A Machine Learning Approach to Software Requirements Prioritization

Anna Perini, *Member*, *IEEE*, Angelo Susi, *Member*, *IEEE*, and Paolo Avesani, *Member*, *IEEE*

Abstract—Deciding which, among a set of requirements, are to be considered first and in which order is a strategic process in software development. This task is commonly referred to as requirements prioritization. This paper describes a requirements prioritization method called Case-Based Ranking (CBRank), which combines project's stakeholders preferences with requirements ordering approximations computed through machine learning techniques, bringing promising advantages. First, the human effort to input preference information can be reduced, while preserving the accuracy of the final ranking estimates. Second, domain knowledge encoded as partial order relations defined over the requirement attributes can be exploited, thus supporting an adaptive elicitation process. The techniques CBRank rests on and the associated prioritization process are detailed. Empirical evaluations of properties of CBRank are performed on simulated data and compared with a state-of-the-art prioritization method, providing evidence of the method ability to support the management of the tradeoff between elicitation effort and ranking accuracy and to exploit domain knowledge. A case study on a real software project complements these experimental measurements. Finally, a positioning of CBRank with respect to state-of-the-art requirements prioritization methods is proposed, together with a discussion of benefits and limits of the method.

Index Terms—Requirements management, requirements prioritization, machine learning

1 Introduction

The problem of prioritizing software requirements amounts to ranking a set of desired functionalities and features of the intended software along one or more concerns such as business aspects (e.g., market competition or regulations, customer satisfaction) or technical aspects (e.g., development costs or risks).

Requirements prioritization plays a crucial role in software development, and in particular it allows for planning software releases, combining strategies for budget management and scheduling, as well as market strategies. It is, in fact, considered a complex multicriteria decision-making process.

State-of-the-art approaches [21], [30], [32], [37] tend to share a common model for this process, which consists of the following steps.

1. The definition of a target criterion for ordering.
2. The specification of requirement attributes to encode the chosen criterion.
3. The acquisition of specific values for those attributes, for all requirements under consideration.
4. The composition of rankings induced by requirement attributes associated to the target criterion.

More precisely, the first step is concerned with the selection of the most appropriate prioritization criterion according to specific strategic goals, such as reducing the development costs or minimizing the overhead of bug fixing. The identification of requirement attributes in the second step is performed in a way to define univariate ranking functions on the requirements set. For example, with reference to the goal of reducing development costs and the choice of "development cost" as a target ranking criterion, requirement attributes such as the estimated number of "lines of code" or of "components" are suitable. The third step, namely, the acquisition of attribute values over the set of requirements, usually represents the most expensive task in the prioritization process since it rests on the availability of expert knowledge or on the elicitation of evaluations from stakeholders. Since a target criterion might be encoded by manifold attributes and each attribute induces a ranking of the requirements set, the fourth step is concerned with the composition of the different attribute-based rankings into a global ordering corresponding to the target criterion. This composition is usually defined in terms of a weighted aggregation schema.

The widespread use of such a process model for requirements prioritization is also confirmed by a survey review that considered more than 200 papers in the field of requirements prioritization for benefit and cost criteria [12]. The assumption underlying the analyzed approaches is that the ranking criteria, the requirement attributes, and the way to compose them in case of multicriteria ranking can be defined independently of the nature of the current set of requirements under evaluation. In other words, they adopt an *ex-ante* perspective about the requirements prioritization problem which prevents exploiting available knowledge on the project's application domain. In contrast, an *ex-post* perspective will enable the exploitation of this knowledge through a prioritization process that is built on the actual set of requirements under evaluation and will lead to a different realization of steps 2 to 4. Namely, project stakeholders are asked to perform a pairwise comparison of the current

• *The authors are with the Fondazione Bruno Kessler, CIT-IRST, Povo, Via Sommarive 18, I-38123 Trento, Italy.*
E-mail: {perini, susi, avesani}@fbk.eu.

Manuscript received 17 Nov. 2010; revised 16 Dec. 2011; accepted 15 July 2011; published online 25 July 2011.
Recommended for acceptance by T. Tamai.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-11-0343.
Digital Object Identifier no. 10.1109/TSE.2012.52.

requirements, allowing them to decide which requirement is to be given a higher rank between two alternatives without the need to identify a specific requirement attribute to encode the evaluation criterion adopted by the stakeholder. So, for instance, the users of an e-voting system may be asked to decide which of the requirements "Graphical layout of the voting form" and "Getting audio feedback during the voting procedure" is more important (this evaluation will be repeated for different requirement pairs).

The difference between *ex-ante* and *ex-post* approaches can be summarized as follows. While in the *ex-ante* perspective the target criterion is chosen in advance, in *ex-post* approaches project stakeholders are required to evaluate pairs of requirements along an underlying target criterion. Consequently, requirements ranking is not computed from the values of requirement attributes, but it is derived from the priority relations that are elicited directly from stakeholders, who may take into account implicit information that might not have been preliminarily encoded as requirement attributes. The composition of rankings in case of multistakeholder prioritization is provided as instances (examples) of pairwise priority relations and not as the result of the application of an analytical composition schema. An interesting advantage of eliciting input regarding relative values rather than absolute values for attributes is that the noise on the input is recognized to be lower.

The *ex-post* perspective derives from the problem solving paradigm known as case-based reasoning [1], according to which a solution to a new problem is inferred from examples of solutions to similar problems.

The Analytical Hierarchy Process (*AHP*) [35] can be considered the reference method among those which are based on the case-based paradigm. In this method, the ranking criteria are defined upon an assessment of the relative priority between a couple of requirements, expressed by project stakeholders. This assessment encompasses all possible pairs of requirements. The effort required by the human evaluator when pair preferences are elicited grows rapidly with the number of requirements since the number of pairs grows quadratically.

This makes *AHP* difficult to use with large sets of requirements, a problem that is typically dealt with by defining ad hoc heuristics for deciding when the pair preference elicitation process can be stopped without compromising the accuracy of the resulting ranking.[1] This may be a main reason for *ex-post* approaches being less commonly used than *ex-ante* approaches in requirements prioritization practices.

Analogously to *AHP*, in our work we follow an *ex-post* perspective to the requirements prioritization problem and propose a method called Case-Based Ranking (*CBRank* from now on). Differently from *AHP*, *CBRank* adopts a highly flexible preference elicitation process which rests on the following basic properties. First, it allows for combining sets of preferences elicited from human decision makers with sets of preferences, which are automatically computed through machine learning techniques. These techniques exploit knowledge about (partial) orders of the requirements that may be encoded in the description of the requirements itself (i.e., in terms of the actual requirement attributes), thus enabling what we call *domain adaptivity*. This accounts for the straightforward applicability of *CBRank* to different application domains and for the fact that the accuracy of machine-estimated ranking increases with the level of significance of the encoded domain knowledge.

Second, *CBRank* is organized according to an iterative schema which allows for deciding when to stop the elicitation process on the basis of a measure of the tradeoff between the elicitation effort and the accuracy of the resulting ranking. With a reasonable effort, the method can be applied up to 100 requirements.

The objective of this paper is to offer a detailed and comprehensive presentation of the *CBRank* method, providing:

1. a formal definition of the prioritization problem it solves,
2. an intuitive description of the machine learning technique it is based on and a characterization of the prioritization process supported by *CBRank*,
3. a comprehensive overview of the empirical measurements which have been performed to assess key properties of the method, and
4. a positioning of *CBRank* with respect to state-of-the-art requirements prioritization methods.

The description of the empirical evaluations conducted on *CBRank* points out the different empirical study techniques that have been applied. In summary, by simulating the prioritization process we collected experimental data at support of the analysis of the effort versus accuracy tradeoff and of the domain adaptive property of the *CBRank* method. Preliminary results have been presented in [4], [5], and [6] and data from new simulations are discussed in this paper. We investigated the effectiveness of the domain adaptivity property of *CBRank* when used in a real case study. This empirical study complements the previous experiments that are based on simulated prioritization processes. We executed a controlled experiment with 23 subjects aiming at comparing two tool-supported versions of *CBRank* and *AHP* with respect to ease-of-use of the methods, time-consumption for performing the prioritization task, and accuracy of the resulting ranking. In this paper, we briefly recall the main result of this experiment and refer the interested reader to [33] for a full description.

The paper is structured as follows: Section 2 defines the ranking techniques and the prioritization process of *CBRank*, Section 3 gives an overview of the set of empirical studies that have been performed on *CBRank* and defines the variables that are used to characterize the method's properties, among them, the *disagreement* between the computed ranking approximation and the target ranking, which is used to measure the accuracy of the ranking obtained applying the method. The experimental studies carried out to evaluate the effort versus accuracy tradeoff and the domain adaptivity property of *CBRank* are then recalled along with their design, execution, and the analysis of the results in Sections 4 and 5, respectively. The new

1. Notice that *AHP* has also been exploited as single-criterion requirements ranking in *ex-ante* approaches [23]. In these cases, the evaluator was chosen in such a way that she/he could use the preassigned criterion to evaluate the requirement pairs.

empirical study performed on a case study extracted from a real project devoted to the development of an e-voting system is illustrated in Section 6. Related work and positioning of *CBRank* with respect to state-of-the-art prioritization methods and techniques is presented in Section 7 and complemented with a discussion about the advantages and the limits of *CBRank* in Section 8. Finally, Section 9 concludes and outlines future work.

2 The Case-Based Rank Method

The *CBRank* method rests on a framework, first introduced in [7] and [8], which supports decision-making for ordering a set of items, e.g., product features or software requirements. The framework provides an iterative prioritization process that can handle single and multiple human decision makers (stakeholders) and different ordering criteria. A peculiarity of this framework is the use of machine learning to reduce the elicitation effort, that is, the amount of information required from stakeholders, for achieving rankings of a given quality degree. Besides the problem of requirements prioritization [4], [5], [6], the framework has been applied to the problem of prioritizing test cases in software testing [38].

In order to illustrate *CBRank*, we first define a set of basic concepts that help describe the prioritization process, then we introduce the specific machine learning techniques it is based on in terms of an algorithmic procedure that we apply to a toy example to give an intuitive account of how the algorithm works.

2.1 Concepts

We consider a finite collection of *Requirements* $Req = \{r_1, \ldots, r_n\}$ that has to be ranked, and for it define the *Universe of pairs* as the set of requirement pairs $U = \{(r_i, r_j); i < j\}$. We call the order relation between two requirements that can be elicited from a decision maker (e.g., a stakeholder) *Priority*. We define it formally in terms of the function $\phi(r_i, r_j)$, where $\phi : U \to \{-1, 0, 1\}$, with the following meaning for its values:[2]

$$\phi(r_i, r_j) = \begin{cases} -1, & \text{if } r_j \prec r_i, \\ 1, & \text{if } r_i \prec r_j, \\ 0, & \text{if there is no order preference between } r_i \text{ and } r_j. \end{cases} \quad (1)$$

An *Unordered Requirement Pair* is a pair of requirements (r_i, r_j) the decision maker has not yet assigned a priority, that is, its priority is unknown. While describing the prioritization process we need to consider the set of *Ordered Requirement Pairs* at a given process iteration τ, namely, $\Phi_\tau = \{(r_i, r_j); i < j | \phi(r_i, r_j) \neq 0\}$. The set changes monotonically upon subsequent process' iterations so that the following property holds: $\Phi_\tau \subseteq \Phi_{\tau+1}$.

