Integrating the Mechanisms of Critiquing-based Recommendation into Constraint Solving

Pavle Knežević[1], Alexander Felfernig[1] and Sebastian Lubos[1]

[1]Institute of Software Technology, Graz University of Technology, Inffeldgasse 16b, 8010 Graz, Austria

Abstract

Critiquing-based recommender systems enhance decision-making by guiding users through a product space to find items that meet their preferences. By incorporating feedback in the form of critiques that constrain feature value spaces, these systems refine user profiles to provide more accurate and tailored recommendations. This paper presents a novel approach that integrates critiquing into constraint solving, offering particular benefits for configurable products where finding optimal configurations is complex. We conducted a preliminary offline evaluation of unit-critiquing in the prototype system to gain initial insights into the approach's efficiency and flexibility. The results suggest that this method has the potential to efficiently generate relevant recommendations, highlighting its promise for addressing challenges in configurable product recommendations.

Keywords

Critiquing-based recommender system, Constraint solving, Decision-making

1. Introduction

In recent decades, critiquing has gained broad recognition as an effective approach in preference-based search and recommender systems. This method allows users to express preferences and provide feedback on various product aspects without specifying exact values [1, 2]. A notable advantage is its capacity to address the cold-start problem prevalent in methods such as collaborative filtering, which depend on detailed user data and past interactions [3]. By using a navigation-based approach, these systems help users explore the item space by presenting a reference product for acceptance or feedback via critiques [3, 4]. Based on this feedback, the system refines its recommendations in subsequent cycles, aiding users in making better decisions. Primarily, critiquing was developed for *Case-Based Reasoning (CBR)* recommendation approaches, which rely on a database (or *case-base*) where items are modeled as cases, and recommendations are produced by retrieving cases most similar to a user's query or profile [5, 6]. Although, this approach has proven useful to generate relevant recommendations, the need for improvements remains, particularly in the system's ability to dynamically adapt the critiquing process in response to the user's evolving preferences [3].

Constraint solving approaches enable the compact representation of complex problems [7]. These methods are widely employed in *constraint-based recommender systems*, which suggest products and services based on a given set of constraints [8]. Such systems include a recommender knowledge base, defined by various sets of variables and constraints, which are the core components of a *constraint satisfaction problem (CSP)* [5, 9]. Solving a CSP involves finding specific assignments for the variables that satisfy all given constraints. Those approaches have been successfully applied in complex domains such as automotive and financial services [8].

In this paper, we present a recommendation approach that applies critiquing while interacting with a constraint solver from the *Choco* [10] library. The core idea is that user-specified critiques can be directly translated into constraints, which are then processed by the solver. This approach leverages constraint solving technology to identify valid configurations meeting user requirements [11]. Additionally, our approach incorporates a search heuristic that considers user preferences which can positively impact

IntRS'24: Joint Workshop on Interfaces and Human Decision Making for Recommender Systems, October 18, 2024, Bari (Italy)
pavle.knezevic@student.tugraz.at (P. Knežević); alexander.felfernig@ist.tugraz.at (A. Felfernig); slubos@ist.tugraz.at (S. Lubos)
0000-0003-0108-3146 (A. Felfernig); 0000-0002-5024-3786 (S. Lubos)

Table 1
Variables of the set V_{PROD} describing the properties of a tennis racket.

Variable	Domain
$name_p$	*text*
$brand_p$	{*Tecnifibre, ProKennex, Volkl, Dunlop, Babolat, Head, Prince, Wilson, Yonex*}
$weight_p$	$\{247, \ldots, 354\}$
$balance_p$	$\{312, \ldots, 381\}$
$headSize_p$	$\{85, \ldots, 120\}$
$price_p$	$\{60, \ldots, 504\}$
$beamWidth_p$	$\{17, \ldots, 29\}$
$stiffness_p$	$\{55, \ldots, 75\}$
$gripSize_p$	$\{0, \ldots, 5\}$

flexibility and efficiency. To validate this concept, we conducted a preliminary offline assessment of unit-critiquing within a prototype system.

The remainder of this paper is organized as follows. In Section 2, we define the recommendation task using an example from the domain of tennis rackets. Section 3 describes the components involved in generating recommendations, along with their underlying principles. In Section 4, we present the prototype system and its interaction model. Section 5 showcases the evaluation of our approach and its results. In Section 6, we discuss the advantages of our approach. Finally, Section 7 concludes the paper with a discussion of open research issues.

