Information and Software Technology 55 (2013) 173–187

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Interactive requirements prioritization using a genetic algorithm

Paolo Tonella, Angelo Susi *, Francis Palma

Software Engineering Research Unit, Fondazione Bruno Kessler, Via Sommarive 18, I-38123 Trento-Povo, Italy

ARTICLE INFO

Article history:
Available online 17 July 2012

Keywords:
Requirements prioritization
Interactive genetic algorithms
Search based software engineering

ABSTRACT

Context: The order in which requirements are implemented affects the delivery of value to the end-user, but it also depends on technical constraints and resource availability. The outcome of requirements prioritization is a total ordering of requirements that best accommodates the various kinds of constraints and priorities. During requirements prioritization, some decisions on the relative importance of requirements or the feasibility of a given implementation order must necessarily resort to a human (e.g., the requirements analyst), possessing the involved knowledge.
Objective: In this paper, we propose an Interactive Genetic Algorithm (IGA) that includes incremental knowledge acquisition and combines it with the existing constraints, such as dependencies and priorities. We also assess the performance of the proposed algorithm.
Method: The validation of IGA was conducted on a real case study, by comparing the proposed algorithm with the state of the art, interactive prioritization technique Incomplete Analytic Hierarchy Process (IAHP).
Results: The proposed method outperforms IAHP in terms of effectiveness, efficiency and robustness to decision maker errors.
Conclusion: IGA produces a good approximation of the reference requirements ranking, requiring an acceptable manual effort and tolerating a reasonable human error rate.

1. Introduction

During the various iterations of software development, prioritization of the requirements to be implemented plays a key role. Different concerns affect the decision of which requirements to implement in the next release and in what order to implement them. These concerns include the availability of resources, the time constraints, the end-user's expectations and of course the technical constraints.

Requirements prioritization aims at finding an order relation on a given set of requirements. This process can be based on information available *a priori* (i.e., independently of the specific requirement instances) or *a posteriori* (i.e., depending on the specific features of the requirements at hand) as also discussed in [1]. In the former case, the preferences are formulated before the specification of the set of requirements via predefined models, for example by defining the requirement attributes which determine the final ranking and by specifying how the final ranking is computed from those attributes. In the *a posteriori* approaches, the ranking is formulated on the basis of the characteristics of the set of requirements under analysis. One of the most widely used methods in this category is pairwise comparison. While the requirements analyst might find it difficult to answer global questions about the priorities of the full set of requirements, expressing a preference between two alternatives is a cognitively more affordable way of capturing the analyst's knowledge, as, for example, pointed out in [2] and in [3].

In this paper, we propose an Interactive Genetic Algorithm (IGA) search-based technique for requirements prioritization. It falls in the category of the *a posteriori* methods and it takes advantage of pairwise preference elicitation, as an effective means to extract relevant knowledge from the user.[1] Such user knowledge is composed with the other constraints and ranking criteria represented in the requirement documents (e.g., in the form of requirement attributes). Our algorithm aims at producing an accurate requirements ordering, while keeping the decision-making effort associated with pairwise comparison affordable for the requirements analyst.

Among the prioritization techniques used in the available requirements prioritization methods [4,3,5–7], Analytical Hierarchy Process (AHP) [8] exploits pairwise comparison to extract user knowledge with respect to the ranking of the requirements. AHP defines the prioritization criteria through a priority assessment of

* Corresponding author. Tel.: +39 0461 314344; fax: +39 0461 302040.
E-mail addresses: tonella@fbk.eu (P. Tonella), susi@fbk.eu (A. Susi), fpalma@fbk.eu (F. Palma).

[1] To avoid confusion we reserve the term "user" to the user of the prioritized requirements, i.e., the requirements analyst or system architect, while the term "end-user" indicates the ultimate user of the system under development.

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.07.003

all the possible pairs of requirements. Hence, its cost is quadratic with the number of requirements. In general, available *a posteriori* prioritization methods, including AHP, suffer scalability problems. To overcome this limitation in AHP, Harker [9] proposed an incomplete version of this method, named Incomplete AHP (IAHP), that allows the limiting of the number of pairs requested to the decision maker, enabling the establishment of a trade-off between the decisional effort and the precision of the ranking.