We define *Ranking Functions* to be the rankings derived by the attributes of each requirement, such as the estimated cost for its implementation or importance for the user. These orderings can be conceived as the set of ranking functions $F = \{f_1, \ldots, f_l, \ldots, f_m\}$, where f_l is defined as $f_l : Req \to \overline{\mathbb{R}}$ ($\overline{\mathbb{R}} = \mathbb{R} \cup \{\perp\}$) and $f_l(r_i) > f_l(r_j)$ means that r_i is preferred to r_j according to the lth attribute, while $f_l(r) = \perp$ if r is unranked with respect to the lth attribute.

The *Target Rank* represents the "ideal" preference ranking on a given set of requirements (that—in real settings—is generally not known). Formally, the target rank can be conceived as a function $K : Req \to \mathbb{R}$, where r_i has a lower rank than r_j, by K, if $K(r_i) < K(r_j)$.

We define *Final Approximated Rank* as the rank produced as output by the *CBRank* process. It is an approximation of the *Target Rank* K. Formally, the approximated ranking is a function $H : Req \to \mathbb{R}$, where r_i has a lower rank than r_j, by H, if $H(r_i) < H(r_j)$. We also define $\hat{H}_\tau$ as the *Approximated Rank*, at process iteration τ, as $\hat{H}_\tau : Req \to \mathbb{R}$, where r_i has a rank lower than r_j, by $\hat{H}_\tau$, if $\hat{H}_\tau(r_i) < \hat{H}_\tau(r_j)$, and $\hat{H}_\tau = H$ when τ corresponds to the last process iteration, $\tau = \Theta$. Further concepts are needed to define heuristics for the selection of pairs that are worth being evaluated, namely, the concept of *Density* function, which is related to the definition of elicited pairwise preferences, Φ, and the concept of ranking error that we will call *rloss*.

The density function is defined as $D : Req \times Req \to \mathbb{R}$ or, in terms of the elicited pairwise preference function, Φ, as

$$D(r_i, r_j) = \gamma \cdot max(\{0, \Phi(r_i, r_j)\}). \quad (2)$$

γ is a positive constant chosen in such a way that D is a distribution, which satisfies the following normalization property:

$$\sum_{r_i, r_j} D(r_i, r_j) = 1. \quad (3)$$

Given the function H, the ranking loss $rloss$ is defined as follows:

$$rloss_D(H) = \sum_{r_i, r_j} D(r_i, r_j) [\![H(r_j) \leq H(r_i)]\!], \quad (4)$$

where $[\![H(r_j) \leq H(r_i)]\!] = 1$ if $H(r_j) \leq H(r_i)$ (when $H(r_i) \leq H(r_j)$ is expected), 0 otherwise.

Given a pair r_i, r_j, the pair is said to be *critical* if $\Phi(r_i, r_j) > 0$ so that the pair gives a nonzero contribution to D.

2.2 The Prioritization Process

The *CBRank* requirements prioritization process interleaves human activities with machine computation. The process is sketched in Fig. 1, where three steps are represented as rounded corner rectangles. The basic artifacts in input and output (see dashed arrows) are: the set of ***Requirements*** (Req), the decision maker's ***Priorities*** (Φ_τ), the set of ***Ranking Functions*** (F), encoding the requirement attributes, and the ***Approximated Rank*** ($\hat{H}_\tau$) or the ***Final Approximated Rank*** (H) when τ corresponds to the last process iteration. Additional artifacts are produced by internal activities of the process's steps, namely, the set of ***Sampled Requirements Pairs*** (a subset of $U \setminus \Phi_\tau$) that is the set of requirements pairs for which the end user preference is unknown and that have been selected for the following priority elicitation. The set of ***Ordered Requirements Pairs*** is a set of requirements pairs ordered by the decision makers (the subset $\Phi_\tau \setminus \Phi_{\tau-1}$).

2. The $\prec$ symbol between r_i and r_j indicates that the element at the left (r_i) is preferred to the element at the right (r_j).

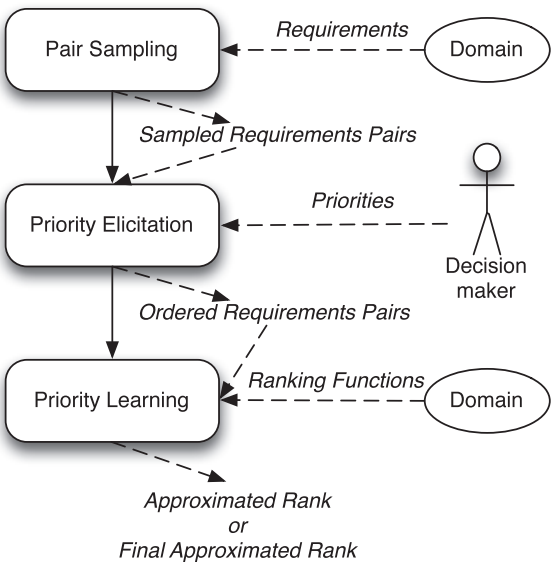

Fig. 1. Basic steps of the requirements prioritization process in *CBRank*.

The process is based on Θ iterations of the three steps, which are herein detailed.

1. Pair Sampling. An automated procedure selects from the set of *Requirements* a set of *Sampled Requirements Pairs* whose relative preference is unknown (i.e., *Unordered Requirement Pair*s, as defined in (1)), according to a sampling policy. A sampling policy can be a random choice or it may take into account the rankings computed in the previous iteration.[3]
2. Priority Elicitation. This takes the collection of *Sampled Requirements Pairs* produced by the Pair Sampling step in input and produces as output a set of *Ordered Requirements Pairs* on the basis of the *Priorities* expressed by a decision maker.
3. Priority Learning. Given a partial elicitation of the stakeholder priority and eventually a set of *Ranking Functions*, the learning algorithm produces an approximation of the unknown preferences and then the correspondent *Approximated Rank* for the requirements.

The *Approximated Rank*, that is, the output of the process, represents an approximation of the exact ranking and may become the input for a further iteration of the process. If the result of the learning step is considered accurate enough (or time to input preferences runs out), the iterations stop and the process gives the last approximated rank (Final Approximated Rank) as output.

Notice that the first and the third steps of the process are automatic. Concerning the second step, we will assume—for simplicity—that the preference elicitation is monotonic (i.e., the decision maker does not evaluate the same pair twice).

Using the concepts introduced in the previous section, the entire prioritization process can be conceived as an approximation problem where, given a finite set of requirements Req and a set of pairwise priority relations between requirements of the kind $r_i \prec r_j$ specified by the stakeholder, the challenge is to learn the *Final Approximated Rank* $H(r)$ 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 ranking (*Target Rank*). The objective is twofold: to minimize the elicitation effort, while reducing the disagreement between the target (K) and the approximate rank (H).

3. Examples of sampling policies could be that of selecting pairs of requirements that appear in the first part of the computed rankings in order to improve accuracy there, or pairs including a requirement taken from the first half of the current ranking and the other from the second half, with the purpose of checking the correctness of their relative position in the actual ranking [7].

2.3 The Priority Learning Techniques

The Priority Learning step produces an approximation of a preference structure, adapting the boosting approach described in [15] and [16]. The boosting method is able to produce highly accurate prediction of the rank by linearly combining many partial orderings which may be moderately accurate.

In the rest of this section, we give an intuitive description of the boosting approach by sketching the *RankBoost* algorithm, given as pseudocode in Algorithm 1, and simulating its execution on a toy example.

Algorithm 1. A sketch of the *RankBoost* algorithm

Input

$Req = \{r_1 \dots r_n\}$
The set of requirements

$F = \{f_1 \dots f_k\}$
Partial orders defining priorities and constraints upon Req

$\Phi = \{(r_i, r_j); i < j | \phi(r_i, r_j) \neq 0\}$
A subsample of elicited pairwise preferences

Output

$H(r)$ $(H : Req \rightarrow \mathbb{R})$
A ranking function defined upon Req

Begin

1: $W =$ *initialize* (Φ)
Uniform weighting of elicited pairs Φ
2: $T =$ *maxNumberOfCycles*
Definition of parameter for the number of learning cycles
3: **For** $t = 1$ **To** T
4: $h_t(r) =$ *LearningWeakClassifier*(Req, Φ, F, W)
Training a weak (e.g., binary) classifier with respect to a weighted set of pairs and most promising $f \in F$
5: $\alpha_t =$ *ComputeAlpha*(h_t, W)
Computation of the coefficient for the linear combination of weak classifiers
6: $W =$ *WeightingCriticalPairs*(Φ, W, h_t)
Weighting of elicited pairs Φ according to misclassification error of h_t
7: **Endfor**
8: $H(r) = \sum_{t=1}^{T} \alpha_t h_t(r)$
Synthesis of ranking function as composition of weakbinary classifiers
9: **return** $H(r)$

End

The algorithm *RankBoost* performs T cycles; it takes the set of ranking functions F in input and gives as output the

final ranking 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 cycle index).

The basic cycle t performs the three steps described below.

- Computes a partial order $h_t : Req \rightarrow \mathbb{R}$, called weak classifier, on the basis of the set of requirements *Req*, the set of the *elicitedPairs* Φ, the *Ranking Functions* F, and the set of *criticalPairs* W eventually identified in the previous cycle and that have to be considered by the procedure to minimize the error of the classifier to be computed with respect to user preferences and ranking attributes. Several possible kinds of weak classifiers can be considered for partially ordering the set of requirements; in our experiments we refer to the $h(t)$ as described in [15] and [16]; it is a binary classifier that produces a dichotomy, in every cycle t, on the sets of requirements and defines a precedence relationship among the resulting subsets.
- Computes a value for the parameter α_t. This value is chosen to minimize the error between H, the user preferences, and the functions F.
- Computes the set of critical pairs W which is passed on to the next cycle of the procedure to compute the partial order h_{t+1} in such a way as to minimize the final ranking loss with respect to user preferences (*elicitedPairs*) and Ranking Functions F. An example of how to compute the set W of *critical pairs* is that of updating the values of the distribution D, assigning high values to pairs that have not been correctly classified by the function h_t with respect to user feedback Φ. These pairs should be correctly ordered in the next cycle of the algorithm when h_{t+1} is computed.

