Facing Scalability Issues in Requirements Prioritization with Machine Learning Techniques

Paolo Avesani, Cinzia Bazzanella, Anna Perini, Angelo Susi
ITC-IRST, Via Sommarive 18, I-38050, Trento, Italy
{*avesani,bazzanella,perini,susi*}*@irst.itc.it*

Abstract

Case-based driven approaches to requirements prioritization proved to be much more effective than first-principle methods in being tailored to a specific problem, that is they take advantage of the implicit knowledge that is available, given a problem representation. In these approaches, first-principle prioritization criteria are replaced by a pairwise preference elicitation process. Nevertheless case-based approaches, using the Analytic Hierarchy Process (AHP) technique, become impractical when the size of the collection of requirements is greater than about twenty since the elicitation effort grows as the square of the number of requirements.

We adopt a case-based framework for requirements prioritization, called Case-Based Ranking, which exploits machine learning techniques to overcome the scalability problem. This method reduces the acquisition effort by combining human preference elicitation and automatic preference approximation.

Our goal in this paper is to describe the framework in details and to present empirical evaluations which aim at showing its effectiveness in overcoming the scalability problem. The results prove that on average our approach outperforms AHP with respect to the trade-off between expert elicitation effort and the requirement prioritization accuracy.

1. Introduction

Requirements prioritization plays a key role in software development when we need to plan for system releases and to decide which requirements to implement in each release, according to budget and time constraints as well as to customer expectations [2]. Different factors influence the definition of priority criteria such as business aspects (e.g. market competition or regulations), customer satisfaction, or technical aspects (e.g. the development cost).

Requirements prioritization can be conceived as an order relation on a given set of requirements, on the basis of which it is possible to derive a partition into many subsets, one for each release. Usually, it is not crucial to get the relative order of core requirements, because all of them will be part of the first release, but the accuracy of the rank becomes more and more important when deciding which of the pending requirements can be removed from the subsequent releases.

Recent approaches [12, 14, 19, 21, 26] to requirements prioritization seem to share a common model for the requirements prioritization process, consisting of the following steps: *(i)* selection of one or more prioritization criteria among business goals and technical features (e.g. customer value, revenue or development cost); *(ii)* acquisition of a requirements ordering according to a specific criterion from one or more stakeholders (e.g. customers, users, project manager); *(iii)* composition of the acquired orderings into a final one based upon an appropriate composition schema. These steps are part of a process that can be iterated many times during the entire life-cycle of a software system.

First principle or *ex-ante* methods for requirements prioritization, define a-priory the ranking criteria (that is the requirements attributes — or priority indexes — and the relative ranges of values), independently of the current set of requirements that are to be evaluated. Partial orders are a possible outcome of *ex-ante* methods and can be the source of less effective decision making support.

Our work follows an alternative approach based on *ex-post* methods. Differently from first principle strategy, the elicitation of prioritization criteria is performed in parallel with the requirements analysis. A process of pairwise comparison allows to define at the same time which requirement and why it has to be preferred between two alternatives. That is, the prioritization criteria are not explicitly encoded but acquired by examples. This approach is known as case-based ranking and it is inspired by case-based reasoning [1], a problem solving paradigm where the solution is built by looking at examples rather than by using a first-principle knowledge representation.

The Analytical Hierarchy Process (AHP) [25] can be considered the reference method among those which adopt a case-based driven strategy. In this technique the ranking

Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE'05)
0-7695-2425-7/05 $20.00 © 2005 IEEE

criteria are defined through an assessment of the relative priority between a couple of requirements considered all the possible pairs of requirements. This becomes impractical as soon as the number of requirements increases and scalability problems limit severely the applicability of this technique.

A straightforward remedy may consist in reducing the amount of elicited pairs. Current solutions to scalability issues tend to define heuristics for supporting the choice of when the pairwise elicitation process can be stopped.

We propose an alternative strategy that aims at learning the prioritization criteria making a ranking hypothesis for the unknown pairs. We show that machine learning techniques can be effective in dealing with the scalability problem by providing an accurate approximation of the final ranking within a limited elicitation effort. Approximation does not necessarily mean poor quality results. A qualitative description of this framework and a first experimental evidence of its effectiveness has been presented in [4].

Our goal in this paper is twofold. First, we give a formal description of the framework and point out its main differences with case-based approaches using the AHP technique. Second, we describe a set of experimental evaluations which intend to characterize the effectiveness of our approach with respect to AHP based approaches when considering large set of requirements. In this analysis we take into account also state-of-the-art solutions to the AHP scalability problem.

The experimental results prove that in average our approach outperforms AHP with respect to the trade-off between expert elicitation effort and the requirement prioritization accuracy.