The IGA requirements prioritization technique is a pairwise comparison method which exploits a genetic algorithm to reduce the number of pairs elicited from the requirements analyst. Only pairs that allow the disambiguation of equally ranked or differently ranked requirements are elicited. The disagreement between the requirements ordering encoded in an individual and the elicited pairs and the initial constraints on the relative ordering of requirements define the fitness function to be optimized (minimized). Elicitation and optimization are conducted at the same time, because they influence each other. In fact, the critical requirement pairs to be elicited are known only after the genetic algorithm has optimized the requirements ordering with respect to the available constraints. But the newly elicited pairs introduce new constraints to be satisfied. Hence, the fitness function used by IGA is constructed incrementally, as long as new pairwise comparisons, introducing new constraints, are made. As a consequence, convergence of the algorithm is not ensured and depends on the stability of the fitness function over time, during incremental knowledge acquisition.

In our previous work [10] we assessed the effectiveness of the IGA algorithm on a real case study, including a number of requirements which makes state of the art techniques based on exhaustive pairwise comparison impractical. The results indicated that IGA converges and improves the performance of the GA (without interaction) in a substantial way, while keeping the user effort (number of elicited pairs) acceptable.

This paper extends our previous work [10] with substantial novel material: (1) the IGA algorithm is now directly compared with IAHP, the state of the art reference algorithm for interactive requirement prioritization; (2) three out of four research questions have been completely reformulated and new experimental data have been collected to answer them; (3) the description of the algorithm has been improved and extended; (4) an appendix has been added, with a short description of the IAHP algorithm.

The paper is organized as follows. We first describe some relevant related works (Section 2), then we give an intuitive description of the IGA approach (Section 3) by means of a small example, followed by the formal presentation of the algorithm (Section 4). Then, we describe a set of experimental evaluations, conducted on a real set of requirements, about convergence, effectiveness and robustness of IGA (Section 5) comparing it to the performance of IAHP. Conclusions and future work are presented in Section 6. A primer on AHP and IAHP is provided in Appendix A.

2. Related work

Search based approaches are increasingly influencing different areas of Software Engineering spanning requirements to testing. In the area of software testing several papers describe the application of search techniques, such as Genetic Algorithms, for the automatic generation of test cases [11]. To the best of our knowledge, interactive genetic algorithms have not been applied to Software Engineering problems before, with the notable exception of the work by Quiroz et al. [12], who used a collaborative, interactive GA to evolve user interfaces in the XUL interface definition language, by combining some heuristic GUI design metrics with user input.

Focusing on the area of requirements, several works pointed out the importance of the problem of requirements prioritization proposing agendas and literature reviews on the field (see for example [13,14]). In the following, we first overview the literature on search based algorithms applied to requirements engineering problems. Then, we recall the main works describing the application of search based approaches, such as Genetic Algorithms or more generally Machine Learning, and pairwise comparison based algorithms, such as Analytic Hierarchy Process (AHP), to the problem of requirements prioritization.

2.1. Search based requirements engineering

Several search based approaches have been exploited for the solution of different kinds of requirements engineering problems. In [15] the Next Release Problem (NRP) is studied as an optimization problem. The work exploits three different heuristics, namely simulated annealing, genetic algorithms, and ant colony optimization (adapted to the NRP problem) to solve NRP in some case studies. The results of the paper show that the solutions discovered by the techniques contain a high percentage of requirements judged by the user as important for a particular release of the system.

Still in the context of NRP, Zhang and Harman [16] propose an archive-based multi-objective evolutionary algorithm, using the non-dominated sorting genetic algorithm (NSGA-II) to solve the problem of Requirements Interaction Management. In particular, this work investigates how the dependencies between requirements influence the automatic selection of requirements during the phase of release planning. They demonstrate that their framework is able to find the optimal balance in the solution space for release planning under different contexts (such as, different decision makers, value/cost relationships).