2. Working Example

For demonstration purposes, we present a critiquing-based recommendation scenario within the domain of tennis rackets. Our approach, similar to constraint-based recommender systems, frames the task of selecting products that meet user preferences and needs (referred to as the *recommendation task*) as a CSP [5]. We define the recommendation task as a tuple $(V_{PROD}, V_C, C_{KB}, C_F, C_{PROD}, C_C)$, where each component is defined as follows:

- V_{PROD}: a set of variables describing the properties of a tennis racket (see Table 1).
- V_C: a set of variables describing the user profile (see Table 2).
- C_{KB}: a set of restricting constraints that systematically limit the possible instantiations of variables. For example, extremely narrow rackets require higher stiffness (see Table 3).
- C_F: a set of filter constraints based on the user profile. For example, a user with a history of arm injuries requires a racket with lower stiffness to help prevent discomfort or injury (see Table 4).
- C_{PROD}: a single constraint in *disjunctive normal form (DNF)* that defines the product catalog by specifying elementary restrictions on the possible values of variables in V_{PROD} (see Formula 1). Here, P is the set of all products, and v_i represents the value of product i with respect to the variable v.

$$C_{PROD} = \bigvee_{i \in P} \Big(\bigwedge_{v \in V_{PROD}} v_i \Big) \tag{1}$$

- C_C: a set of constraints specified by the user during interaction with the system to reflect the user's critiques. For example, this set might include a constraint like $brand_p \neq Dunlop$, excluding *Dunlop* from the variable $brand_p$. Initially empty, this set is updated as critiques are applied (see Section 3.2).

Given this definition, a solution to a recommendation task is an assignment of the variables in V_{PROD} and V_C, such that none of the constraints in C_{KB}, C_F, C_{PROD}, and C_C are violated [5].

Table 2
Variables of the set V_C describing the the user profile.

Variable	Domain
$gender_c$	{*male, female*}
$experience_c$	{*beginner, experienced*}
$knowledge_c$	{*basic, advanced*}
$playingProfile_c$	{*baseline, serve-volley, all-court, counterpuncher, defensive*}
$armInjury_c$	{*yes, no*}

Table 3
Restricting constraints of the set C_{KB}.

Constraint
$weight_p \leq 290 \vee headSize_p \geq 104 \implies beamWidth_p \geq 22$
$weight_p \geq 321 \implies beamWidth_p \leq 25$
$beamWidth_p \leq 19 \implies stiffness_p \geq 65$

Table 4
Filter constraints of the set C_F based on the user profile.

Constraint
$experience_c = beginner \implies weight_p \leq 300 \wedge headSize_p \geq 100$
$gender_c = female \implies weight_p \leq 305$
$armInjury_c = yes \implies stiffness_p \leq 63$

3. Generating Recommendations

In this section, we present the key components involved in generating recommendations and describe their core principles.

3.1. Modelling Preferences

Acquiring and modeling user preferences is essential in critiquing-based recommender systems. One common approach is to translate preferences directly into constraints to guide the search process [3]. However, this method restricts the search space, potentially excluding products that might better suit user needs. An alternative, outlined in [1, 3], uses *Multi-Attribute Utility Theory (MAUT)* to model preferences, accounting for conflicting values and scoring items based on their overall alignment with user preferences. Another method, mentioned in [3], employs a probabilistic model, which captures uncertainty and variability in user desires.

In our approach, we employ MAUT to model user preferences. Specifically, each tennis racket variable is assigned an importance and preference value, as well as a preference metric (see Section 3.4), initially set to *nearer-is-better*. These preferences are dynamically updated through user interactions (i.e., critiquing, see Section 3.2), and play a key role in adjusting the search heuristic (see Section 3.3) and determining the best recommendation when multiple or no solutions meet the user criteria (see Sections 3.4 and 3.5). Additionally, the selection and presentation of attribute-oriented system-suggested critiques are influenced by current user preferences (see Section 3.2). These concepts address the adaptive needs of the critiquing process, making the system more flexible and responsive to changes in user preferences.

3.2. Supporting Critiquing

To support critiquing, a hybrid system incorporating both user-initiated and system-suggested critiquing is proposed in [3]. *User-initiated critiquing* allows users to generate their own self-motivated critiques, while *system-suggested critiquing* involves presenting users with proposals to refine the current recommendation. As argued in [3, 12], suggested compound critiques can enhance system performance and should align with users' interests. Generally, critiques can be categorized as either unit or compound forms [4]. *Unit critiques* enable users to comment on individual item attributes (e.g., heavier), whereas *compound critiques* combine multiple unit critiques to address several features simultaneously.