The paper is structured as follows. In Section 2, we describe in details our case-based prioritization framework and the referred machine learning techniques. In Section 3, we recall briefly how an AHP-based process looks like and recall the *stopping rule* technique that has been proposed as a solution to the scalability problem in AHP. In Sections 4 and 5, we describe the experimental evaluations and discuss their results. Related work are presented in Section 6 and conclusion are given in Section 7.

2. The Prioritization Methodology

We recently proposed a novel framework for prioritizing requirements [4]. It adapts the Case-Based Ranking (CBRanking) methodology described in [6] which exploits machine learning techniques to reduce the elicitation effort in the prioritization process. The framework rests on an iterative process that can handle single and multiple decision makers (stakeholders) and different criteria (both business goals and technical parameters).

Figure 1 sketches the basic steps of the prioritization process, where manual elicitation interleaves with machine supported steps.

The main input to the process is the finite collection of requirements $Req = \{r_1, r_2, \ldots, r_n\}$ that have to be ranked. The final output of the process is an approximation of the target ranking formulated as a function $H : Req \rightarrow \mathbb{R}$ where $r_i \prec r_j$ when $H(r_i) < H(r_j)$.

The basic process iteration rests on the following three main steps:

`Pair sampling`. It is an automatic procedure which selects a pair (or a sample of pairs) of requirements, (r_i, r_j), on the basis of a predefined selection policy which may take into account, information on the currently available rankings.

`Preference elicitation`. This step is in charge of the stakeholder. It takes in input a collection of pairs of requirements and it produces in output their ranks. More formally: at a certain iteration τ of the process, a function Φ_τ, simply Φ from now on, describes the preferences elicited from the user in terms of pair relations. In particular $\Phi : Req \times Req \rightarrow \{-1, 0, 1\}$ where $\Phi(r_i, r_j) = 1$ means that r_j be ranked above r_i, $\Phi(r_i, r_j) = -1$ means that r_i be ranked above r_j, and $\Phi(r_i, r_j) = 0$ indicates that no preference has been given between r_i and r_j (we assume $\Phi(r_i, r_j) = 0$ and $\Phi(r_i, r_j) = -\Phi(r_j, r_i)$ for all $r_i, r_j \in Req$).

`Ranking learning`. It takes in input the stakeholder preference Φ, and it computes an approximation of the ranking function $H(r)$ that, while preserving the elicited preferences, tries to make a ranking hypothesis of the unknown pairs. The learning procedures may exploit also available knowledge on the requirements rankings induced by other prioritization criteria (e.g. the cost for the realization of the requirements, the estimated utility) defined on the initial set of requirements. We call this knowledge ranking criteria and denote it as a finite set of m functions $F = (f_1, \ldots, f_l, \ldots, f_m)$, where $f_l : Req \rightarrow \overline{\mathbb{R}}$ ($\overline{\mathbb{R}} = \mathbb{R} \cup \{\perp\}$) and the inequality $f_l(r_i) > f_l(r_j)$ means that r_i is preferred to r_j according to the l^{th} criterion, while $f_l(r) = \perp$ if r is unranked with respect to the l^{th} criterion. The **`Ranking learning`** procedure exploits machine learning techniques, as detailed below.

The final ranking, that is the output of the process represents an approximation of the exact ranking and may become the input to a further iteration of the process. We denote the target ranking with the function $K : Req \rightarrow \mathbb{R}$ where r_j is ranked higher than r_i by K if $K(r_j) > K(r_i)$.

Notice that the step of **`Preference elicitation`** is usually a manual task; in the off-line simulation this task

Prioritization Process
Input: the set of requirements Req, previous iteration rankings $H(r)$
Output: the final Ranking $H(r)$

Iteration steps:
1. Pair sampling (Req, $H(Req)$);
2. Preference Elicitation ((r_i, r_j));
3. Ranking learning ($\Phi(r_i, r_j)$, F);

Figure 1. Basic steps of the prioritization process.

is performed by the machine through a sampling of the target ranking function K that is assumed to be given.

If the result of the learning step is considered enough accurate or the end user has been overloaded, the iteration halts and the latest approximated rank is given as output; otherwise another cycle of the loop is carried on. Notice that the first and the third steps are automated while the second step is in charge of the stakeholder. In the following, for simplicity, we will assume that the preference elicitation is monotonic (i.e. the user does not see the same pair twice).

To summarize we can conceive the prioritization process as an approximation problem where, given a set of requirements $Req = \{r_i\}$ and a subset of pairwise priority relations $\Phi_\tau \subseteq \Phi$, the challenge is to learn a function $H(r)$ such that $\forall r_i, r_j$ we have $H(r_i) > H(r_j)$ if $K(r_i) > K(r_j)$ where $K(r)$ is the unknown target prioritization criteria. Of course the objective is twofold: to minimize the elicitation effort, while reducing the disagreement between the target (K) and the approximate rank (H).

2.1. The learning algorithm

The **`Ranking learning`** step produces an approximation of a preference structure, exploiting the boosting approach described in [10]. In the following we give a brief description of the problem that we handle with the boosting approach and of the algorithm.