2.4 Example

An intuitive comprehension of the learning process can be achieved by applying the algorithm to a toy example, step by step. Let us consider a prioritization problem defined over a set of four requirements $Req = \{r_1, r_2, r_3, r_4\}$, where a couple of ranking attributes and the associated functions are given $F = \{f_1, f_2\}$. The first ranking function, which is defined as follows, $f_1(r_3) = 1$, $f_1(r_2) = 2$, $f_1(r_1) = 3$, and $f_1(r_4) = 4$, introduces a priority relation where $r_4 \prec r_1 \prec r_2 \prec r_3$; in a similar way, the second ranking function is defined as $f_2(r_1) = 1$, $f_2(r_3) = 2$, $f_2(r_4) = 3$, and $f_2(r_2) = 4$, according to the following priority relation: $r_2 \prec r_4 \prec r_3 \prec r_1$. Let us suppose that the decision-maker accomplishing the preference elicitation task has considered the pairs (r_1, r_2) and (r_2, r_4), producing a subsample of the priority relation, namely: $\Phi = \{r_2 \prec r_1, r_4 \prec r_2\}$. Let us suppose that the above defined *Req*, F, and Φ are given as input to the *RankBoost* algorithm sketched above, and simulate the execution of the numbered steps as follows:

Initialization—Algorithm 1: lines 1-2. At the beginning all elicited pairs Φ are considered equally important, i.e., $w_{1,2} = w_{2,4} = 0.5$, where $w_{i,j}$ corresponds to the weight for the elicited preference on the pair (r_i, r_j). Moreover, for the purpose of this example, let's consider two iterations of the inner loop, i.e., the number of cycles: $T = 2$.

Inner loop—Algorithm 1: lines 4-6, first iteration: $t = 1$. The step at line 4, is devoted to making a hypothesis for a weak learner. Here it has chosen an instance-based approach and—for the sake of simplicity—considered one of the ranking function as weak classifier; therefore h_1 might be f_1 or f_2. The choice will be driven by the criterion of loss minimization. For $h_1 = f_1$, the loss is computed as $w_{1,2} \cdot Loss(r_1, r_2) + w_{2,4} \cdot Loss(r_2, r_4)$, where $Loss(r_1, r_2) = f_1(r_2) - f_1(r_1) = -1$ and $Loss(r_2, r_4) = f_1(r_4) - f_1(r_2) = 2$. On the other side, for $h_1 = f_2$ the loss is $Loss(r_1, r_2) = f_2(r_2) - f_2(r_1) = 3$ and $Loss(r_2, r_4) = f_2(r_4) - f_2(r_2) = -1$. Since the loss for the hypothesis $h_1 = f_1$ is lower than $h_1 = f_2$, 0.5 versus 1, respectively, the chosen weak learner corresponds to the attribute f_1. The step at line 5 computes coefficients for the linear combination of weak classifiers. As a simplifying assumption, the expression $\alpha(t) = 1/t$ is chosen and, for the first cycle, this amounts to $\alpha(t = 1) = 1$. Finally, the step at line 6 updates the weighting schema. Since the loss on pair (r_2, r_4) is higher than (r_1, r_2), the weights are revised accordingly: $w_{1,2} = 0$ and $w_{2,4} = 1$.

Inner loop—Algorithm 1: lines 4-6, second iteration: $t = 2$. During the second cycle $t = 2$ the step at line 4 computes the weak learner h_2 with reference to the pair (r_2, r_4) only. The computation of the loss is restricted to one pair only: $f_1 : Loss(r_2, r_4) = f_1(r_4) - f_1(r_2) = 2$ and $f_2 : Loss(r_2, r_4) = f_2(r_4) - f_2(r_2) = -1$. The best hypothesis for the weak learner is $h_2 = f_2$ and the composition parameter $\alpha(t = 2) = 0.5$ (step at line 5).

Final Step—Algorithm 1: line 8. The ultimate step is the aggregation of weak learners to synthesize the final approximation of ranking function: $H = \alpha_1 h_1 + \alpha_2 h_2 = 1 \cdot f_1 + 0.5 \cdot f_2$. The application of such an H to our set of requirements *Req* gives $H(r_1) = 3.5$, $H(r_2) = 4.0$, $H(r_3) = 2.0$, and $H(r_4) = 5.5$, corresponding to the following priority relation: $r_4 \prec r_2 \prec r_1 \prec r_3$.

The example shows how the learning algorithm computes a priority relation for the whole set of requirements, although not all of them have been analyzed by the decision-maker, namely, r_3.

3 Empirical Evaluations

The evaluation of prioritization methods is not a straightforward task. Although the empirical evaluation can be carried out by controlled experiments with real subjects or by simulations with artificial data and virtual subjects, both approaches suffer from specific limitations. A main limit of the former is the lack of ground truth, namely, the target rank, which prevents a quantitative measure of the quality of the resulting priority rank; in the latter, the lack of real subjects does not allow for taking into account nondeterministic factors of real world settings.

For a comprehensive empirical assessment of the properties of *CBRank* and to limit the weaknesses intrinsic to the evaluation approaches mentioned above, we used different evaluation settings: simulations with synthetic data and a case study with stakeholders. A summary of the empirical evaluations described in this paper is given in Table 1. They are characterized in terms of the main goal of the

TABLE 1
Overview of Empirical Measurements Performed on *CBRank*

	Experiment 1 Section 4	**Experiment 2** Section 5	**Experiment 3** Section 6
Goal	Analyze the effort vs. accuracy trade-off for two different prioritization methods: *CBRank* and *AHP*	Analyze the *domain adaptivity* property of the *CBRank* method, also with some comparisons with *AHP*	Analyze the *domain adaptivity* property of the *CBRank* on a real case study
Independent variable	Prioritization methods: *CBRank*; *AHP* (with local stopping rule [22])	Prioritization methods: *CBRank*; *AHP*. Different sets of attributes (with non-projective and isotone ordering with respect to the target ranking)	The tool-supported prioritization method *CBRank* (SCORE)
Dependent variables and measures	Elicitation effort (measured as number of —or percentage of— requirement pairs preferences); accuracy (measured in terms of disagreement)	Elicitation effort (measured as number of —or percentage of— requirement pairs preferences); accuracy (measured in terms of disagreement)	Elicitation effort (measured as percentage of requirement pairs preferences); accuracy (measured in terms of partition- disagreement)
Empirical Study Approach	Simulation with synthetic data; set of req. of size 25, 35, 50, 75, 100	Simulation with synthetic data; set of req. of size 25, 35, 50, 75, 100	Case Study: *e-voting* system; subjects: 6 project's stakeholders; 46 requirements concerning voting operations; management; and definition of the voting procedures

experiment, independent, dependent variables, empirical evaluation approach, and other factors, following empirical study terminology [42].

The goal of Experiment 1 concerns the analysis of effort versus accuracy tradeoff, which becomes relevant when the number of requirements increases, while Experiments 2 and 3 aim at investigating domain adaptivity property of the *CBRank* method, which refers to the capability of the method to exploit the knowledge encoded in the requirement formulation.

In Experiment 1, we evaluate *CBRank* through experiments based on simulations of the requirements prioritization process over problems whose complexity is varied along the following dimensions: number of the requirements to be prioritized, number of pair-preferences to be elicited from stakeholders, and accuracy of the resulting prioritization. Comparisons with a state-of-the-art method such as *AHP* and related extensions are conducted. The ultimate goal is to investigate whether *CBRank* outperforms the state-of-the-art solutions with respect to the tradeoff between accuracy and elicitation effort.

With analogous experiments we evaluate the property of domain adaptivity in Experiment 2. The research question is to understand whether the performance of *CBRank* is dependent not only on user behavior but also on the specific instance of the requirement prioritization problem. Our goal is to collect evidence that *CBRank* is able to exploit the knowledge encoded in the requirement formulation, if any, and to produce better results accordingly. We complemented the evaluation of domain adaptivity by a case study with real stakeholders on a software project for e-voting in Experiment 3.

Simulated experiments on effort versus accuracy tradeoff and domain adaptivity are illustrated in Sections 4 and 5, respectively, while the case study is described in Section 6.

A further empirical study on *CBRank* is reported in [33]. It consists of a controlled experiment devoted to the comparison between tool-supported *AHP* and *CBRank* along *time-consumption*, *ease of use*, and *accuracy* properties. The results of this experiment show, in summary, that *CBRank* overcomes *AHP* with respect to the first two properties, while for the accuracy of the resulting ranking, *AHP* performs better than *CBRank*, even if the resulting ranks from the two methods are very similar. Moreover, the majority of users found *CBRank* to be the *overall best* method.

3.1 Disagreement Measures

Details on the measures for the dependent variables of the experiments are given here. The basic reference measure consists of a quantitative assessment of the disagreement between two order relations defined over the same set of requirements.

Given two rankings for a set of requirements Req, called A and B, respectively, we define the disagreement on a pair of requirements (r_i, r_j) (where r_i, $r_j \in Req$) as

$$d_{A,B}(r_i, r_j) = \begin{cases} 1, & A(r_i) < A(r_j) \quad \text{and} \quad B(r_i) > B(r_j) \\ & \text{or} \\ & A(r_i) > A(r_j) \quad \text{and} \quad B(r_i) < B(r_j), \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

A comprehensive measure of disagreement can be obtained by computing the pairwise disagreement over the universe of requirement pairs. The total disagreement (simply TDA in the rest of the paper) of the two rankings A and B is defined as

$$TDA(A, B) = \sum_{U} d_{A,B}(r_i, r_j), \tag{6}$$

and its normalized definition is

$$NDA(A, B) = \frac{1}{|U|} TDA(A, B), \tag{7}$$

where the normalization factor ensures that NDA will assume values between 0 and 1.

In some cases it might be helpful to adopt a weaker measure to evaluate the relationships between partial

orders. A partial order on a set of requirements Req can be defined as a partition over a ranking function H as $H_p : Req \rightarrow P$, where $P = \{p_1, \ldots, p_m\}$ and r_i has a rank lower than r_j if $H_p(r_i) < H_p(r_j)$. Given a partition H_p, we may consider that the universe of pairs is reduced to pairs of requirements (r_i, r_j) such as $H_p(r_i) \neq H_p(r_j)$ (that is, we consider a representative requirement for each partition). We denote with U_p the resulting set of pairs, and with $|U_p|$ its cardinality. Following these definitions, we may introduce a measure of total disagreement between two order rankings, A_p and B_p, defined on the partition P, as

$$TDP(A_p, B_p) = \sum_{U_p} d_{A_p,B_p}(r_i, r_j), \tag{8}$$

where $d_{A_p,B_p}(r_i, r_j)$ defines the disagreement on a pair, as per (5), on the partition P. The normalized total disagreement is defined as

$$NDP(A_p, B_p) = \frac{1}{|U_p|} TDP(A_p, B_p), \tag{9}$$

where the normalization factor ensures that NDP will assume values between 0 and 1.

Disagreement measures can be applied to compare estimated and target ranks or between an attribute's ranking and a target rank. In the first case, the disagreement measures the accuracy of the outcome of the prioritization process. Lower values mean better ranking of requirements. In the latter case, the disagreement provides a measure of how much the ranks encoded by the attributes are close to the target rank. Low disagreements denote easy prioritization problems since the requirement representation already encodes useful information to approximate the target rank. When evaluating the effort versus accuracy tradeoff property, we measure the disagreement between estimated and target rankings, while when evaluating the domain adaptivity property we use disagreement measures between attribute rankings and the target ranking.

4 Effort versus Accuracy Tradeoff

4.1 Experiment Definition

This first experiment aims at investigating the tradeoff between pairwise elicitation effort and ranking accuracy, following the approach summarized in Table 1, column Experiment 1. The variables evaluated in the experiment are the accuracy of the final rank obtained with a specific method, measured as disagreement with respect to the target rank, and the elicitation effort, measured as the number of elicited pairwise comparisons.