In our working example, user-initiated critiquing is facilitated by the navigation panel described in [3], which allows users to choose whether to keep or improve a specific variable. As a result, within a single cycle, the user can submit either a unit or compound critique, with the system deciding on any variables not mentioned in the critique. In contrast, system-suggested critiques are generated based on the user's playing profile and racket attributes. These critiques are suggested as follows:

- *Profile-based Critiques (PBC)*: these critiques address various performance preferences and are suggested based on the user's playing profile. Each profile, as defined by the *knowledge base (KB)*, is associated with several PBCs, each linked to specific constraints. For instance, the *serve-volley* profile is paired with an "*Increased Serve Power*" PBC, which, when selected, applies increased $weight_p$ and $stiffness_p$ constraints. A PBC is recommended if the constraint solver identifies a solution that satisfies its constraints. The process begins with the user's playing profile and follows a similarity-based order, evaluating each profile and its PBCs in sequence until either three PBCs are identified or all profiles have been examined. Additionally, similarity is measured using the *Euclidean distance* between default profile attributes and their respective importance.
- *Attribute-oriented Critiques (AOC)*: these critiques are suggested using the approach described in [1]. Rackets are ranked by utility (see Section 3.4), and up to four distinct compound critiques (AOCs) are selected from this ranking, each highlighting the differences from the current recommendation. The order of attributes in each AOC is determined by the current importance of the variables (see Section 3.1), with more important variables and their differences listed before those of lower importance. This approach helps users efficiently understand and evaluate the key factors and differences while still providing an overview of other relevant attributes.

After the user selects or specifies critiques, these are translated into corresponding constraints. The system then performs the following steps on all variables in the set V_{PROD} (see Section 2):

1. *Update Importance*: the system updates the importance of variables depending on whether they are constrained in the current cycle or not (modified from [1]). The update is performed as follows:

$$importance(v) = importance(v) * \begin{cases} \lambda, & \textit{if } v \textit{ is critiqued} \\ \frac{1}{\lambda}, & \textit{otherwise} \end{cases}, \tag{2}$$

   where v is a variable and λ represents an importance factor, set to *1.5* in our working example.
2. *Store Preference Metrics*: the system stores the preference metrics (see Section 3.4) for variables based on critiques' impact on their domains. For example, if the user specifies a critique such as *"Heavier"*, the metric *more-is-better* is assigned to the variable representing the racket's weight. For each variable not constrained in the current cycle, the default metric *nearer-is-better* is applied.

Furthermore, as demonstrated in [13], monitoring successive critiques can significantly enhance recommendation efficiency. Consequently, we maintain a model that includes critiques selected by the user in previous cycles. Before adding a new critique (constraint) to the C_C set (see Section 2), we first remove any existing critiques from the model that are inconsistent with the new one.

3.3. Finding Valid Configurations

To find valid configurations, we deploy a custom search heuristic over the search space. As noted in [8], identifying valuable products and services within a set of constraints is often necessary in systems operating within domains where millions of potential recommendations are available. Consequently, we limit the number of solutions provided by the constraint solver to *15*. The following describes the custom variable and value-order selectors used by the system:

- *Variable-order Selector*: this selector prioritizes variables that are deemed more important at a given moment. For instance, if $importance(\textit{weight}_p) > importance(\textit{stiffness}_p)$, then the variable $\textit{weight}_p$ will be selected before $\textit{stiffness}_p$. This approach ensures that the instantiation of less important variables does not affect those with higher importance. Additionally, if constraints are relaxed (see Section 3.5), the affected variables will be given priority for selection.
- *Value-order Selector*: this selector considers the current preference metric of the corresponding variable (see Section 3.1). It assigns either the *IntDomainClosest* or *IntDomainMedian* value-order heuristic from the Choco [10] constraint solver to each variable. These selectors choose the value from the variable domain that is closest to the specified target value or the median value, respectively. If the preference metric is *nearer-is-better* or *equal-is-better*, the *IntDomainClosest* heuristic is used, with the target value set to the current recommendation's variable value or the new critique, respectively. Conversely, if the metric is *more-is-better*, *less-is-better*, or *in-range-is-better*, the *IntDomainMedian* heuristic is employed.