The basic concepts, we introduced so far, are the set of requirements, the ranking features, the exact (or target) ranking and the elicitated pairwise preferences. The input to the algorithm are described below.

- A finite set of requirements $Req = \{r_1, \ldots, r_n\}$.
- The ranking criteria $F = (f_1, \ldots, f_m)$.
- The initial user preferences represented by the function $\Phi_0(r_i, r_j)$
- Related to the Φ we also define a density function D : $Req \times Req \rightarrow \mathbb{R}$ such that

$$D(r_i, r_j) = \gamma \cdot max(\{0, \Phi(r_i, r_j)\})$$

setting to 0 all negative entries of Φ; γ is a positive constant chosen in such a way that D is a distribution, satisfying the following normalization property:

$$\sum_{r_i, r_j} D(r_i, r_j) = 1$$

The goal of the learning step is to produce a ranking of all requirements in Req. This ranking is represented in the form of a function $H : Req \rightarrow \mathbb{R}$. The function H represents the approximated ordering of Req induced by the user preference function Φ using the information from the set of features F; the objective is to minimize a measure of error called ranking loss $rloss$, which is defined in the following. Given a pair r_i, r_j, the pair is said to be *crucial* if $\Phi(r_i, r_j) > 0$, so that the pair receives non-zero weight under D. The algorithm we use has been designed to find an order function H with a small weighted number of crucial-pair misorderings, in other words with a small ranking loss $rloss_D(H)$ defined as:

$$\begin{aligned} rloss_D(H) &= \sum_{r_i, r_j} D(r_i, r_j) [\![H(r_j) \leq H(r_i)]\!] \\ &= Pr_{(r_i, r_j) \sim D}[H(r_j) \leq H(r_i)] \end{aligned}$$

where $[\![H(req_j) \leq H(req_i)]\!] = 1$ if $H(req_j) \leq H(req_i)$ is true, 0 otherwise.

In our framework, the function H is computed by an adaptation of the boosting method that is able to produce highly accurate prediction rules by combining many weak rules which may be moderately accurate; here we refer to the boosting algorithm introduced in [9], which is sketched, in pseudocode, in Figure 2.

The algorithm *RankBoost* performs T iterations; it takes as input the initial distribution D and the set of functions F and gives as output the final hypothesis H in the form of a linear combination of partial order functions $h_t : Req \rightarrow \mathbb{R}$ with a set of coefficients $\alpha = \{\alpha_1, \ldots, \alpha_t, \ldots\}$ (where t is the iteration index).

Algorithm **RankBoost**
Input: the set of requirements $Req = \{r_1, \ldots, r_i, r_j, \ldots, r_n\}$,
the set of ranking criteria F, the function Φ_τ, the initial distribution D
Output: the final Hypothesis $H(r)$

begin
$D_1 = D$;
For $t = 1, \ldots, T$:
$h_t = WeakLearner(Req; \Phi_\tau, F, D_t)$ where $h_t : Req \rightarrow \mathbb{R}$;
Choose α_t, where $\alpha_t \in \mathbb{R}$;
$D_{t+1}(r_i, r_j) = \frac{D_t(r_i,r_j)}{Z_t} e^{\alpha_t(h_t(r_i)-h_t(r_j))}$, where $D_{t+1} : Req \times Req \rightarrow \mathbb{R}$
return $H(r) = \sum_{t=1}^{T} \alpha_t h_t(r)$;
end.

Figure 2. A sketch of the RankBoost algorithm.

The basic iteration t performs the three steps described below.

- Compute a partial order $h_t : Req \rightarrow \mathbb{R}$ via the function $h_t = WeakLearner(Req; \Phi_\tau, F, D)$ of the elements in Req tacking into account the function Φ_τ, the ranks induced by the functions in F, and the distribution D_t. In our experiments we referred to the $WeakLearner$ function described in [9]; it is a binary classifier that in every iteration t produces a dichotomy on the sets of requirements and defines a precedence relationship among the resulting subsets.

- Compute a value for the parameter α_t. This value is a measure of the accuracy of the partial order h_t respect to the final order H.

- Compute a new distribution D over the set of pairs whose value has been given by the users, which is passed, on the next iteration, to the procedure that computes the partial order h_t. The algorithm, in fact, uses the distribution D to emphasize some pairs in $Req_\Phi \times Req_\Phi$; D is computed as in the equation: $D_{t+1}(r_i, r_j) = \frac{D_t(r_i,r_j)}{Z_t} e^{\alpha_t(h_t(r_i)-h_t(r_j))}$.
In particular at the iteration t a high value for D_{t+1} assigned to a pair of requirements indicates a great importance that the function h_{t+1} (so the $WeakLearner$) orders that pair as indicated by the user.

The total number of iterations, T, can be fixed a-priori or the algorithm stops when a stable ordering configuration has been found.