Among the methods exploiting genetic algorithms for requirements management, the EVOLVE method [17] supports continuous planning for incremental software development. In particular, this method allows for optimal allocation of requirements to increments and it is at the same time a means for assessing and optimizing the degree to which the ordering conflicts with stakeholder priorities, within technical precedence constraints, and a means for balancing required and available resources for all increments. The approach is based on an iterative optimization method making use of a genetic algorithm.

Zhang et al. [18] focused on the specific Multi-Objective Next Release Problem (MONRP) in Requirements Engineering and presented the results of an empirical study on the suitability of weighted and Pareto optimal genetic algorithms, together with the NSGA-II, providing evidence to support the claim that NSGA-II is well suited to the MONRP.

Both EVOLVE [17] and the work by Zhang et al. [18] overcome the problems of scalability of methods such as AHP, but they do not produce a total ordering of requirements. Rather, they group requirements for the planning of the next release.

The Next Release Problem is also the focus of [19]. Here an Ant Colony Optimization (ACO) approach to the NRP problem in the presence of dependent requirements is presented. The evaluation of the proposed approach is performed over 72 synthetic datasets and considered, besides ACO, Genetic Algorithms and Simulated Annealing. Results show the ability of the proposed ACO algorithm to generate more accurate solutions to the Software Release Planning problem when compared to the other two methods.

In [20] a multi-objective optimization approach to support investigation of the trade-offs in various aspects of fairness between multiple customers when a decision on the set of requirements to be implemented has to be taken. The approach is validated using two real-world data sets and also using data sets created specifically to stress test the approach. Moreover,

experiments are reported to determine the most suitable algorithm for this problem, comparing the results of the NSGA-II algorithms, and the Two-Archive evolutionary algorithm.

2.2. Requirements prioritization approaches

Several techniques used in the current prioritization approaches consist of assigning a rank to each requirement in a candidate set according to a specific criterion, such as value of the requirement for the customer or requirement 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* or *Binary search* procedures, or as an absolute measure of the evaluation criterion for the requirement, as in *Cumulative voting* [21].

Alternative techniques consist of assigning each requirement to one specific class among a set of predefined priority classes, as for instance in *Numerical Assignment* [21,22] and in *Top-10 requirements* [23].

Requirements prioritization methods can be split into two categories, depending on the order relation they eventually produced. The rank assigned to requirements by a prioritization method may define either a *total* ordering of the requirements or a *partial* ordering. Having a total ordering could simplify the allocation of requirements to the next release, but it may also overconstrain their implementation. Our work belongs to the first category, but we recognize that methods aiming at a partial ordering represent also an important stream of research in the area, which had some impact on the practice of software development (e.g., the planning game of extreme programming [24]).

Other approaches share a prioritization scheme that passes through the selection of one or more prioritization criteria including business goals and technical features, the acquisition of requirements priorities on the basis of criteria elicited from the stakeholders, and the integration of the acquired orderings into the final one [3,5–7,25].

Among the pairwise techniques, CBRank [1] adopts a preference elicitation process that combines sets of preferences elicited from human decision makers with sets of constraints which are automatically computed through machine learning techniques. It also exploits knowledge about (partial) rankings of the requirements that may be encoded in the description of the requirements themselves as requirement attributes (e.g., priorities or preferences). However, it cannot handle relations among requirements such as dependencies.

The Analytical Hierarchy Process (AHP) [8] can be considered one of the reference methods. It adopts a pairwise comparison strategy, allowing the definition of the prioritization criteria through a priority assessment of all the possible pairs of requirements, thus requiring the elicitation of a number of pairwise comparisons which is quadratic with respect to the number of requirements being prioritized; in fact, given n requirements the decision maker has to perform $n(n-1)/2$ pair-wise comparisons. This method becomes impractical as the number of requirements increases, as observed, for example, in [26] where the problem of ranking a set of 20 requirements using AHP (so requiring 190 pair-wise comparisons) caused a huge cognitive effort to the decision makers. The scalability problem in using this technique has been only partially addressed by the introduction of heuristic stopping rules.