Trends of these two variables while the number of requirements increases are also investigated. This analysis is performed with both the *CBRank* and *AHP* methods. Since *AHP* is known to be effective only for small sets of requirements [35], we are interested in assessing whether the advantage of using *CBRank* increases more than linearly as the set of requirements becomes larger. In addition, the comparison is extended to include a state-of-the-art implementation of *AHP* which adopts some heuristics to reduce the elicitation effort.

4.2 Experiment Execution

The investigation is performed on artificially generated sets of requirements, which are obtained by applying the following procedure. A given number of requirements are defined with unique identifiers. Each requirement is described by a set of attributes, which take integer values in the range between 0 and the cardinality of the requirement set. Attribute values are assigned in such a way that each attribute induces a total rank over the set of requirements. A total rank over the set of requirements is generated as reference target rank, namely, K, which may partially overlap with one of the ranks encoded by the attributes. This procedure allows for parametric generation of data set with an increasing number of requirements.

Following the steps of the *CBRank* prioritization process, as depicted in Fig. 1, instances of this process are simulated by applying the following procedures.

The first step of the prioritization process, which concerns pair sampling, at the beginning aims to provide a coverage of all requirements. The initial set of pairs is defined in such a way that it has cardinality $n/2$ and that each requirement $r_i \in Req$ is a member of at least one pair in the initial set.

The second step, namely, *Preference elicitation*, is the only step that requires human input. We simulate preference elicitation from decision makers as follows: First, a subset of pairs (r_i, r_j) is selected from the Cartesian product $Req \times Req$ with the restriction that $i \neq j$ and $(r_i, r_j) \neq (r_j, r_i)$. The relative preference function $\Phi(r_i, r_j)$ value is retrieved by directly sampling the given target ranking function K. For simplicity we restrict the simulation to a monotonic behavior, that is, we assume that the simulated 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, $H(r)$, fully defined over the set Req resulting in an approximated priority rank.

For the comparative evaluations we also run an *AHP*-based simulated prioritization process. In this case, the candidate pairs of the set of requirements Req are generated by exploiting a spanning tree (composed by $n - 1$ pairs), as described in [22]. The preference elicitation step is simulated following the same approach used in the case of *CBRank* and the computation of the rankings is performed through a procedure that implements the *AHP* algorithm described in [35].

The experimental comparison between *CBRank* and *AHP* has been conducted on sets of requirements with cardinality n equal to 25, 35, 50, 75, and 100, respectively. For each set of requirements, we run two prioritization processes, one based on *CBRank* and the other on *AHP*. In both cases, we computed the disagreement corresponding to a given number of elicited pair preferences.

A second set of measurements focuses on the application of the *local stopping rule*, which was proposed in [18] as a means of determining when the ranking estimate computed by *AHP* has reached an acceptable error threshold and new pairwise comparisons are no longer needed.

Adopting the notation introduced in the previous sections, the *local stopping rule* can be expressed as follows.

TABLE 2
Disagreement for 25, 35, 50, 75, 100 Requirements, with 100 Elicited Pairs, Computed as Average on 10 Runs in AHP and CBRank

n	25	35	50	75	100
TDA_{AHP}	31	71	208	693	1584
TDA_{CBRank}	22	54	122	416	990
Difference	9	17	86	277	594

Given the function K_{AHP} that represents the correct ranking and the function $H_{AHP}(r)_\sigma$ that is the ranking computed by the *AHP* algorithm at a certain stage σ of the pairwise elicitation step, the *local stopping rule* can be represented by the following expression:

$$|H_{AHP}(r_i)_{\sigma-1} - H_{AHP}(r_i)_\sigma| < a \quad \forall r_i \in Req, \qquad (10)$$

where a is a positive real number. Essentially, the rule requires that in two subsequent stages $\sigma - 1$ and σ the acquisition of new knowledge in terms of pairwise comparisons is not going to significantly modify the value of the ranking of r_i.

A critical problem when using this technique concerns how to a priori choose 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.

Input data for this second type of experiment are: a number of requirements n, a value for K, and a value for a. The output is a measure of the disagreement computed for the number of elicited pairs at which the process stops.

4.3 Results

Table 2 reports on the disagreement measures collected from 10 executions the prioritization process over the same dataset, with the *AHP* and the *CBRank* methods, respectively, while varying the number of requirements, namely, $n = 25, 35, 50, 75, 100$, and keeping the number of elicited pairs fixed to 100. The second and third rows correspond to the average over 10 runs of the disagreement computed with *AHP* and *CBRank*, respectively, and the fourth row reports the difference between the average disagreement computed with *AHP* and *CBRank*. We computed the p-value via Mann-Whitney-Wilcoxon test as an indicator of the statistical significance of the difference of the means obtained by the two methods. For all the measures the p-value is < 0.05, so it is possible to confirm that the difference between the two methods is statistically significant.

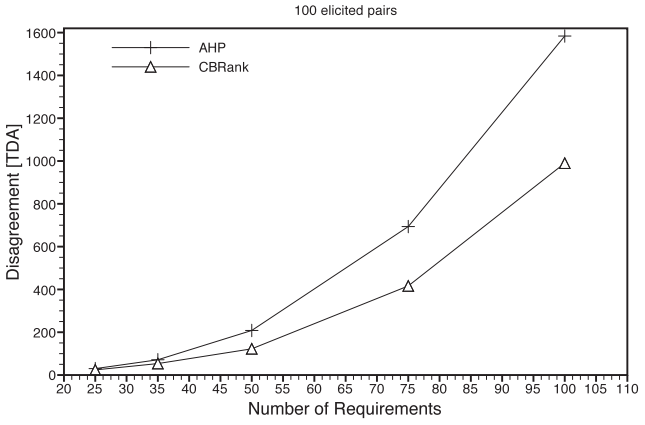

Fig. 3. The plot of disagreement (y-axis) for 25, 35, 50, 75, 100 requirements (x-axis) in *AHP* and the *CBRank* frameworks for 100 elicited pairs.

The series of plots, Figs. 2, 3, and 4, aims at comparing the two methods along the prioritization problem dimensions (i.e., number of requirements, elicited pairs, accuracy), considered two by two. Fig. 2 reports on the x-axis the number of elicited requirement pairs as a measure of the elicitation effort. The y-axis corresponds to the ranking accuracy measured as TDA, computed via (6). The plotted values have been obtained as the average of the disagreement values measured on 10 runs on the same data set of 100 requirements.

Fig. 3 shows the average disagreement (y-axis) for 25, 35, 50, 75, 100 requirements (x-axis) for *AHP* and the *CBRank* for a fixed elicitation effort, namely, for 100 elicited pairs.

Fig. 4 plots the average number of pairs to be elicited (y-axis) to limit the disagreement of the resulting ranks to 15 percent of the total number of pairs of requirements for 25, 35, 50, 75, 100 requirements (x-axis), computed with the *AHP* and *CBRank* methods.

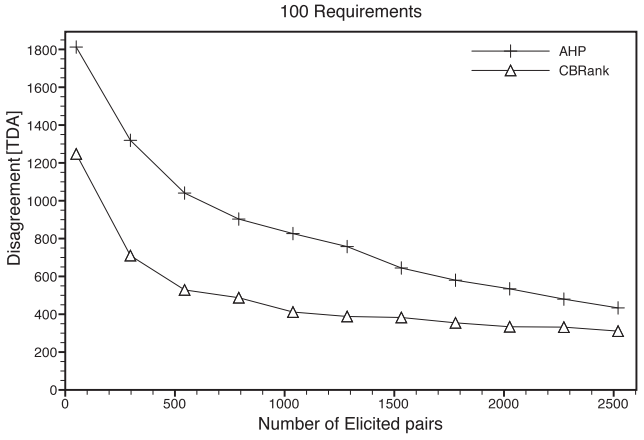

Fig. 2. The plot of disagreement (y-axis) for 100 requirements in *AHP* and the *CBRank* frameworks for an increasing number of elicited pairs (x-axis).

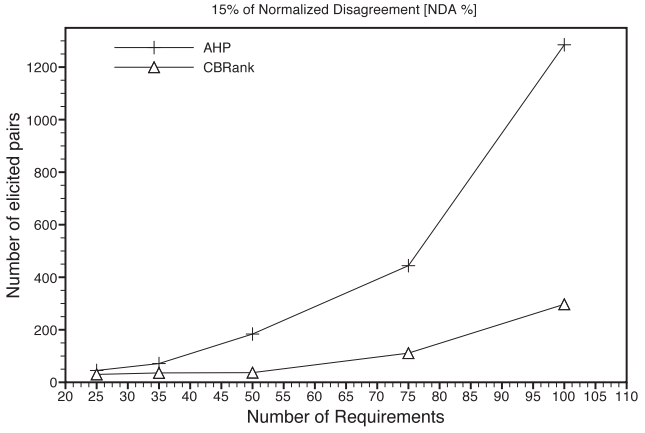

Fig. 4. The plot of the number of elicited pairs (y-axis) to have 15 percent of disagreement for 25, 35, 50, 75, 100 requirements (x-axis) in *AHP* and the *CBRank* frameworks.

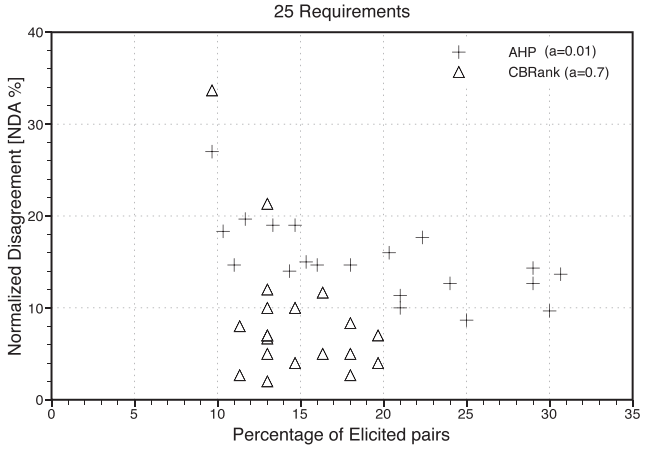

Fig. 5. The result of the application of the local stopping rule with *CBRank* and *AHP* for prioritizing the same set of 25 requirements.

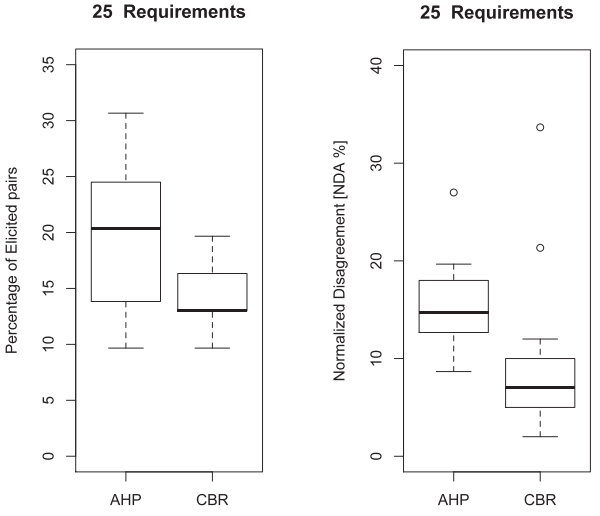

Fig. 6. Variance in the results of the application of the stopping rule in terms of Percentage of Elicited pairs (left) and Normalized Disagreement (NDA%).