3. The AHP prioritization process

The Analytic Hierarchy Process (AHP) [25] is a multiple criteria decision making techniques based on a pairwise comparison approach. It has been largely applied in software engineering, for instance in software package and component selection [17, 18], in COTS evaluation [20] and in requirements prioritization [14] .

From a practical point of view, when using AHP for prioritizing requirements, the first step is the representation of the set of requirements under investigation in a matrix whose rows and columns represent the candidate requirements. Given a prioritization criterion, the second step implements a pairwise comparison process where each element of the matrix[1] are assigned an integer belonging to the interval $[1 \ldots 9]$ which represents a qualitative measure of the preference relation between the corresponding requirements (e.g. if the requirement A is "equally important" than requirement B respect to the given criterion, the value 1 is given, if the requirement A is "essentially more important" than requirement B the value 5 is given). When this step has been completed, a total order is synthesized through the computation of a vector of weights that specifies the rank of each requirement.

In case of multiple criteria, the whole process is repeated for each criterion and a further step is required, that is the synthesis of a global rank based on a weighted composition of the different criteria orderings is computed. The weights are derived using an analogous preference elicitation process performed on a matrix where rows and columns represent the different criteria.

Among the main, well known, limits of AHP, the growth of the number of comparisons needed as long as the number

[1] More precisely, half of the matrix values is elicited while the other half is computed by symmetry.

of candidate requirements increases. Notice that even with a small set of requirements, say 10, it is necessary to elicit 45 pairwise preferences, for each criterion[2].

In Saaty [25] is proposed a technique to handle this scalability problem by introducing the so called "dominance hierarchy" whose top levels elements refer to criteria and the lowest level correspond to the requirements. Intuitively, following the dominance hierarchy, the general prioritization problem is decomposed into sub-problems allowing for a reduction of the elicitation effort, but, at the same time introducing a strong bias. In fact the dominance hierarchy reflects the a-priori knowledge on the relative importance of the criteria, independently from the current candidate requirements.

A recent approach to handle the AHP scalability problem in prioritizing requirements has been proposed in [14]. This approach rests on the exploitation of the *local stopping rule* technique proposed in [11]. This technique allows to determine when new pairwise comparisons are no longer needed.

More precisely, adopting the notation introduced in the previous section, we can define the function K_{AHP} that represents the correct ranking and the function $H_{AHP}(req)_\theta$ that represents the ranking computed by the AHP algorithm at a certain stage θ of the pairwise elicitation step (i.e. the second step of the AHP process described above). The *stopping-rule* can be represented by the expression (1)

$$(|H(r_i)_{\theta-1} - H(r_j)_\theta| < a) \tag{1}$$

where a is a positive real number.

4. Empirical Evaluation

In this section we present two types of experimental evaluation aiming at characterizing the effectiveness of our approach for a large set of requirements: the first consists in comparing the CBRanking approach to AHP when prioritizing requirements sets of increasing cardinality; the second aims at verifying the applicability and the effectiveness of the stopping rule technique, briefly recalled in Section 3, a state of the art solution to overcoming the AHP scalability problem.

Both experimental evaluations rest on a simulation of the prioritization process for a given target ranking, in other words we assume that a simulated decision maker knows the preference function K for a predefined set of requirements, or, equivalently the correct prioritization criterion. More precisely, a simulation of the prioritization process can be accomplished as follows. First a subset of pairs (req_i, req_j) is selected from the cartesian product $Req \times Req$ with the restriction that $i \neq j$ and $(req_i, req_j) \neq (req_j, req_i)$. In particular, in the CBRanking experiments, the initial subset of pairs has cardinality $n/2$ and it is chosen in order to guarantee that each requirement is a member of at least one pair in the subset, while for AHP a spanning tree (composed by $n-1$ pairs) is chosen, as described in [15]. The following step, related to preference elicitation, simulates the user behavior retrieving the $\Phi(req_i, req_j)$ value by sampling directly the given preference function K. For simplicity we restrict the simulation to a monotonic behavior and we assume that a decision maker acts without giving inconsistent answers during the process. The third step, invokes the RankBoost algorithm to produce an estimate of the proper prioritization, $H(req)$, fully defined over the set Req resulting in an approximated priority rank.

We call this experimental approach off-line empirical evaluation, in contrast to the on-line approach which involve human decision-makers. The usefulness of the off-line experimentation when setting up a set of on-line experiments have been presented in [4] where also a set of on-line tests aiming at measuring the accuracy of the final ranking corresponding to a given elicitation effort, have been discussed.

Notice that the complexity of a prioritization process it is not directly dependent on the choice of a specific preference function K but to the relationship that holds between a given preference function K and the set of ranking features F. As illustrated in [5] it is possible to recognize four different types of relationships that correspond respectively to an increasing prioritization complexity: isotonic, anti-tonic, non-monotonic, non-projective. According to this considerations, we generated a set of prioritization problems of different degrees of difficulty.