Harker [9] proposed a method to overcome the AHP scalability issues. This method relies on the key idea of minimizing the number of pairwise comparisons elicited from the decision maker while maintaining a good trade-off between the precision of the final solution and the effort of the decision maker. This is done by calculating, at each iteration of the elicitation process, a prediction of the next most promising pair to be asked in order to find a stable and good approximation of the target ranking before eliciting all the $n(n-1)/2$ pairs. The missing pairwise comparison scores are estimated by means of a heuristic. The interested reader can find a summary description of the AHP and IAHP methods in Appendix A of this paper.

Our approach exploits IGA to minimize the amount of knowledge, in terms of pairwise evaluations, that has to be elicited from the user. This makes the approach scalable to requirements sets of realistic size. Similarly to CBRank and differently from AHP, IAHP and the GA used in other approaches, our algorithm exploits the initial constraints specified in terms of partial rankings (e.g., priorities) to reduce the number of elicited pairwise comparisons. However, differently from CBRank it takes advantage not only of rankings, but also of precedence constraints specified as precedence graphs (e.g., dependencies). This allows further reduction of the elicited pairs. Moreover, the proposed approach does not elicit all pairwise comparisons. It resorts to elicitation only for those pairs which are expected to affect the final ordering to a major extent.

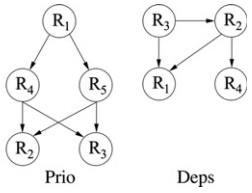
3. Approach

The prioritization approach we propose aims at minimizing the disagreement between a total order of prioritized requirements and the various constraints that are either encoded with the requirements or that are expressed iteratively by the user during the prioritization process. We use an interactive genetic algorithm to achieve such a minimization, taking advantage of interactive input from the user whenever the fitness function cannot be computed precisely based on the information available. Specifically, each individual in the population being evolved represents an alternative prioritization of the requirements. When individuals having a high fitness (i.e., a low disagreement with the constraints) cannot be distinguished, since their fitness function evaluates to a plateau, user input is requested interactively, so as to make the fitness function landscape better suited for further minimization. The prioritization process terminates when a low disagreement is reached, the time out is reached or the allocated elicitation budget has been consumed.

Let us consider the five requirements listed in Table 1. For each requirement we consider the priority expressed by the analyst who collected them and the dependencies the requirement may have with other requirements. For conciseness, in Table 1 we omit other important elements of a requirement (e.g., the textual description). Constraints and properties of requirements can be represented by means of a *precedence* graph. In the following, we always make the assumption that precedence graphs are actually DAGs (directed acyclic graphs). In fact, cyclic (sub-) graphs provide no useful information, since they do not constrain requirements ordering in any way. In a precedence graph, an edge between two requirements indicates that, according to the related constraint or property, the requirement associated with the source of the edge should be

Table 1
Requirements with priority and dependencies.

Req	Prio	Deps
R_1	High	R_2, R_3
R_2	Low	R_3
R_3	Low	
R_4	Medium	R_2
R_5	Medium	



implemented before the target requirement. Edges may be weighted, to actually quantify the strength or importance of such a precedence and an infinite weight is used for precedence relations that must necessarily hold in the final ordering of the requirements. Weights could be given to individual edges or, alternatively, to the whole precedence graph representing a set of constraints or properties. The changes of the algorithm necessary to accommodate variable weights are described in detail in the next section.

At the right of Table 1, we show the precedence graph induced by the *priority* (Prio) property of the requirements and the precedence graph induced by the *dependencies* (Deps) between requirements. Requirements with *High* priority should precede those with *Medium* priority. Hence the edges (R_1, R_4) and (R_1, R_5) in the graph Prio. Similarly, the precedence between *Medium* and *Low* priority requirements induces the four edges at the bottom of the precedence graph Prio. Requirement R_1 depends on $R_2,\ R_3$, hence the implementation of $R_2,\ R_3$ should precede that of R_1, which gives rise to the edges (R_2, R_1) and (R_3, R_1) in Deps.