3.4. Calculating Utility of a Recommendation

When multiple solutions are possible, ranking the items becomes necessary. The method presented in [8] determines the degree of similarity between user requirements and recommendations. We adapt these metrics to meet our needs (see Formulae 4–8). The utility of a recommendation r is calculated as a weighted sum over the variables describing racket properties V, based on user preferences (see Formula 3). Specifically, $u(i, r_i)$ denotes the utility of recommendation r with respect to variable i, $prefval(i)$ is the preferred value for i stored in the user's preference model, and $range(i)$ corresponds to the constrained range of variable i. For example, if a critique indicates that the optimal racket's stiffness is mid-stiff, then the constrained range for $\textit{stiffness}_p$ is from *64* to *67*. The terms $minval$ and $maxval$ represent the minimum and maximum values for variable i according to the KB definitions. Additionally, the metrics are categorized as follows: *more-is-better* (*MIB*; e.g., a heavier racket is better), *less-is-better* (*LIB*; e.g., lower stiffness is better), *nearer-is-better* (*NIB*; e.g., stiffness closer to *65* is better), *equal-is-better* (*EIB*; e.g., a grip size of *2* is optimal), and *in-range-is-better* (*IRIB*; e.g., medium stiffness is preferable). The choice of metric for each variable depends on its assigned metric (see Section 3.1).

In our working example, calculating utility is crucial for ranking *attribute-oriented critiques* (see Section 3.2) and valid configurations. In the latter case, the item with the highest utility is recommended, and the preferred values of variables in the user's preference model are updated to match this item.

$$utility(r) = \sum_{i \in V} u(i, r_i) * importance(i) \tag{3}$$

$$MIB : u(i, r_i) = \frac{r_i - minval(i)}{maxval(i) - minval(i)} \tag{4}$$

$$LIB : u(i, r_i) = \frac{maxval(i) - r_i}{maxval(i) - minval(i)} \tag{5}$$

$$NIB : u(i, r_i) = 1 - \frac{|prefval(i) - r_i|}{maxval(i) - minval(i)} \tag{6}$$

$$EIB : u(i, r_i) = \begin{cases} 1, & if\ r_i = prefval(i) \\ 0, & otherwise \end{cases} \tag{7}$$

$$IRIB : u(i, r_i) = \begin{cases} 1, & if\ r_i \in range(i) \\ 0, & otherwise \end{cases} \tag{8}$$

3.5. Dealing with Unsatisfiable Critiques

In scenarios where no item satisfies the critiques (constraints), displaying a message such as "no product found" is highly undesirable [5]. Additionally, research indicates that users cannot always be relied upon to provide consistent feedback throughout a recommendation session [13].

To address this issue, we employ a linear-time constraint relaxation technique based on the data structure described in [5]. The core idea is to associate each constraint with a list of items that satisfy it. This approach makes it straightforward to identify which constraints need to be relaxed to include a specific product in the result set. We define the *Optimal Relaxation Set (ORS)* as follows:

- The ORS is a set of relaxations, where each relaxation is a set of constraints that includes the smallest number of new constraints (introduced in the current cycle) and is minimal in size.

For example, if product i requires the relaxation of two new constraints and one prior constraint (specified in previous cycles), while product k requires the relaxation of one new constraint and two prior constraints, then product k will be included in the ORS. After generating the ORS, we rank its repair alternatives using the relaxation utility formula (see Formula 9) presented in [14]. Here, *S* denotes the set of constraints to be relaxed, and $imp(c)$ represents the importance of a constraint *c* from the user's perspective, corresponding to the importance of the constrained variable.

$$relaxation\_utility(S) = \frac{1}{\sum_{c \in S} imp(c)} \tag{9}$$

If the top-ranked relaxation includes only prior constraints, it is automatically applied by removing the diagnosed user critiques (constraints). This method is known as hard relaxation (see [8]). Conversely, if the top-ranked relaxation includes at least one new constraint, the user is presented with a relaxation proposal that includes only the new constraints and omits prior ones. This approach helps prevent the user from being overwhelmed by potentially irrelevant information due to changes in preferences. Additionally, relaxation proposals are suggested based on the ranking. If the user does not accept any proposal, no constraints will be removed from the model. Conversely, accepting a proposal implies the application of hard relaxation to the constraints within the corresponding constraint set.