The first experimental evaluation has been conducted on sets of requirements with cardinality n equal to 10, 25, 50, 100 respectively. On a given set of requirements, we run two prioritization processes, one based on our approach, the other on AHP. In both processes we computed the the disagreement corresponding to a given number of elicitated pair preferences. That is, given a pair (r_i, r_j) we introduce the measure $dp(r_i, r_j)$ which assumes the value 1 if the functions H, K gives different ordering relationships for that pair, and 0 otherwise, namely:

$$dp(r_i, r_j) = \begin{cases} 1 & \text{if } (H(r_i) < H(r_j) \wedge K(r_i) > K(r_j)) \\ & \vee (H(r_i) > H(r_j) \wedge K(r_i) < K(r_j)) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The total $disagreement(H, K)$ of the ordering function H respect to the target ranking K is then defined in the formula (3).

$$disagreement(H, K) = \frac{1}{n(n-1)} \sum_{i,j,i \neq j} dp(r_i, r_j) \tag{3}$$

[2]The number of comparison required goes as $n(n-1)/2$, where n is the number of candidates

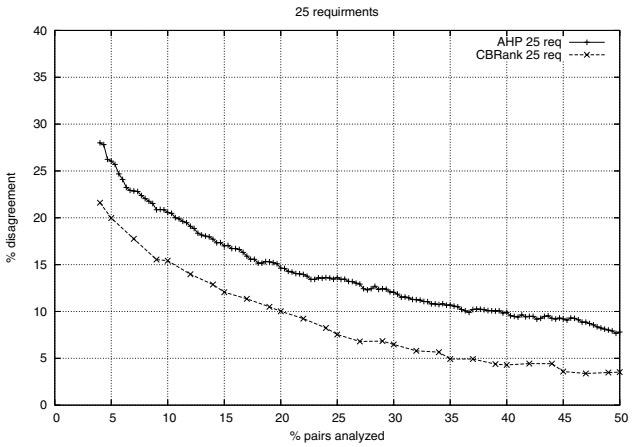

Figure 3. The plot of disagreement (y-axis) for a set of 25 requirements in AHP and the CBRanking frameworks for an increasing number of elicitated pairs (x-axis).

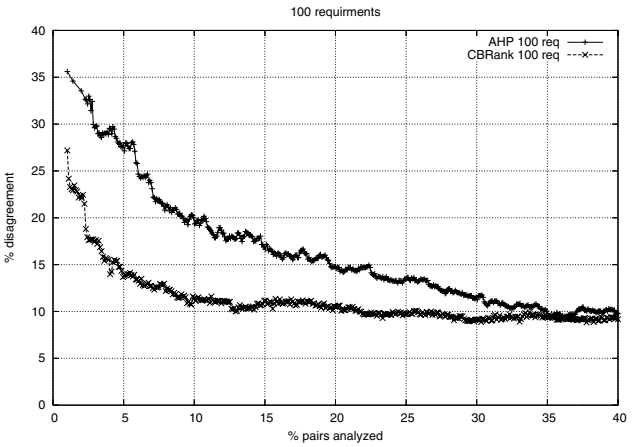

Figure 5. The plot of disagreement (y-axis) for a set of 100 requirements in AHP and the CBRanking frameworks for an increasing number of elicitated pairs (x-axis).

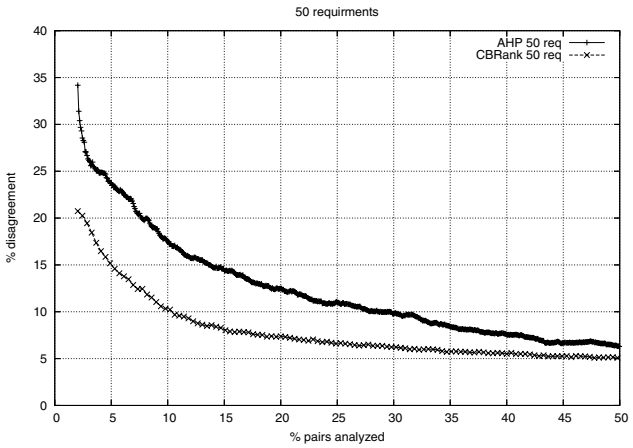

Figure 4. The plot of disagreement (y-axis) for a set of 50 requirements in AHP and the CBRanking frameworks for an increasing number of elicitated pairs (x-axis).