The interactive genetic algorithm we use for requirements prioritization evolves a population of individuals, each representing a candidate prioritization. Table 2 shows six individuals (i.e., six prioritizations). An individual is a permutation of the sequence of all requirements to be prioritized. In order to evolve this population of individuals, their fitness is first evaluated. We measure the fitness of an individual as the *disagreement* between the total order encoded by the individual and the partial orders encoded in the precedence graphs constructed from the requirement documents. Further precedence relationships are obtained from the user during the prioritization process. Such relations are also considered in the computation of the disagreement. They constitute the *elicited precedence graph*, which is initially empty. In our running example, it would be the third precedence graph, to be added to Prio and Deps.

Initially no elicited precedence graph is available. Hence, disagreement is computed only with respect to the precedence graphs obtained directly from the requirements documents. The disagreement between a prioritized list of requirements and a precedence graph is the cardinality of the set of pairs of requirements that are ordered differently in the prioritized list and in the precedence graph. With reference to individual Pr_1 in Table 2, we can notice that R_3 comes before $R_1,\ R_4,\ R_5$ in the prioritized list it encodes, while R_3 is a (transitive) successor of $R_1,\ R_4,\ R_5$ in the precedence graph Prio. This accounts for three pairs in the disagreement (namely, $(R_3, R_1),\ (R_3, R_4),\ (R_3, R_5)$). Similarly, R_2 comes before $R_1,\ R_4,\ R_5$ in the prioritization, while it transitively follows them in Prio, hence three more disagreement pairs can be determined. On the contrary, no disagreement exists between the total order Pr_1 and the partial order Deps. As a consequence, the total disagreement for individual Pr_1 is 6. In a similar way, we can compute the disagreement between the other five individuals in Table 2 and the precedence graphs.

The best individuals are then evolved into the new population by applying some mutation and crossover operators to them (these operators are described in detail in the next section). In order to select the best individuals, we consider the disagreement measure as an indicator of fitness. However, it may happen that such an indicator does not allow a precise discrimination of some individuals, because they are associated with the same disagreement value. When two or more individuals have the same disagreement measure, we say they form a *tie*. In such a case we resort to the user, who supplies additional information to produce a precise fitness score. In other words, we resort to interactive user input whenever the score for some individuals produces a tie. In Table 2, this happens for individuals $Pr_1,\ Pr_2,\ Pr_3$ (having disagreement equal to 6) and for $Pr_5,\ Pr_6$ (9). The available fitness function cannot guide the search for an optimal prioritization, since $Pr_1,\ Pr_2,\ Pr_3$ (and $Pr_5,\ Pr_6$) cannot be ranked relative to each other. This indicates that the currently available precedence relationships do not allow choosing the best from these sets of individuals.

Table 2
Prioritized requirements and related disagreement.

Id	Reqs	Disagree
Pr_1	$\langle R_3, R_2, R_1, R_4, R_5\rangle$	6
Pr_2	$\langle R_3, R_2, R_1, R_5, R_4\rangle$	6
Pr_3	$\langle R_1, R_3, R_2, R_4, R_5\rangle$	6
Pr_4	$\langle R_2, R_3, R_1, R_4, R_5\rangle$	7
Pr_5	$\langle R_2, R_3, R_4, R_5, R_1\rangle$	9
Pr_6	$\langle R_2, R_3, R_5, R_4, R_1\rangle$	9

Table 3
Pairwise comparisons to resolve ties.

Tie	Pairs
Pr_1, Pr_2, Pr_3	$(R_1, R_2), (R_1, R_3), (R_4, R_5)$
Pr_5, Pr_6	(R_4, R_5)

We ask the user for information that allows us to rank the prioritizations in a tie. The pairs of requirements that are ordered differently in two (or more) prioritizations that form a tie are called *disagreement pairs* and these are the pairs to be elicited from the user. Specifically, we consider the disagreement between each couple of prioritizations in a tie. The pairs in the disagreement are those on which the equally scored prioritizations differ. Hence, we can discriminate them if we can decide on the precedence holding for such pairs. As a consequence, the information elicited from the user consists of a pairwise comparison between requirements that are ordered differently in equally scored prioritizations. If we consider the disagreement between Pr_1 and Pr_2, we get the pair (R_4, R_5). When comparing Pr_1 and Pr_3 we get (R_1, R_2) and (R_1, R_3).[2] The disagreement between Pr_2 and Pr_3 consists of all of these three pairs, so in the end the pairs in the disagreement for the tie $Pr_1,\ Pr_2,\ Pr_3$ is the set of three pairs shown in Table 3. The other tie (Pr_5, Pr_6) has only one pair in the disagreement, (R_4, R_5).