Fig. 5 reports some results of the application of the local stopping rule with *CBRank* and *AHP*. In particular the diagram represents the results (computed via (7)) over 20 *AHP* runs, on the same set of 25 requirements and for $a = 0.01$ (cross label). Every point in the diagram represents the final value of one of these 20 runs. The results obtained running *CBRank* with the stopping rule, on the same set of requirements, with $a = 0.7$, are depicted in the same plot (triangle label). At these two slightly different values for the parameter a, the *AHP* and *CBRank* prioritization processes exploit similar percentages of the elicited pairs. Fig. 6 shows the variance on the results taken when applying the stopping rules for *AHP* and *CBRank* on the two dimensions of Percentage of Elicited pairs and Normalized Disagreement (NDA%).

4.4 Discussion

The first set of measurements allows a deep analysis of the tradeoff between the elicitation effort and the ranking accuracy with the two methods, and provides empirical evidence that *CBRank* uniformly outperforms *AHP*.

The results show that *CBRank* is more effective than *AHP*, especially for lower numbers of elicited pairs, since the difference between the *disagreement* measured with the two methods decreases with the increase of the elicited pairs, as shown in the plot depicted in Fig. 2.

Looking at the behavior of the two methods for low elicitation effort in greater detail (the most interesting case for practical purposes), we can see that the improvement of *CBRank* with respect to *AHP* increases when considering larger sets of requirements. In fact, considering in particular, the cases of 50, 75, 100 requirements when 100 pairs are elicited (Fig. 3 and Table 2) this trend in the difference between *CBRank* and *AHP* is evident. For instance, the difference between the *disagreement* measured with *CBRank* and with *AHP* in the case of 100 requirements is around 594 pairs and is clearly larger than the difference for 25 requirements, which is nine pairs.

This observation is particularly remarkable since, in practice, only a small portion of pairs can be manually elicited. For example, 100 pairwise analysis represents 2 percent of the total pairs in the case of 100 requirements, while 10 percent pairwise analysis on 100 requirements requires performing 495 comparisons.

The difference between the two methods is also evident when considering the objective of having a fixed percentage of disagreement, as shown in Fig. 4. Also, in this case the difference between the number of pairs to be elicited in *AHP* and *CBRank* for a disagreement of 15 percent increases, while the number of requirements grows.

During the experiments, the effectiveness of the spanning tree initialization strategy used in *AHP* has also been tested with *CBRank*. The results show that the adoption of this strategy does not produce a faster convergence of the *RankBoost* algorithm.

A better performance of *CBRank* with respect to *AHP* is also found when analyzing the results obtained using the *local stopping rule*, namely, better accuracy and lower effort, as well as lower variance on the respective measurements. The first result could be expected. In fact, applying the same policy to reduce the elicitation effort to both *CBRank* and *AHP* for a class of problems for which *CBRank* outperforms *AHP* will not change the relative trend of the two methods.

Concerning the variance of elicitation effort measurements, the following observations are worth being added. The behavior of the *local stopping rule* as a technique to reduce the elicitation effort is strongly context sensitive. The final outcome is affected by high variance of the numbers of pairs that have to be elicited. For example, in our experiment with 25 requirements depicted in Fig. 5, the percentage of elicitation pairs for *AHP* spans from 10 to 30 percent for 20 runs on the same problem (as also shown in Fig. 6 on the left). For *CBRank*, the effort variance is half with respect to *AHP*: In fact, for *CBRank* the effort is in the range between 10 and 20 percent of pairs and the median is around 13 percent, while *AHP* has a median around 21 percent. It is important to note that for both methods the results show a high variance with respect to the disagreement obtained when the process is stopped by the rule. This is plotted in Fig. 6 on the right. The behavior of *CBRank* seems to reveal a greater benefit from the *local stopping rule*. *CBRank* not only halves the variance of the

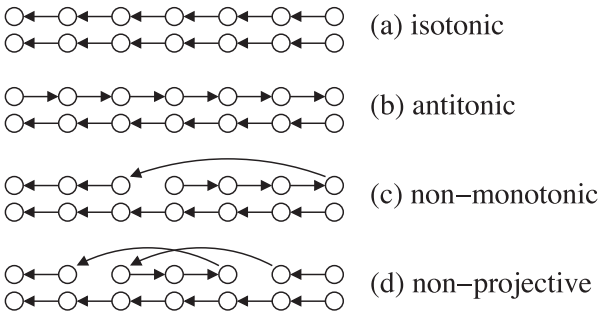

Fig. 7. Knowledge utility. A graphical representation of the four main relationships between two orderings of a given set of elements.

elicitation effort but it lowers the upper bound of the number of pairs that have to be elicited.

5 The Adaptivity Property

5.1 Experiment Definition

We are interested in giving experimental evidence of the capability of the method to exploit available domain knowledge encoded as ranking attributes.

For this purpose we designed an experiment, summarized in the second column of Table 1, Experiment 2, in which we evaluate the adaptive property of *CBRank*, also in comparison with *AHP*. The approach in this experiment is analogous to the one described in Section 4, except for the sets generated for the input to the prioritization process. For the same set of requirements and target ranking, we randomly choose different sets of ranking attributes, thus defining different prioritization problem instances.

The hypothesis is that when the representation of a specific set of requirements is encoding useful information with respect to the prioritization process, the performance of *CBRank* methods should improve, both by reducing the required elicitation effort and by increasing the rank accuracy.

To characterize the variability of the instances of a requirements prioritization problem, we propose a notion of utility that allows us to discriminate between useful and useless knowledge. Useful knowledge will be represented and encoded by those ranking attributes that improve the prioritization process. Knowledge utility can be defined in terms of the relationships that hold between the target ranking and the order relations encoded by the ranking attributes. It is possible to recognize four main kinds of relationships that can occur between two order relations: isotonic, antitonic, nonmonotonic, and nonprojective. They are depicted in Fig. 7.

For simplicity we focus our attention on the notion of direct order relationships on a given ordering, which is a pairwise precedence relation $r_i \prec r_j$ (among two requirements r_i and $r_j \in Req$) such that for each $r_k \in Req$ it never happens that $r_i \prec r_k \prec r_j$ where $i \neq j \neq k$; this relationship is depicted as a simple arrow pointing from r_j to r_i in Fig. 7. Fig. 7a shows the isotonic relationship. Both orderings sort the requirements with respect to the same priority criteria. An antitonic relationship holds when one direct order relationship is the reverse of the other. A nonmonotonic relationship applies when the ordering can be decomposed into smaller portions such that for each of them an isotonic or antitonic relationship holds (see Fig. 7c). When such a decomposition cannot be applied, we refer to the relationship as nonprojective (see Fig. 7d). These four categories well represent the notion of utility for a piece of knowledge encoded as a ranking attribute. Ranking attributes that hold an isotonic relationship with respect to the target ranking better support the learning process. Therefore, isotonic ranking attributes can be considered as useful knowledge. On the other hand, nonprojective ranking attributes do not enable an effective learning process; remember that the learning algorithm exploits a linear composition of ranking attributes. Notice that the complexity of the *CBRank* prioritization process is not dependent only on the choice of a specific target ranking function K but also on the above discussed relationship that holds between K and the set of ranking attributes F.

TABLE 3
Disagreement between the Attribute-Rankings and Target Ranking for 25, 35, and 50 Requirements

n	25	35	50
$NDA\%_{isotone}$	0%	0%	0%
$NDA\%_{non-projective}$	20%	21%	21%

5.2 Experiment Execution

On the basis of the above definitions, we can generate sets of prioritization problems with different degrees of difficulty.

After 10 runs on the same dataset we calculate the average of the disagreement measure with respect to the percentage of elicited pairs. We compute the disagreement until 50 percent of the total number of pairs has been elicited. After this percentage, the distance between the disagreement curves obtained with the different categories of knowledge (from isotone to nonprojective) becomes lower and so less relevant for the purpose of our comparative analysis.

The experiments with *CBRank* have been repeated for different classes of ranking attributes. The same set of experiments has been repeated, varying the cardinality of the requirements set from 10, 25, 35, 50, 75 to 100 requirements. In Table 3, the disagreement measures (reported in percentage with respect to the number of possible pairs) between the ranking used in the experiment and the target ranking K are given for the isotone and nonprojective cases.

The same simulated experiments have also been performed using the *AHP* method.

5.3 Results

In Fig. 8, we show the boxplots of the disagreement measured for two classes of domain knowledge (nonprojective, left side, and isotone, right side), through (7), for a set of 25 requirements. The boxplots are computed on 10 runs of the same problem instance. The variance of the disagreement measurements (the distance between the first and the third quartile) decreases when the percentage of the elicited pairs increases in both plots. Also, the median of the measurements decreases with the increase of the percentage of elicited pairs. Moreover, the decreasing trend in the median and the variance of the disagreement measurements are faster for the class of nonprojective

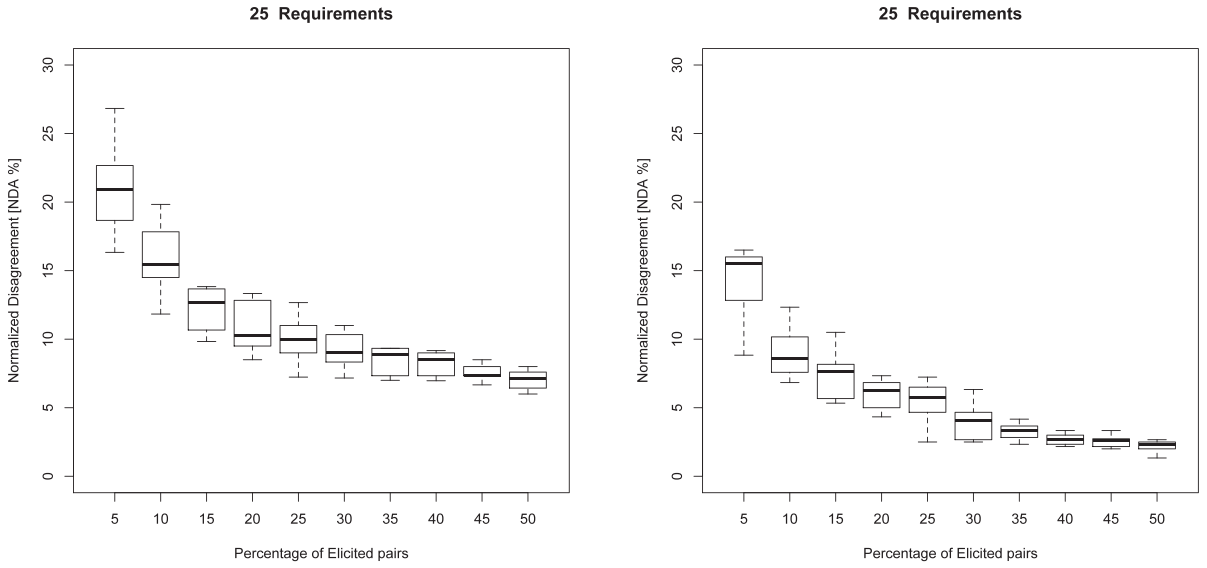

Fig. 8. Experimental results: Boxplots describing the prioritization disagreement variance in the case of nonprojective (left) and isotone knowledge (right) for 25 requirements.

knowledge (the worst case in domain knowledge exploitable for deriving the target ranking) than for the case of the isotone knowledge (the best case).