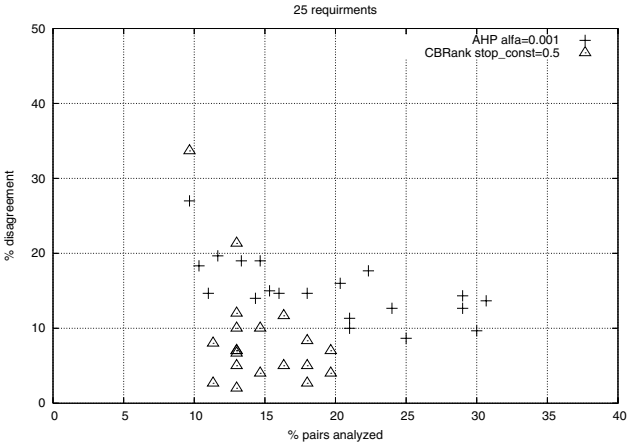

Figure 6. The result of the application of the stopping rule over a set of 20 run of both the Case-Based Ranking and AHP process for prioritization the same set of 25 requirements.

The values plotted in Figure 3, have been obtained as the average of the disagreement values measured on ten runs. On the x axis is shown the percentage of requirements pair preferences that have been elicited (which is a measure of the elicitation effort). Notice that 5% of requirements pairs corresponds to the cardinality of the set of requirements. This corresponds to the amount of evaluations that are usually requested by methods that are linearly related to the number of requirements (within the hypothesis of monotonicity).

In Figure 3, 4 and 5 are shown the disagreement measures in the case of 25, 50 and 100 requirements respectively.

The second set of experiments focuses on the application of the *local stopping rule* briefly described in Section 2, that have been used both with AHP and with our approach.

A critical problem when using this technique is how to choose a-priori, the right value for the parameter a, having in mind the precise value for the maximum elicitation effort at which the process should be stopped.

The input to this second type of experiments is a set of

requirements, a value for K and a value for $\mathtt{a}$. The output is a measure of the disagreement computed for the elicited pairs number at which the process stop.

In Figure 6 are shown some results. In particular the diagram represents the results over twenty AHP runs, on the same set of 25 requirements and for $\mathtt{a} = 0.001$. Every point in the diagram represents the final value of one of these twenty runs.

The stopping rule technique has been adapted for our methodology and values for the parameter $\mathtt{a}_{CBR}$ for our approach which may be compared to those for $\mathtt{a}_{AHP}$ has been found experimentally.

The results obtained running our approach with the stopping rule, on the same set of requirements, and with a comparable value for the stopping parameter $\mathtt{a}$ are depicted in the same plot.

Note that other values for parameter $\mathtt{a}$ have been tested giving analogous results. The choice of showing the results for $\mathtt{a}_{AHP} = 0.001$ and respectively for $\mathtt{a}_{CBR} = 0.5$ is because they cover a region of the elicitation effort from 10% to 30%, which is a region of interest with respect to the previously discussed experiments.

5. Discussion

The first set of experiments allows a deep analysis of the trade-off between the elicitation effort and the ranking accuracy, in the two approaches, namely CBRanking and AHP, and provides an empirical evidence that the first uniformly outperforms the second.

First, the CBRanking framework results more effective especially for lower values of elicited pairs, as shown in the plots depicted in Figure 3, 4, 5, where, for instance the improvement in the approximation accuracy obtained with the CBRanking framework with respect to that obtained with AHP, at 5% of the elicited pairs is greater then the relative improvement measured at 10% of the elicited pairs. This is particularly remarkable when we are considering large numbers of requirements since, in practice, only a small portion of pairs can be manually elicited. For example, the 10% pairwise analysis of 200 requirements requires to perform around 2000 comparisons.

Moreover, if we focus our attention to the behaviour of the two techniques for low elicitation effort (the most interesting case for practical approaches, as claimed above) we can see that the improvement of CBRanking with respect to AHP increases when considering larger and larger sets of requirements. Considering the approximated ranking obtained with an elicitation effort of 5% of pairs, it is characterized by a 20% of disagreement in the case of 25 requirements, by a 15% of disagreement in the case of 50 requirements and by a a 10% of disagreement in the case of 100 requirements (see Figure 3, Figure 4 and Figure 5 respectively). This can be considered a relevant empirical evidence of the effectiveness of the CBRanking framework in dealing with the scalability issue. During the experiments the effectiveness of the spanning tree initialization strategy used in AHP have been tested also in our framework. The results show that the adoption of this strategy does not produce a faster convergence of the *RankBoost* algorithm.

The results discussed so far concern with the average behaviour analysis. Since the % disagreement measured with the CBRanking technique has a greater variance with respect to that measured with AHP, the results of CBRanking can be much more better. The analysis of complexity of a requirement prioritization process is out of the scope of this paper and is discussed in details in [5]; here it is important to notice that one of the source of complexity in the CBRanking method, other than the complexity related to number of requirements, is the relationship between the target ranking K and the domain knowledge represented by the ranking criteria in F; they can give effective support to the ranking process in the case they specify a ranking close to the target ranking. Nevertheless it is important to remark, even though not depicted in the plots, that the worst behaviour of CBRanking performs at least as AHP.