After performing the relaxation, the preference metric for variables affected by the relaxation (i.e., those involved in relaxing constraints) is set to *nearer-is-better*. These variables are then prioritized by the *Variable-order Selector* when identifying the next solution (see Section 3.3). Once this process is complete, the prioritization of these variables is removed.

4. Prototype System

The prototype follows the typical interaction model between users and a critiquing-based recommender system, as described in [3]. Initially, users are asked to provide their profile information, including gender, knowledge of tennis rackets, experience, playing profile, and any history of arm injuries. For users with basic knowledge of tennis rackets, the system requests additional details on the racket features they are familiar with.

After specifying their profile information, users can enter preferred values for known racket features (or all features, in the case of advanced knowledge), along with their importance on a scale from *0* to *5*, where *0* indicates the lowest importance and *5* the highest. The system stores these preferences and calculates the importance as follows: $importance(variable) = \frac{1}{n} + selected\_importance * \mu$, where n represents the number of racket variables, $selected\_importance$ is the chosen importance value, and μ is a constant factor, set to *0.2*. If users do not specify a preference for a particular variable, its default value and importance are assigned based on their profile.

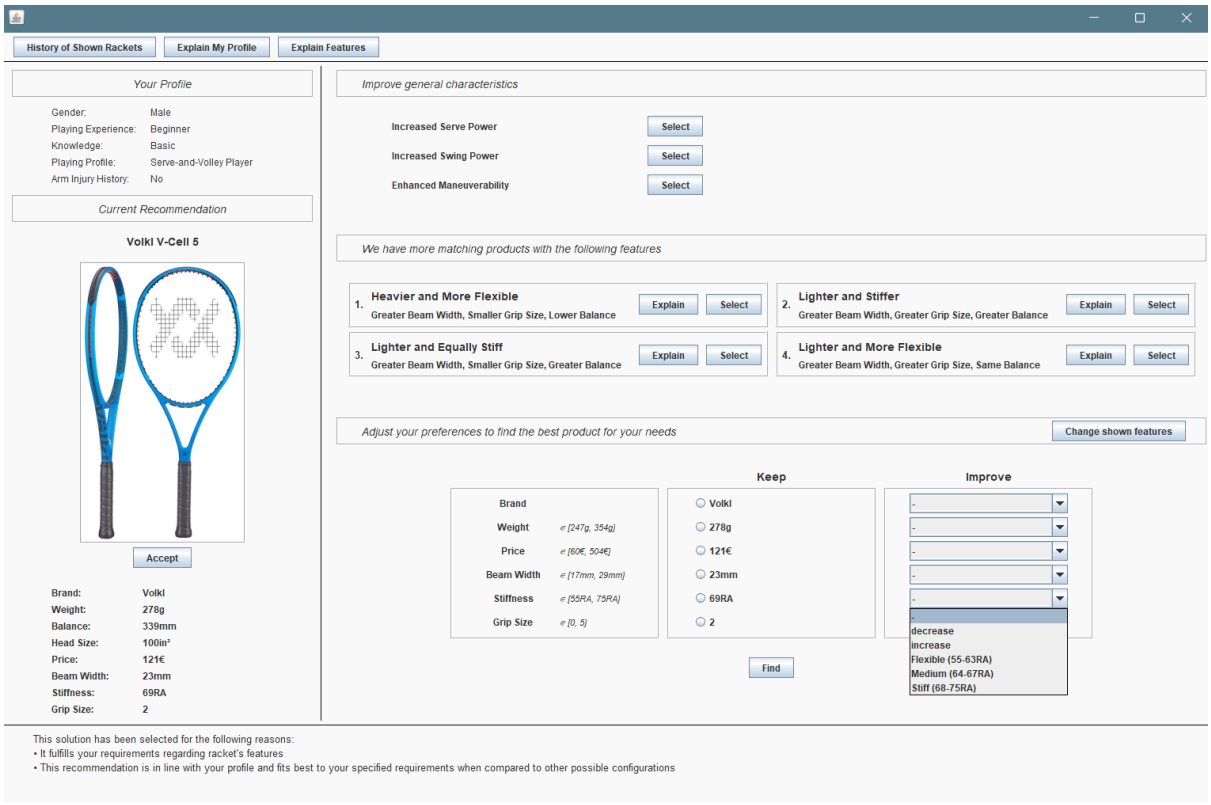

Figure 1: Critiquing interface of the tennis racket recommender. *Profile-based* and *attribute-oriented* critiques are displayed to the right of the user's profile information and the recommended racket. User-initiated critiquing support and the rationale behind the selected recommendation are provided below these elements.