The plot in Fig. 9 shows the average of the disagreement measured through (6) in the case of 25, 35, 50, 75, 100 requirements, when 100 pairs are elicited from the decision maker with *CBRank* and with *AHP*, which have been run on the same datasets. Both worst (nonprojective) to best (isotone) cases for the *CBRank* method are depicted.[4]

5.4 Discussion

The disagreement measures in Fig. 8 confirm the intuition that when an isotonic relationship holds between ranking attributes and the target ranking, the *CBRank* learning process can be very effective, resulting in a good tradeoff between effort and accuracy. Moreover, the empirical analysis shows that knowledge utility and learning complexity are closely related. In fact, more pairs have to be elicited to reach the same level of ranking accuracy when nonprojective ranking attributes are exploited. For instance, in Fig. 8 we can see that in order to achieve a ranking with a median disagreement around 7 percent, we need to elicit about 15 percent of pairs in the case where we are exploiting isotone ranking attributes, and about 45 percent of pairs when we use nonprojective ranking attributes.

Focusing on Fig. 9, it is possible to recognize that in the case of 100 requirements for 100 elicited pairs (so exposing the decision maker to an acceptable decision effort), both when useful knowledge is available or not available, *CBRank* behaves better than *AHP*, resulting in a lower disagreement.

6 A Case Study

6.1 Experiment Definition

A further empirical study to assess the *domain adaptivity* property of *CBRank* is based on a case study performed on a medium-sized software application project (referred to as Experiment 3 in Table 1).

The main purpose of this case study is to find an answer to the following question: In a real situation, that is, when stakeholders of a third-party project use *CBRank*, is it possible to observe a positive effect of the domain adaptivity property on the requirements prioritization process as suggested by the simulated experiments illustrated previously?

The outline of the empirical study is the following. We consider a set of requirements of a real software project. Given the same set of requirements, we execute two prioritization processes with different target criteria, that is, we consider two different prioritization problems. Estimates for the associated target ranks can be derived at the end of the project, taking advantage of information available upon project conclusion. Based on these premises, we can compute two rank disagreement measures, the first between the target and estimated rank and the second between the target and the ranking attributes, for each

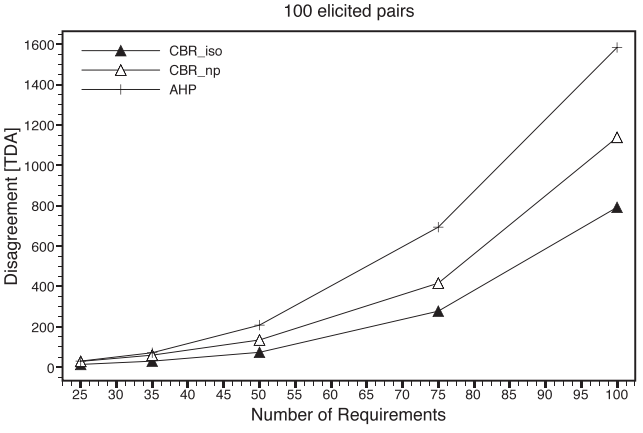

Fig. 9. The plot of disagreement (y-axis), for 25, 35, 50, 75, 100 requirements in *AHP* and the *CBRank* frameworks for 100 elicited pairs (x-axis) considering isotone and nonprojective ranking functions.

4. This distinction cannot be made with *AHP* that cannot benefit from domain knowledge of different levels of knowledge utility.

prioritization problem. As mentioned in the previous section, we may consider the disagreement between the target rank and the ranking attributes as a measure of the complexity of the prioritization process. A further question is whether there is a correlation for the two prioritization problems between the disagreement of estimated ranks and that computed on ranking attributes.

The case study has been provided by ProVotE [39], [40], a 4-year project in the area of e-Democracy whose objective was to introduce electronic elections (e-Voting) in Trentino, a region in Northern Italy. The project aims at eliminating risks of digital divide and at providing technological solutions to support the voting process from the voting event to the publication of the results.

Due to the critical nature of the application, a huge effort is devoted to the production of a highly secure, "bug free" system. This motivated the need for ranking the e-Voting system requirements along corresponding criteria, referred to in the following as "secure," when the evaluation is driven by security issues and "bug" when the evaluation is driven by an estimate of the possible debugging effort required at the implementation stage. The final objective is to detect system requirements that affect system dependability as early as possible, i.e., those relevant with respect to security as well as implementation effort concerns, and to plan for their development earlier with respect to project milestones.

The requirements document of the ProVotE project includes 104 requirements. A set of 32 requirements describes the general features, while the others are divided into six functional areas related to the various kinds of activities that have to be performed during a voting session.

The functional areas are: *Management of the polling place*, which concerns procedures that are usually performed at the polling station, such as the installation of the voting machines (six requirements); *Voting station set-up*, concerning the set-up phases of the voting place and booths, such as the possibility for the president of the voting place to login and initialize the voting procedures (15 requirements); *Voting operations—management*, related to the management of voting operations, such as the continuous monitoring of the voting machines (seven requirements); *Definition of the voting procedures*, concerning the definition of different characteristics for the voting procedures, such as the need to avoid multiple votes by a single person or the need to adhere to the national voting rules (36 requirements); *Ballot counting*, related to the procedures to be executed after the voting phase, such as ballot counting, with eight requirements. Some of the system features are also modeled as a set of Use-Cases.

In our experiment, we focused on a restricted set of 46 requirements, including three requirements from the *Voting station set-up* area (those which are more specific to the e-voting application domain) and all the requirements from the *Voting operations—management* area (7), and from the *Definition of the voting procedures* area (36). Notice that there are no dependencies between these requirements; moreover, information on their implementation in a first release was available.

Each requirement is described by the basic attributes recalled in Table 4.

TABLE 4
Basic Attributes of the ProVotE Project Requirements

Title	*A short description of the requirement*
Description	*The textual description of the requirement*
Importance	*The importance of the requirement, as perceived by the project managers, expressed as "Low", "Medium", "High"; if the requirement is intended to be implemented in the second phase of the project, the attribute is set "Phase 2"*
Notes	*Notes about the requirement*

Concerning the selection of the ranking attributes, the following knowledge is available from project documents, stakeholder expertise, and project management recorded information:

1. *"Importance of a requirement"*: As specified in the requirements document.
2. *"Importance of the functional areas"*: Upon advice of the ProVotE Project Manager, the most critical and important area in the project is the *Voting operations—management* area (VO area), so a higher weight is given to requirements belonging to this area.
3. *"Feature weight from Use-Cases"*: Use-Cases can aggregate a different number of system features. We associate weights to system features depending on the fact they appear in a more or less "feature-rich" Use-Case.
4. *"Number of updates to requirements"*: We observed that if a requirement has been modified several times, it is likely to be a critical one. We extracted the total number of updates of each requirement from the Requirements Document.

All this knowledge is encoded as partial or total ranking functions.

The subjects of the empirical study are project stakeholders. They are six technicians involved in the design and implementation of the ProVotE system who have different skills and comprehension of the project. Two of them play coordination roles over the whole project, one of them is in charge of realizing the operating system of the e-Voting application, the remaining three are in charge of the implementation of the application.

6.2 Experiment Execution

The six project members were divided into two groups: the first including the two project coordinators, the second the four project developers. Members of the second group performed two different prioritization processes, individually. In the first prioritization, we asked the developers to rank the requirements according to their expectation about bugs that could have been introduced during the implementation stage. That is, on each proposed requirement pair the developers specified which one among the two they considered more at risk of a buggy implementation. The target rank for this criterion was derived from the log of the bug tracking systems, in that case Bugzilla.

The second prioritization process aimed at ranking requirements according to the "secure" factor. Differently from the previous case, the four developers were asked to elicit pairwise comparisons about which requirement was

TABLE 5
Average of the Partition Disagreement for: Ranking Attributes (as Labeled in Parentheses) versus Target Ranking (Column 2); Estimated versus Target Ranking (Column 3)

Target	*Disagreement [NDP%]* **Ranking Attributes**	*Disagreement [NDP%]* **Estimated**
secure	(VO area) 16%	12%
bug	(use cases) 24%	41%

to be considered more critical with respect to security concerns. The target rank for the "secure" factor was obtained by interviewing the two project coordinators.

These prioritization processes were carried out using SCORE,[5] a web-based tool which implements the three-step prioritization process of *CBRank*, also depicted in Fig. 1. The SCORE tool provides users a GUI to input their pairwise preferences. The requirements, and the related attributes, as exemplified in Table 4, were uploaded in advance to the system. Subjects were briefly introduced to the use of the SCORE tool before starting the prioritization process.

In order to limit subjects' effort, we stopped preference elicitation when 14 percent of the total number of pairs for the 46 requirements was acquired.

6.3 Results

The rankings obtained at the end of the prioritization processes were divided into three categories, each composed of about 15 requirements. The first category includes the 15 higher rank requirements in the output, the second category the requirements ranked from the 16th position to the 30th position, and so on. The average disagreement measures are the average of the partition disagreement NDP, computed with the formula in (9), and reported as percentage with respect to the number of possible pairs in the partition.

Table 5 shows the results related to the use of the ranking attributes in the process. In column "Ranking Attributes" we report the name of the ranking attribute that produces the closest ranking (lowest disagreement) with respect to the target ranking relative to the "secure" criterion (first row) and to the "bug" criterion (second row), respectively. The percentage in this second column represents the disagreement between the requirements ordered according to this ranking attribute and the target ranking.

The third column, "Estimated," reports the disagreement between the ranking resulting from the online process (running with the ranking attribute reported at the left) and the target ranking ("secure," first row, "bug," second row). The results show that the closer the ranking attribute is to the target, the lower the disagreement between the estimated and the target rankings is. In fact, for the prioritization along the "secure" criterion, *CBRank* obtained only a 12 percent disagreement, exploiting knowledge on the weights of requirements belonging to the *Voting operations—management* functional area. The disagreement of the requirements along this ranking attribute with respect to the target ranking amounts to 16 percent. On the contrary, the best ranking attribute with respect to the "bug" criterion (the one related to the Use-cases) had a disagreement of 24 percent with respect to the target ranking, causing a worse performance of the prioritization method.

6.4 Discussion

The possibility of easily encoding available domain knowledge does not necessarily improve the performance of the prioritization process.