The successful results of CBRanking can be extended to the analysis of *stopping-rule*. The achievement is twofold: better average accuracy and lower effort variance. The first result is of course a straightforward consequence of the previous achievement. Since CBRanking outperforms AHP, if we apply a policy to reduce the elicitation effort the final results will be related to the general behaviour of the two techniques.

A meaningful additional result is concerned with the variance of elicitation effort. The behaviour of *stopping-rule*, as technique to reduce the elicitation effort, is strongly context sensitive. The final outcome is an high variance of the numbers of pairs that have to be elicited. For example, in our experiment with 25 requirements depicted in Figure 6, the percentage of elicitation effort spans from 10% to 30% for 10 iterations of the same problem. For the same experiment the variance of CBRanking is half with respect to AHP: the values are in the range between 10% and 20%. It is important to notice that for both the methods the results show a high variance with respect to the disagreement obtained when the process is stopped by the rule, so, especially for AHP, the low effort in the elicitation process is not balanced with the accuracy of the final ranking.

The behaviour of CBRanking seems to get much more effective the use of *stopping-rule*. Not only CBRanking halves the variance of elicitation effort but at the same time it lower the upper bound of the number of pairs that have to be elicited. In our experiment illustrated in Figure 6, the worst case of 30% is reduce to 20%. As mentioned in advance, reducing the elicitation effort is dual problem of

scaling with respect to the number of requirements.

6. Related work

In this paper we focused on how to deal with the scalability problems that arise in managing the prioritization of a large number of requirements. Among the relevant approaches that face these issues, we already mentioned [15] which proposes the concept of *stopping-rule* to reduce the pairwise elicitation effort. This rule allows to define a threshold on the variation of the estimated requirements ordering under which new comparisons are no longer needed. In Section 5 we pointed out critical aspects related to the fact that the value of the threshold has to be chosen *a-priori*, and showed how our approach can limit them.

Considering the requirements prioritization in a more general perspective, in [8] we can find a discussion of the possible purpose and benefits, challenges and risks that arises during the process of requirement prioritization as well as a classification of currently used techniques. Among them, of particular interest for our work are the following. The Cost-Value approach, described in [14], exploits the AHP technique for the evaluation of requirements against two main criteria: business value, that is the ability of a requirement to contribute to the customer satisfaction, and cost of implementation. The result is then plotted in a cost-value diagram that shows in a visual way the relative position of the requirements. This composition of the two requirements rankings offers a rather effective method for classifying the requirements, nevertheless the method suffers from all the limits of the AHP technique and in particular those related to the scalability.

In [21] is proposed another AHP-based methodology named Soft Requirements Negotiator (SRN). The SRN method aims at addressing the incompleteness and uncertainty of the initial set of requirements to be prioritized and for this reason integrates AHP, multi-criteria approaches and simulation techniques for the estimation of quantitative ranking features. Multi-criteria techniques are exploited with the attempt to deal with incomplete information, and in particular to support the selection of balancing "for" and "against" arguments for a given requirement.

Other studies on basic multi-criteria decision making methods to be used during requirements prioritization are [23], SMART [7], Quality Function Deployment (QFD) [3], the Multi-criteria Preference Analysis Requirements Negotiation (MPARN) [12], and Quantitative WinWin [24].

Main limits of the above mentioned approaches are related to the strong assumptions which are inherent to the decision making techniques they adopt, such as, the completeness and certainty of the set of requirements to be evaluated and the plausibility of a rating scale based on discrete categories. Moreover most of them typically do not scale unless the requirements are previously grouped in some manner. This considerations [19, 26] motivated the development of methodologies which are less accurate, but more easy-to use and less time consuming that can be preferred in industrial practice. For instance, in [22] is described an experimental study where an AHP based approach is compared to an approach based on Planning Game, a technique used in Extreme Programming [16], showing that the second approach was preferred.

Other works propose the integration of different decision making techniques as well as methods for the identification of relevant criteria for requirements prioritization derived from other disciplines, (e.g. portfolio-based reasoning) [12, 26].

Concerning the specific techniques that has been exploited in our CBRanking framework, in [13] are proposed interesting modifications to the boosting algorithms, and in particular to the Ada Boost algorithm [10]. A new algorithm has been defined in order to cope with the problem of overfitting and with the problem of improving the selection of the most promising patterns that are able to approximate the final target ranking; this may result in a better behavior of the algorithm in finding a good approximated solution with reducing the elicitation effort for the user.

7. Conclusions

In this paper we presented in details a novel framework for requirements prioritization, called Case-Based Ranking. Similarly to the AHP process, our framework adopts an elicitation process based on the acquisition of pairwise preferences, differently from AHP it enables a prioritization process even over a large set of requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations from the acquired data.