The system then identifies and presents the best-fitting recommendation to the user (see Figure 1). The user can either accept this recommendation or provide critiques. Each *profile-based* and *attribute-oriented* critique can be selected to apply the corresponding critique. For users with advanced knowledge of tennis rackets, the suggestion of *profile-based* critiques is omitted. Additionally, *attribute-oriented* critiques follow a simple presentation pattern [15] and include an "Explain" option for a detailed explanation of the critique's impact on the search space. Conversely, the navigation panel for user-initiated critiquing allows users to manually specify critiques using the options from "*Keep*" and "*Improve*" components. The "*Improve*" component for a specific racket attribute includes possible critiques to modify the recommended product, such as *"decrease", "increase", "Flexible", "Medium"*, and *"Stiff"* for racket stiffness. For example, the user can choose to retain the current recommendation's brand by selecting the "*Keep*" option for the brand, while simultaneously constraining the stiffness by selecting the "*Medium*" option (see Figure 1). For users with advanced knowledge, this panel includes all racket features, while for basic users, it displays only features they identified as familiar, which can be modified using the "Change shown features" option. By clicking on the "Find" option, the system applies the selected critiques from the navigation panel. If no solutions meeting the new user criteria exist, the system presents relaxation proposals (see Figure 2). The "Previous Proposal" and "Next Proposal" options allow the user to navigate through a maximum of three proposals. The user can then either accept the proposal via the "Apply Relaxation" option, or select "Continue Without Relaxation".

Additionally, users can access explanations of their profile through the "Explain My Profile" option and receive details about racket features via the "Explain Features" option. These explanations facilitate a deeper exploration of the item space. Furthermore, the system incorporates explanations for the rationale behind the given recommendation or relaxation (see Figures 1 and 2), adhering to the criterion of satisfaction described in [16]. Finally, the "History of Shown Rackets" option displays the last three recommendations, their specifications, and an option to set them as the current recommendation again.