The improvement (i.e., lowering the disagreement with respect to the target ranking) depends on how informative the knowledge encoded as ranking attributes is with respect to the target ranking. Table 3 suggests that when the percentage disagreement between an attribute-based ranking and the target ranking is greater than 20 percent, the domain knowledge encoded in that attribute is no more effective toward increasing the accuracy of the computed ranking for a given effort (i.e., it falls in the nonprojective category). Results from the case study show that the attribute *VO area* is informative within a certain degree with respect to the *secure* criterion (in fact, the disagreement between the attribute and the target ranking amount to 16 percent, measured as NDP percent). And in this case, *CBRank* achieves a better estimate of the target ranking, given the same amount of elicited pairwise preference, in fact, the $NDP\%$ of estimated versus target ranking amounts to 12 percent. The functional area *VO area* is in fact the most important among the six functional requirements groups, according to the Project Manager, and this fact in our opinion reflects a correlation with project stakeholders' concerns about security aspects. Vice versa, the *use case* attribute, which characterizes the feature-richness of functional requirements represented in use cases, seems not relevant to identifying requirements that may lead to a more *bug*-prone implementation. In fact, while the attribute versus target ranking disagreement amounts to 24 percent, the disagreement between estimated and target ranking becomes even worse (41 percent).

7 Related Work

A rich literature on requirements prioritization is available. It includes studies that analyze the requirements prioritization role in software development processes, proposals of a variety of approaches to perform requirements prioritization, and an increasing set of empirical studies devoted to comparisons giving account of advantages and drawbacks (see, for instance, [10], [29], [31], [36]).

Focusing on requirements prioritization approaches, we distinguish basic ranking techniques, which usually allow prioritization of a set of candidates along a single evaluation criterion, from requirements prioritization methods, which integrate ranking techniques inside a requirements engineering process and aim at taking different aspects (criteria) into account, which may correspond to the different goals of project's stakeholders.[6]

Rank elicitation is performed by relevant stakeholders in a project (e.g., customers, users, system architects) and may

5. Information about the SCORE tool is available at http://se.fbk.eu/en/tools.

6. A similar terminology is proposed also in [11].

TABLE 6
Prioritization Methods

Req. Prior. Methods	Criteria	Rank. Technique
Quality Function Deployment - QFD [3]	customer goals and design requirements	numerical assignment
Win-Win [34]	various, e.g. business values development effort	*AHP*
AHP [35]	pairwise evaluation on alternatives, and hierarchies of criteria	*AHP*
Triage [13]	various, e.g. business goals; development resources	numerical assignment and basic ranking
Cost-Value [23]	development costs; value for the customer	*AHP*
Planguage [17]	stakeholders goals	basic ranking
Fairness Analysis [14]	stakeholders goals	Pareto optimal Search Based software engineering
Planning Game [9]	customer preferences; development time	numerical assignment and basic ranking
Wiegers method [41]	value for the customer; implementation cost and risk	numerical assignment and method rules
CBRank	domain adaptive	*RankBoost*, ML technique

be executed in different ways. A basic approach consists of assigning a rank to each requirement, in a candidate set, according to a specific criterion (e.g., value for the customer, development cost). The rank of a requirement can be expressed as its relative position with respect to the other requirements in the set, as in *Bubble sort* [24] or *Binary search* [20] procedures, or as an absolute measure of the evaluation criterion for the requirement, as in *Cumulative voting* [28]. Alternatively, pairwise comparison consists of assigning a preference value to pairs of candidate requirements. This type of rank elicitation is used in *AHP* and *CBRank*. A third, alternative way consists of assigning each requirement to one specific class among a certain number of predefined, different priority, classes, as, for instance, in *Numerical Assignment* [28], [41] and in *Top-10 requirements* [27].

The type of rank elicitation performed by a prioritization technique has an impact on the usability property of the technique itself. For instance, pairwise evaluation lowers the cognitive effort when the number of requirements to be evaluated is a few dozen, but the number of pairs to be elicited grows quadratically with the number of requirements, thus making it expensive (or even impractical) with a large set of requirements.

The ranking obtained with the different techniques ranges from requirements lists ordered according to an ordinal scale (Binary Search, Bubble Sort) to a requirement list ordered according to a rational scale (*AHP*, 100 Points) and to an ordinal scale (groups or classes), as in the Numerical Assignment and Top-10 techniques.

Concerning the scalability of these techniques when applied to sets of requirements of increasing size, it can be characterized in terms of the corresponding growth of the amount of information requested from the human decision maker by the techniques, namely, the elicitation effort. Given n requirements to be prioritized, computational complexity varies from a linear function in n, as in Numerical Assignment or Cumulative Voting, to a quadratic function, as in *AHP*. Heuristics have been exploited to overcome the problem of knowledge acquisition, trying to minimize it, also paying off in terms of precision.

Ranking techniques, as those recalled above, are used by more structured requirements prioritization methods which aim at handling multiple prioritization criteria. Table 6 collects well-known approaches and shows a characterization along the criteria used by each single method (second column), and the ranking techniques it exploits (third column). Basically, most of the methods aim at considering the customer point of view and development aspects such as the development cost or effort, or design constraints. Some of the methods combine different ranking techniques, as, for example, Planning Game, an approach which is used in eXtreme Programming [9]. Planning Game combines Numerical Assignment and basic ranking by first dividing the different requirements into priority groups and then ranking requirements within each group.

Moreover, methods usually adopt a specific approach or ad hoc rules to combine single-criterion rankings into the final rank. For instance, the *AHP* [35] method proposes a hierarchy-based approach to combine different and more abstract sets of criteria; in this case, criteria themselves are managed as alternatives and the decision maker can express the priority for the criteria in a pairwise comparison fashion. The output of the process is a set of weights to be used to combine the set of alternative criteria. Win-win [34] adopts negotiation techniques to derive a final requirements ranking from an agreement among subjective evaluations by different stakeholders, while Wieger's method [41] adopts a rule of thumb to compute the final ranking dividing the "value for the customer" of a given requirement by a penalty proportional to its implementation cost and

risks. In this line, [14] presents a multi-objective optimization approach to support investigation of the tradeoffs in various notions of fairness between multiple customers.

As to understanding to what extent it is possible to say that an allocation of requirements to customers is fair, when there are multiple customers, each with their own idea of what the next set of requirements should be, the framework proposes that each notion of fairness should form an objective in a multi-objective, Pareto optimal Search-Based Software Engineering (SBSE) setting.

It may be noticed that most of these methods require a priori definition of the ranking criteria, their measurement scale, and the rules to combine the single-criterion ranking into the final ranking, independently of the set of requirements to be prioritized. This limits the flexibility of those methods that are usually tuned to a specific application domain and development setting in order to be effective.

Comparative evaluations of some of these techniques and methods have been performed by means of experimental studies designed to analyze their usability and performance properties [24], [25]. Some of the properties, e.g., usability, were measured on the basis of a statistical analysis of subjective evaluations expressed by the users of a prioritization approach.

An experiment was performed [2] on five techniques, including *AHP*, Binary Search, Planning Game, the 100 Points Method. Fourteen evaluators, masters and PHD students, were asked to prioritize a set of 13 requirements. The Binary Search turned out to be the best technique for prioritizing requirements. The following properties were measured: the average time consumption, the "ease of use" of a technique as perceived by the subjects, and the accuracy of each technique's output with respect to the "ideal ranking" of the set of requirements, corresponding to the decision maker's opinion. According to the authors, both experiments are limited with respect to the number of requirements to be prioritized and of the evaluators sample. This study complements but apparently also partially contradicts a previous extensive study on six techniques [24]. The compared techniques included *AHP*, Hierarchy *AHP*, Bubble Sort, and Binary Search. They have been evaluated against technical properties, such as the ability of a technique to indicate consistency in the decision maker's judgment, a technique's ranking scale, time consumption (from preference elicitation to the computation of the final ranking), and against subjective measures, namely, ease of use, reliability of results, and fault tolerance with respect to judgmental errors. For this evaluation, the authors exploited a set of 13 quality requirements of a product like a small telephony system. Three evaluators were asked to prioritize requirements according to their importance, using the six techniques. The *AHP*-based techniques resulted as the most promising according to this study, although presenting limits in scalability. More generally, [11] reports an analysis of these studies, based on a systematic review [26], which points out that the results of these studies are often difficult to compare. Among the identified reasons for this problem is the fact that these studies often compare methods with techniques, or there may be differences in contexts, measured variables, and datasets used. Finally, in a recent study, Daneva and Herrmann [12] presented an analysis of hundreds of sources of works on requirements prioritization based on benefit and cos, which aimed at deriving a common, underlying conceptual model for these prioritization methods. The resulting conceptual model served as a classification framework for a subsequent survey review [19]. Not surprisingly, the description of the activities performed during requirements prioritization within this conceptual model fit the *ex-ante* approach since the prioritization criteria, benefit and cost, are assumed a priori. Worth mentioning is the follow-up of the survey review based on this classification framework [19], which is a set of open issues for a research agenda that include the need for a systematic management of the nonfunctional requirements in the prioritization processes and for a systematic management of requirements dependencies, as still pointed out by authors of some of the specific methods. In particular, this last point is judged as an important issue in the requirements community, but it is largely neglected in the current prioritization works.

8 Discussion

The positioning of *CBRank* with respect to requirements prioritization approaches, which has been elaborated in Section 7, will be complemented here with a discussion on benefit and limits, which is supported by the results of the empirical measurements.

CBRank can be used as a single-criterion ranking technique as well as a multicriteria prioritization method. Differently from *ex-ante* approaches that require associating a chosen target criterion to one or more predefined requirements attributes whose values are elicited from stakeholders, in *CBRank* requirements prioritization is obtained by acquiring the stakeholders priorities on pairs of requirements directly, rather than on their attributes.

Similarly to *AHP*, *CBRank* rests on pairwise priority elicitation. Nevertheless, the cognitive overload of the elicitation process is different in the two methods. In *AHP* the stakeholders have to provide a fine grain priority between two requirements, usually expressed in a range of 10 values. Since a clear and unambiguous semantics for these different values is not straightforward, the elicitation process is prone to a noisy acquisition of data from the evaluator. On the contrary, *CBRank*'s elicitation of the priority between candidate requirements rests on a Boolean scale. The information acquired from the stakeholder is not a quantitative measure of a priority index; nevertheless, a binary elicitation process enables a much robust data acquisition process. Less noisy data are an advantage for the computation of the prioritization rank, and at the same time Boolean elicitation lowers the evaluators' effort. In fact, choosing between two alternatives is less demanding than discriminating among 10 alternatives.

A further difference between *AHP* and *CBRank* concerns the core algorithm they rest on.

While *AHP* applies a multicriteria-based optimization algorithm that can rely, for instance, on the computation of the principal right eigenvector of the matrix of alternatives, *CBRank* aims to reduce a loss function which minimizes the

pairwise priority disagreement, exploiting machine learning techniques to minimize the set of pairs to be elicited.

Focusing on the comparison with other prioritization methods, two main aspects are of interest in this discussion: the fact that *CBRank* explicitly allows for the introduction of the concept of approximated rank as result of the prioritization process, and that it allows for exploiting available domain knowledge expressed as (partial) requirements ranking along different prioritization criteria.

The different computational methods exploited by *AHP* and *CBRank* have an impact on the elicitation process and on the quality of final rank. As pointed out before, *AHP* requires an exhaustive elicitation of pairwise relations; partial elicitation is allowed and it can be performed only when the missing values will not affect the final rank. *CBRank* introduces the notion of approximated rank. The quality of final rank might have different levels of accuracy, depending on the amount of elicitation effort considered acceptable from the stakeholders.