The user is requested to express a precedence relationship between each pair in the disagreement computed for prioritizations in a tie. Given a pair of requirements (e.g., R_1, R_2), the elicited ranking may state that one requirement should have precedence over the other one (e.g., $R_1 \rightarrow R_2$; or, $R_2 \rightarrow R_1$) or the user may answer *do not know*. In the first case a precedence edge is introduced in the elicited precedence graph. In the second case no edge is introduced. In our running example, the user would be requested to compare $(R_1, R_2), (R_1, R_3)$ and (R_4, R_5). Indeed, the first two cases represent a situation where the available precedence information is contradictory. In fact, the precedence graph Prio gives precedence to R_1, while Deps gives precedence to $R_2,\ R_3$. Hence, it is entirely justified to request additional user input, in order to determine a relative ordering of $R_1,\ R_2,\ R_3$, which is not obvious from the existing constraints. The third comparison requested to the user, (R_4, R_5), is a case where no precedence information is available in the requirement documents. As a consequence, it is impossible for the genetic algorithm to distinguish between Pr_1 and Pr_2, whose only difference consists of the ordering of R_4 w.r.t. R_5. Again, asking for additional user input makes perfect sense. When precedence information is available and is not contradictory, no additional user input is necessary.

[2] The order in which requirements are given in a disagreement pair is not meaningful. This means that (R_3, R_1) could be reported as a disagreement pair equivalent to (R_1, R_3).

After collecting user input in terms of pairwise comparisons, the existing elicited precedence graph is augmented with the new precedence edges. The appearance of cycles in the elicited graph indicates the existence of contradictory information. Since no precedence relation is associated with a cycle, cycles do not contribute to the disagreement increment.

When the new elicited precedence graph is available, the fitness function is recomputed for the individuals. Such a fitness evaluation is expected to be much more discriminating than the previous one, thanks to the additional information gathered through interaction. This means that the input to the fitness function used to score the individuals is partially provided by the user in the form of pairwise comparisons. Only pairs that actually make a difference in the fitness evaluation are submitted to the user for assessment. The best individuals, scored according to the new fitness function, are selected and mutated to constitute the next population. After a number of generations, the algorithm is expected to have successfully discriminated all best individuals in the population thanks to the input elicited from the user, leading to a final selection of the prioritization with lowest disagreement w.r.t. all precedence graphs (including the elicited one).

4. Algorithm

In this section, we describe an interactive genetic algorithm that implements the approach presented in the previous section. Before introducing the algorithm, we provide a formal definition of the intuitive notion of disagreement (taken from [1]), which plays a fundamental role when evaluating the fitness of an individual and when deciding which pairwise comparisons to elicit from the user. We give the definition in the general case where two partial orders are compared. A special case, quite relevant to the proposed method, is when one or both orders are total ones.

$$dis(ord_1, ord_2) = \{(p,q) \in ord_1^* | (q,p) \in ord_2^*\} \quad (1)$$

The disagreement between two (partial or total) orders ord_1, ord_2, defined upon the same set of elements R, is the set of pairs in the transitive closure[3] of the first order, ord_1^*, that appear reversed in the second order closure ord_2^*. A measure of disagreement is given by the size of the set $dis(ord_1, ord_2)$, under the assumption that all pairs are weighted the same. However, sometimes it is useful to give different weights to different precedence graphs, or even to specific edges of a given precedence graph. Different stakeholders, such as top managers and development team, may come out with different priority graphs, which can be given different weights depending on the stakeholders' importance. In these cases, the measure of disagreement must take weights into account. For example, if there are three precedence graphs *P, D, E* (priorities, dependencies and elicited pairs), we may want to weight differently the pairs in the disagreement set according to the graph they belong to. This can be easily achieved by applying the following equation:

$$wDis(ord, P \cup D \cup E) = w_P dis(ord, P) + w_D dis(ord, D) + w_E dis(ord, E) \quad (2)$$

where w_P, w_D, w_E are the weights given respectively to *P, D, E*. If individual weights are given to specific edges, we get a disagreement measure by summing up the weights of the edges appearing in the set $dis(ord_1, ord_2)$. In mixed cases, where some precedence graphs are weighted while others contain differently weighted edges, we can get the final disagreement measure by combining the two computations described above.