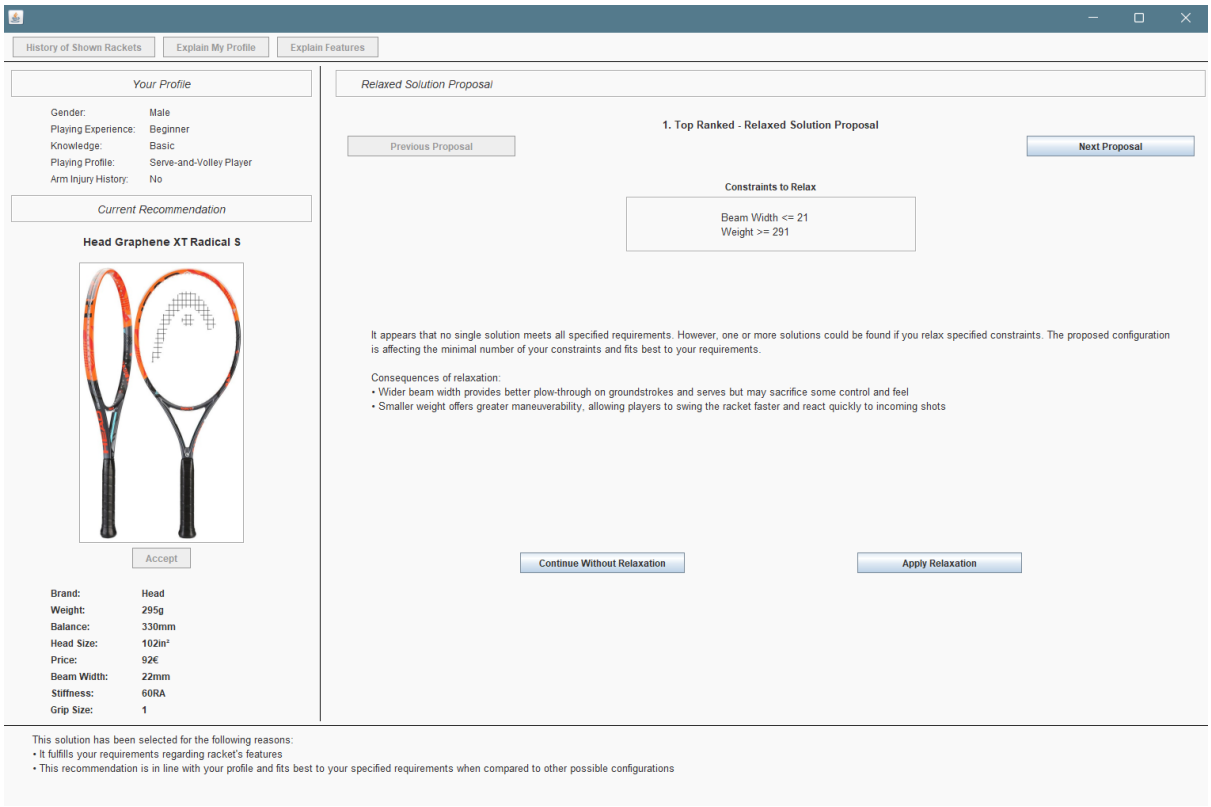

Figure 2: Relaxation proposal in the tennis racket recommender. The figure illustrates how constraints to be relaxed are presented to the user, along with their corresponding explanations.

5. Evaluation and Results

In this evaluation, we compared our approach with the conventional CBR approach by examining the performance of unit critiquing. For the evaluation, we used a custom dataset of *523* real-world rackets obtained from online selling platforms. Each racket is described by *9* variables (see Section 2), and all of these variables, except for the racket's name, were subject to critique. We conducted an offline experiment employing the *leave-one-out* approach described in [2]. At the start of each evaluation session, a random racket is temporarily removed from the dataset and used as the *target* for critiquing. Each session simulated a male, experienced user with a random playing profile, advanced knowledge of tennis rackets, and no arm injuries, ensuring unrestricted target selection. Additionally, two random variables are assigned preferred values matching those of the target, with importance randomly selected on a scale from *0* to *5*. Once these steps are completed and the initial recommendation is generated, the target racket is reintroduced into the dataset. To simulate user critiques, a variable differing from the target is randomly selected for critique, aiming to align the recommended racket with the target. This critique is specified using one of the corresponding "*Improve*" options (see Section 4). For example, if the selected variable is stiffness, and the recommended racket has a stiffness of *64* while the target racket has *67*, an *"increase"* option for stiffness is applied as a critique. This process continues until the target racket is recommended, marking the end of a single evaluation session. Finally, we measured performance based on the number of cycles required, conducting a total of *1000* sessions.

For generating recommendations, our approach follows the principles mentioned in previous sections. Conversely, in the conventional CBR approach, all items in the product catalog *CB* are modeled as product cases [6], which capture product details through predefined variables. When a user applies a critique c_i to a recommended item r, the goal is to find an item that satisfies c_i and is maximally similar to r [2, 6]. Specifically, items in *CB* that do not meet criteria of c_i are filtered out, and the next recommendation is selected from the remaining items based on their similarity with r. Similarity is measured using the utility described in Section 3.4, where each variable is assigned a *nearer-is-*

better metric, except for the variable mentioned in c_i, which is excluded from the utility calculation. Additionally, user preferences are modeled in the same way as in our approach, and the same user interface of the prototype system is utilized.

The evaluation results show that the conventional approach needed an average of *8.04* critiquing cycles to recommend the target item, whereas our approach required *5.18* cycles. These results highlight the potential efficiency and flexibility of our approach.

6. Discussion

In summary, conceptualizing the recommendation task within a critiquing-based recommender system as a CSP enables the effective utilization of constraint solving technology. Unlike conventional methods that typically require exhaustive scanning of the entire case-base to identify optimal products, a CSP-based approach narrows the solution space and employs a targeted search heuristic. This not only mitigates issues related to poor runtime performance but also has the potential to enhance the system's ability to identify highly preferred configurations, as indicated by the preliminary evaluation conducted in Section 5. This aligns well with the objective of aiding users in effective product configuration [5]. Additional key advantages include the flexibility provided by relaxation options and the diagnosis inherent in constraint technology. These features facilitate the efficient handling of scenarios where user requirements conflict with underlying constraints, as detailed in [5, 8].