Moreover, *CBRank* exploits available knowledge on the requirements, making it highly adaptable to different domains also supporting the minimization of the pairs to be elicited from the stakeholder (in Table 6, this property is referred to as being *domain adaptive* with respect to the ranking criteria a method can be used for).

9 Conclusion and Future Work

In this paper, we provided a detailed account of the *CBRank* method for requirements prioritization.

The *CBRank* method follows the case-based paradigm for problem solving, according to which a solution to a new problem can be derived from (partial) examples of previous solutions to similar problems. In the context of requirements prioritization, these examples are elicited from project stakeholders as pairwise preferences on samples of the set of requirements to be prioritized, and used to compute an approximated ranking for the whole set.

The machine learning technique exploited by the method has been presented, both with the help of an intuitive example and by describing the *RankBoost* algorithm, which is implemented in the method. The prioritization process based on *CBRank* has been presented.

A discussion of the method performance, which is defined in terms of tradeoff between preference elicitation effort and ranking accuracy and of its domain adaptivity, has been given, with the support of a set of different experimental measurements and of a case study. The experimental measurements were taken by applying *CBRank* to different prioritization problems, varying the number of requirements, the number of elicited pairs, and the accuracy of the computed ranking. Indicators for the statistical significance of the measurements have been provided.

Finally, the *CBRank* method has been positioned with respect to state-of-the art approaches, with particular reference to the *AHP* method, which can also be considered an instance of the case-based problem solving paradigm. Differently from *AHP*, the *CBRank* method enables a prioritization process, even over 100 requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations from the acquired data.

Some assumptions have been considered in the described work, such as the monotonicity of the elicitation process and random selection as pair sampling policy in the *CBRank* prioritization process. Future work should address the nonmonotonic case and more sophisticated pair sampling policies, possibly contributing to improving the effectiveness of the method in more complex real settings.

Further characteristics of the *CBRank* method that are worth investigating are its ability to support coordination among different stakeholders through negotiation [8]. Moreover, potential advantages of integrating *CBRank* with other techniques such as planning game or *AHP* deserve further analysis.

Well-known open issues in the requirements prioritization problem such as handling requirements dependencies and "anytime" prioritization, which is updating requirements ranking when new requirements are added (or removed), are also worth being further investigated.

Acknowledgments

The authors thank the participants to the case study; Cinzia Bazzanella and Sara Ferrari who contributed to this research with their master's theses; Paolo Tonella, John Mylopoulos, and the anonymous reviewers for their valuable feedback. They gratefully acknowledge the support of the Editorial office of FBK.

References

[1] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Artificial Intelligence Comm.*, vol. 7, no. 1, pp. 39-59, 1994.

[2] V. Ahl, "An Experimental Comparison of Five Prioritization Methods—Investigating Ease of Use, Accuracy and Scalability," master's thesis, School of Eng., Blekinge Inst. of Technology, Aug. 2005.

[3] Y. Akao, *Quality Function Deployment: Integrating Customer Requirements into Product Design.* Productivity Press, 1990.

[4] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Supporting the Requirements Prioritization Process. A Machine Learning Approach," *Proc. 16th Int'l Conf. Software Eng. and Knowledge Eng.*, pp. 306-311, June 2004.

[5] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Exploiting Domain Knowledge in Requirements Prioritization," *Proc. 17th Int'l Conf. Software Eng. and Knowledge Eng.*, pp. 467-472, July 2005.

[6] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Facing Scalability Issues in Requirements Prioritization with Machine Learning Techniques," *Proc. 13th IEEE Int'l Conf. Requirements Eng.*, pp. 297-306, Sept. 2005.

[7] P. Avesani, S. Ferrari, and A. Susi, "Case-Based Ranking for Decision Support Systems," *Proc. Fifth Int'l Conf. Case-Based Reasoning: Research and Development*, pp. 35-49, 2003.

[8] P. Avesani, A. Susi, and D. Zanoni, "Collaborative Case-Based Preference Elicitation," *Proc. Int'l Conf. Innovations in Applied Artificial Intelligence*, pp. 752-761, 2005.

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

[10] P. Berander and A. Andrews, "Requirements Prioritization," *Eng. and Managing Software Requirements*, A. Aurum and C. Wohlin, eds., Springer, 2005.

[11] P. Berander, K.A. Khan, and L. Lehtola, "Towards a Research Framework on Requirements Prioritization," *Proc. Sixth Conf. Software Eng. Research and Practice in Sweden*, Oct. 2006.

[12] M. Daneva and A. Herrmann, "Requirements Prioritization Based on Benefit and Cost Prediction: A Method Classification Framework," *Proc. 34th Euromicro Conf. Software Eng. and Advanced Applications*, pp. 240-247, 2008.

[13] A. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing.* Dorset House, 2005.

[14] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, and Y. Zhang, "A Search Based Approach to Fairness Analysis in Requirement Assignments to Aid Negotiation, Mediation and Decision Making," *Requirements Eng.*, vol. 14, no. 4, pp. 231-245, 2009.
[15] Y. Freund, R.D. Iyer, R.E. Schapire, and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences," *Proc. 15th Int'l Conf. Machine Learning*, pp. 170-178, 1998.
[16] Y. Freund, R.D. Iyer, R.E. Schapire, and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences," *J. Machine Learning Research*, vol. 4, pp. 933-969, 2003.
[17] T. Gilb, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage.* Elsevier, 2005.
[18] P.T. Harker, "Incomplete Pairwise Comparisons in the Analitic Hierarchy Process," *Math. Modeling*, vol. 9, no. 11, pp. 837-848, 1987.
[19] A. Herrmann and M. Daneva, "Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research," *Proc. IEEE 16th Int'l Conf. Requirements Eng.*, pp. 125-134, 2008.
[20] F. Hivert, J.-C. Novelli, and J.-Y. Thibon, "The Algebra of Binary Search Trees," *Theoretical Computer Science*, vol. 339, no. 1, pp. 129-165, 2005.
[21] H. Peter, D. Olson, and T. Rodgers, "Multi-Criteria Preference Analysis for Systematic Requirements Negotiation," *Proc. 26th Int'l Computer Software and Applications Conf.*, pp. 887-892, Aug. 2002.
[22] J. Karlsson, S. Olsson, and K. Ryan, "Improved Practical Support for Large Scale Requirements Prioritizing," *J. Requirements Eng.*, vol. 2, pp. 51-67, 1997.
[23] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software*, vol. 14, no. 5, pp. 67-74, Sept./Oct. 1997.
[24] J. Karlsson, C. Wohlin, and B. Regnell, "An Evaluation of Methods for Prioritizing Software Requirements," *Information and Software Technology*, vol. 39, nos. 14/15, pp. 939-947, 1998.
[25] L. Karlsson, T. Thelin, B. Regnell, P. Berander, and C. Wohlin, "Pair-Wise Comparisons versus Planning Game Partitioning—Experiments on Requirements Prioritisation Techniques," *Empirical Software Eng.*, 2006.
[26] K.A. Khan, "Systematic Review of Requirements Prioritization—A Research Framework," master's thesis, Blekinge Inst. of Technology, 2006.
[27] S. Lauesen, *Software Requirements: Styles and Techniques.* Addison Wesley, 2002.
[28] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach.* Addison-Wesley Longman Inc., 2000.
[29] L. Lehtola, "Providing Value by Prioritizing Requirements Throughout Product Development: State of Practice and Suitability of Prioritization Methods," PhD thesis, HUT/Dept. of Computer Science, 2006.
[30] F. Moisiadis, "Prioritising Software Requirements," *Proc. Int'l Conf. Software Eng. Research and Practice*, June 2002.
[31] F. Moisiadis, "A Framework for Prioritizing Software Requirements," PhD thesis, Macquarie Univ., July 2003.
[32] A.N. The and G. Ruhe, "Requirements Negotiation under Incompleteness and Uncertainty," *Proc. 15th Int'l Conf. Software Eng. and Knowledge Eng.*, pp. 586-593, July 2003.
[33] A. Perini, F. Ricca, and A. Susi, "Tool-Supported Requirements Prioritization: Comparing the AHP and CBRank Methods," *Information and Software Technology*, vol. 51, no. 6, pp. 1021-1032, 2009.
[34] G. Ruhe, A. Eberlein, and D. Pfahl, "Quantitative Winwin: A New Method for Decision Support in Requirements Negotiation," *Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng.*, pp. 159-166, 2002.
[35] T.L. Saaty, *Fundamentals of the Analytic Hierarchy Process.* RWS Publications, 1994.
[36] J.I.A. Siddiqi and M.C. Shekaran, "Requirements Engineering: The Emerging Wisdom," *IEEE Software*, vol. 13, no. 2, pp. 15-19, Mar. 1996.
[37] S. Sivzattian and B. Nuseibeh, "Linking the Selection of Requirements to Market Value: A Portfolio-Based Approach," *Proc. Seventh Int'l Workshop Requirements Eng.: Foundation for Software Quality*, June 2001.
[38] P. Tonella, P. Avesani, and A. Susi, "Using the Case-Based Ranking Methodology for Test Case Prioritization," *Proc. 22nd IEEE Int'l Conf. Software Maintenance*, pp. 123-133, 2006.
[39] A. Villafiorita and G. Fasanelli, "Transitioning to E-Voting: The Provote Project and the Trentino's Experience," *Proc. Int'l Conf. Electronic Government*, 2006.
[40] K. Weldemariam, A. Villafiorita, and A. Mattioli, "Assessing Procedural Risks and Threats in E-Voting: Challenges and an Approach," *Proc. Conf. e-Voting and Identity*, 2007.
[41] K.E. Wiegers, *Software Requirements. Best Practices.* Microsoft Press, 1999.
[42] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, 2000.

Anna Perini received the doctoral degree in Physics from the University of Trento, Italy, in 1981 and joined IRST (FBK) in 1985. She is a research scientist at Fondazione Bruno Kessler (FBK), Trento, Italy. She has served on the program committees of many international workshops and conferences, among which are RE, SEAMS, AOSE, AAMAS; she was organizing chair of ECAI '06, IEEE RE '11, and program chair of STAIRS '06. Her research interests include requirements engineering, agent-oriented software engineering, and empirical software engineering. She is a member of the IEEE.

Angelo Susi received the Dr. degree in Computer Science Engineering from the University of Roma "La Sapienza," Italy, and joined IRST (FBK) in 2000. He is a researcher in the Software Engineering Unit of the Fondazione Bruno Kessler, Trento, Italy. He has served on the program committees of several international workshops and conferences, was organizing chair of IEEE RE '11, and general chair of SSBSE '12. His research interests include requirements engineering, goal-oriented software engineering, software testing, and machine learning. He is a member of the IEEE.

Paolo Avesani received the doctoral degree in Information Science from the University of Milan, Italy, in 1988 and joined the Automated Reasoning System division at IRST (FBK) in 1989. He is a researcher at Fondazione Bruno Kessler (FBK), in Trento, Italy, where he leads NILab, the joint Neuroinformatics Laboratory of FBK and the Center for Mind/Brain Sciences of the University of Trento. He has participated in many EU projects. His research interests include machine learning, decision-support systems, and distributed autonomous information systems. He is a member of the IEEE.