The Case-Based Ranking framework has been evaluated with respect to AHP on simulated data. We described this set of tests and discussed the results which shows that our framework is effective in dealing with large sets of requirements. Basically, they prove that in average, our approach outperforms AHP with respect to the trade-off between expert elicitation effort and the requirement prioritization accuracy and that in the worst case, it works as well as AHP technique.

Current results seems to be promising and we are going to further investigate the framework addressing other issues of requirements prioritization such as the negotiation among the points of view of different stakeholders, the handling of requirements dependencies and of "anytime" prioritization, a relevant issue when new unexpected requirements have to be added. Moreover, in [4] we addressed the problem of evaluating the effectiveness of the CBRanking method in a real setting. Here an interesting emerging problem is repre-

sented by the need of setting up methods for the validation of the experimental results.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, adn system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] A. Abran, J. Moore, and R. Dupuis, editors. *SWEBOK - Guide to the Software Engineering Body of Knowledge*. IEEE, 2001. http://www.swebok.org.

[3] Y. Akao. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press, 1988.

[4] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Supporting the Requirements Prioritization Process. A Machine Learning approach. In *Proceedings of 16th International Conference on Software Engineering and Knowledge engineering (SEKE 2004)*, pages 306 – 311, Banff, Alberta, Canada, June 2004. KSI press.

[5] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Exploiting Domain Knowledge in Requirements Prioritization. In *Proc. of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE 2005)*, Taipei, Taiwan, July 2005. KSI press.

[6] P. Avesani, S. Ferrari, and A. Susi. Case-Based Ranking for Decision Support Systems. In *Proceedings of ICCBR 2003*, number 2689 in LNCS, pages 35 – 49. Springer-Verlag, 2003.

[7] W. Edwards and F. Baron. *Smarts and smarter: Improved simple methods for multiattribute utility measurement*, pages 306 – 325. Number 60. 1994.

[8] D. Firesmith. Prioritizing Requirements. *Journal of Object Technology*, 3(8):36 – 47, 2004.

[9] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. In *Proceedings 15th International Conference on Machine Learning*, 1998.

[10] Y. Freund and R. Schapire. A Short Introduction to Boosting. Journal of Japan. Soc. for Artificial Intelligence, 14(5) (1999), 771-780. 11, 1999.

[11] P. Harker. Incomplete Pairwise Comparisons in the Analitic Hierarchy Process. *Mathematical Modeling*, 9:837–848, 1987.

[12] H. In, D. Olson, and T. Rodgers. Multi-Criteria Preference Analysis for Systematic Requirements Negotiation. In *26th Annual International Computer Software and Applications Conference*, Oxford, England, August 2002.

[13] R. Jin, Y. Liu, L. Si, J. Carbonell, and A. Hauptmann. A New Boosting Algorithm Using Input-Dependent Regularizer. In *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*, Washington, DC, USA, August 2003. AAAI Press.

[14] J. Karlsson. Software requirements prioritizing. In *ICRE'96*, 1996.

[15] J. Karlsson, S. Olsson, and K. Ryan. Improved Practical Support for Large Scale Requirements Prioritizing. *Journal of Requirements Engineering*, 2:51 – 67, 1997.

[16] K.Beck. *Extreme Programming Explained*. Addison-Wesley, 1999.

[17] J. Kontio. Otso: A systematic process for reusable software component selection, 1995.

[18] N. A. M. Maiden, P. Pavan, A. Gizikis, H. K. O. Clause, and X. Zhu. Integrating Decision-Making Techniques into Requirements Engineering. In *Proceedings of REFSQ'02 workshop in RE'02 Conference*, Essen, Germany, 2002.

[19] F. Moisiadis. The fundamentals of prioritising requirements. In *System Engineering, Test and Evaluation Conference*, Sydney, Australia, 2002.

[20] D. Morera. COTS Evaluation Using Desmet Methodology & Analytic Hierarchy Process (AHP). In *Proceedings of the PROFES 2002 conference*, Rovaniemi, Finland, December 2002. Springer-Verlag.

[21] A. Ngo-The and G. Ruhe. Requirements Negotiation under Incompleteness and Uncertainty. In *Software Engineering Knowledge Engineering 2003 (SEKE 2003)*, San Francisco, CA, USA, July 2003.

[22] B. Regnell, M. Host, J. N. och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, 2001.

[23] B. Roy and D. Bouyssou. *Aide Multicritère à la Decision: Methods et Cas*. Economica, Paris, 1993.

[24] G. Ruhe, A. Eberlein, and D. Pfahl. Quantitative winwin - a quantitative method for decisione support in requirements negotiation. In *Proceedings 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 159 – 166, Ischia, Italy, July 2002.

[25] T. L. Saaty. Fundamentals of the analytic network process. In *Proceedings of International Symposium on Analytical Hierarchy Process*, 1999.

[26] A. Sivzattian and B. Nuseibeh. Linking the selection of requirements to market value: A portfolio-based approach. In *REFS 2001*, 2001.