7. Conclusions, Limitations and Future Work

This paper introduced a new approach to designing critiquing-based recommender systems by combining the strengths of critiquing with constraint solving technologies, particularly relevant for assisting users in configuring products. We demonstrated this approach within the context of a tennis racket recommender system by representing the recommendation task as a constraint satisfaction problem and detailing its key components for user support and recommendation generation. Additionally, we conducted a preliminary offline experiment to evaluate the performance of unit-critiquing within the prototype system. The results indicate that this approach is promising in terms of efficiency and flexibility. However, a limitation of our evaluation strategy is the reliance on simulated users, which may not fully reflect real-world scenarios. To address this, we plan to validate the approach in a real-world study to obtain practical insights. Future work will also focus on integrating additional concepts, such as optimizing performance. Finally, we plan to extend our approach by incorporating information from successfully completed critiquing sessions to enhance the efficiency of the critiquing process [2].

References

[1] J. Zhang, P. Pu, A Comparative Study of Compound Critique Generation in Conversational Recommender systems, in: V. P. Wade, H. Ashman, B. Smyth (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 234–243.

[2] M. Mandl, A. Felfernig, Improving the Performance of Unit Critiquing, in: J. Masthoff, B. Mobasher, M. C. Desmarais, R. Nkambou (Eds.), User Modeling, Adaptation, and Personalization, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 176–187.

[3] L. Chen, P. Pu, Critiquing-based recommenders: survey and emerging trends, User Modeling and User-Adapted Interaction 22 (2012) 837–852. doi:10.1007/s11257-011-9108-6.

[4] M. Uta, A. Felfernig, V.-M. Le, T. N. T. Tran, D. Garber, S. Lubos, T. Burgstaller, Knowledge-based recommender systems: overview and research directions, Frontiers in Big Data 7 (2024). doi:10.3389/fdata.2024.1304439.

[5] A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, Constraint-Based Recommender Systems, Springer US, Boston, MA, 2015, pp. 161–190. doi:10.1007/978-1-4899-7637-6_5.

[6] B. Smyth, Case-Based Recommendation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 342–376. doi:10.1007/978-3-540-72079-9_11.

[7] M. Torrens, B. Faltings, P. Pu, Smartclients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems, Constraints 7 (2002) 49–69. doi:10.1023/A:1017940426216.

[8] M. Atas, T. N. T. Tran, A. Felfernig, S. P. Erdeniz, R. Samer, M. Stettinger, Towards Similarity-Aware Constraint-Based Recommendation, in: F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, M. Ali (Eds.), Advances and Trends in Artificial Intelligence. From Theory to Practice, Springer International Publishing, Cham, 2019, pp. 287–299.

[9] E. C. Freuder, A. K. Mackworth, Chapter 2 - Constraint Satisfaction: An Emerging Paradigm, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, volume 2 of *Foundations of Artificial Intelligence*, Elsevier, 2006, pp. 13–27. doi:10.1016/S1574-6526(06)80006-4.

[10] C. Prud'homme, J.-G. Fages, X. Lorca, Choco documentation, TASC, INRIA Rennes, LINA CNRS UMR 6241 (2014) 64–70.

[11] A. Felfernig, A. Falkner, D. Benavides, Interacting with feature model configurators, in: Feature Models: AI-Driven Design, Analysis and Applications, Springer International Publishing, Cham, 2024, pp. 73–93. doi:10.1007/978-3-031-61874-1_4.

[12] J. Reilly, K. McCarthy, L. McGinty, B. Smyth, Dynamic Critiquing, in: P. Funk, P. A. González Calero (Eds.), Advances in Case-Based Reasoning, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 763–777.

[13] J. Reilly, K. McCarthy, L. McGinty, B. Smyth, Incremental Critiquing, in: M. Bramer, F. Coenen, T. Allen (Eds.), Research and Development in Intelligent Systems XXI, Springer London, London, 2005, pp. 101–114.

[14] A. Felfernig, R. Burke, Constraint-based recommender systems: technologies and research issues, in: Proceedings of the 10th International Conference on Electronic Commerce, ICEC '08, Association for Computing Machinery, New York, NY, USA, 2008. doi:10.1145/1409540.1409544.

[15] K. McCarthy, J. Reilly, L. McGinty, B. Smyth, On the Dynamic Generation of Compound Critiques in Conversational Recommender Systems, in: P. M. E. De Bra, W. Nejdl (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 176–184.

[16] N. Tintarev, J. Masthoff, Designing and Evaluating Explanations for Recommender Systems, Springer US, Boston, MA, 2011, pp. 479–510. doi:10.1007/978-0-387-85820-3_15.