial Intelligence Review*, 18(3-4):245–267, 2002.
4. R. Burke, K. Hammond, and B. Young. Knowledge-based navigation of complex information spaces. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 462–468. AAAI Press/MIT Press, 1996. Portland, OR.

5. R. Burke, K. Hammond, and B.C. Young. The FindMe Approach to Assisted Browsing. *Journal of IEEE Expert*, 12(4):32–40, 1997.
6. P. Cunningham, D. Doyle, and J. Loughrey. An Evaluation of the Usefulness of Case-Based Explanation. . In K. Ashley and D. Bridge, editors, *Case-Based Reasoning Research and Development. LNAI, Vol. 2689.*, pages 191–199. Springer-Verlag, 2003. Berlin.
7. Z. Hu, W.N. Chin, and M. Takeichi. Calculating a New Data Mining Algorithm for Market Basket Analysis. *Lecture Notes in Computer Science*, 1753:169–175, 2000.
8. A. Kohlmaier, S. Schmitt, and R. Bergmann. Evaluation of a Similarity-based Approach to Customer-adaptive Electronic Sales Dialogs. In S. Weibelzahl, D. Chin, and G. Weber, editors, *Empirical Evaluation of Adaptive Systems. Proceedings of the workshop held at the 8th International Conference on User Modelling*, pages 40–50, 2001. Sonthofen, Germany.
9. L. McGinty and B. Smyth. Comparison-Based Recommendation. In Susan Craw, editor, *Proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR-02)*, pages 575–589. Springer, 2002. Aberdeen, Scotland.
10. L. McGinty and B. Smyth. On The Role of Diversity in Conversational Recommender Systems. In D. Bridge and K. Ashley, editors, *Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR-03)*, pages 276–290. Springer, 2003. Troindheim, Norway.
11. L. McGinty and B. Smyth. Tweaking Critiquing. In *Proceedings of the Workshop on Personalization and Web Techniques at the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 20–27. Morgan-Kaufmann, 2003. Acapulco, Mexico.
12. D. McSherry. Diversity-Conscious Retrieval. In Susan Craw, editor, *Proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR-02)*, pages 219–233. Springer, 2002. Aberdeen, Scotland.
13. D. McSherry. Balancing User Satisfaction and Cognitive Load in Coverage-Optimised Retrieval. In Preece A. Macintosh A. Coenen, F., editor, *Research and Development in Intelligent Systems XX. Proceedings of AI-2003*, pages 381–394. Springer-Verlag, 2003. Cambridge, UK.
14. D. McSherry. Explanation of Retrieval Mismatches in Recommender System Dialogues. . In *Proceedings of the ICCBR-03 Workshop on Mixed-Initiative Case-Based Reasoning*, pages 191–199, 2003. Trondheim, Norway.
15. R. Srikant H. Toivonen R. Agrawal, H. Mannila and A. Inkeri Verkamo. Fast Discovery of Association Rules in Large Databases. *Advances in Knowledge Discovery and Data Mining*, pages 307–328, 1996.
16. H. Shimazu. ExpertClerk : Navigating Shoppers' Buying Process with the Combination of Asking and Proposing. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 1443–1448. Morgan Kaufmann, 2001. Seattle, Washington, USA.
17. B. Smyth and L. McGinty. An Analysis of Feedback Strategies in Conversational Recommender Systems. In P. Cunningham, editor, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Cognitive Science (AICS-2003)*, pages 211–216, 2003. Dublin, Ireland.
18. B. Smyth and L. McGinty. The Power of Suggestion. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 127–138. Morgan-Kaufmann, 2003. Acapulco, Mexico.