[3] The transitive closure is defined as follows: $(p,q) \in ord^*$ *iff* $(p,q) \in ord$ or $\exists r | (p,r) \in ord \wedge (r,q) \in ord^*$.

Algorithm 1. Compute prioritized requirements

Input R: set of requirements
Input $ord_1, \ldots, ord_k$: partial orders defining priorities and constraints upon R ($ord_i \subseteq R \times R$ defines a DAG)
Output $\langle R_1, \ldots, R_n \rangle$: ordered list of requirements
1: initialize *Population* with a set of ordered lists of requirements $\{Pr_i, \ldots\}$
2: *elicitedPairs* := 0
3: *maxElicitedPairs* := *MAX* (default = 100)
4: *thresholdDisagreement* := *TH* (default = 5)
5: *topPopulationPerc* := *PC* (default 5%)
6: *eliOrd* := $\emptyset$
7: **for each** Pr_i in *Population*
8: compute sum of *disagreement* for Pr_i w.r.t. $ord_1, \ldots, ord_k$
9: **end for**
10: **while** *minDisagreement* > *thresholdDisagreement* $\wedge$ *execTime* < *timeOut* **do**
11: sample *Population* with bias toward lower disagreement, e.g. using tournament selection
12: sort *Population* by increasing *disagreement*
13: **if** *minDisagreement* did not decrease during last *G* generations $\wedge$ there are ties in the *topPopulationPerc* of *Population* $\wedge$ *elicitedPairs* < *maxElicitedPairs* **then**
14: *eliOrd* := *eliOrd* $\cup$ elicit pairwise comparisons from user for ties
15: increment *elicitedPairs* by the number of elicited pairwise comparisons
16: **end if**
17: mutate *Population* using the *requirement-pair-swap* mutation operator
18: crossover *Population* using the *cut-head (tail)/fill-in-tail (head)* operator
19: **for each** Pr_i in *Population*
20: compute sum of *disagreement* for Pr_i w.r.t. $ord_1, \ldots, ord_k, eliOrd$
21: update *minDisagreement*
22: **end for**
23: **end while**
24: return Pr_{min}, the requirement list from *Population* with minimum *disagreement*

Algorithm 1 contains the pseudocode of the interactive genetic algorithm used to prioritize a set of requirements *R*. The other input of the algorithm is a set of one or more partial orders ($ord_1, \ldots, ord_k$), derived from the requirement documents (e.g., priority, dependencies, etc.).

The algorithm initializes the population of individuals with a set of totally ordered requirements (i.e., prioritizations). The initial population can be either computed randomly, or it can be produced by taking into account one or more of the input partial orders, so as to have an already good population to start with. Greedy heuristics may be used in this step to produce better initializations.

Steps 3–5 set a few important parameters of the algorithm. Namely, the maximum number of pairwise comparisons that can be reasonably requested to the user, the target level of disagreement we aim for (which is exploited at step 10 to stop the process of elicitation when the disagreement is below the given threshold), and the fraction of individuals with highest fitness (best individuals) that are considered for possible ties (exploited in the lines 13 and 14 of the algorithm), to be resolved through user interaction. Another relevant parameter of the algorithm is the maximum execution time (*timeOut*), which constrains the total optimization time. Moreover, the typical parameters of any genetic algorithm (population size, proportion of mutation w.r.t. crossover) apply here as well.

Initially, the fitness of the individuals is measured by their disagreement computed with reference to the input partial orders (steps 7–9). Then the main loop of the algorithm is entered. New generations of individuals are produced as long as the disagreement is above the given threshold. After the maximum allowed execution time, the algorithm stops anyway and reports the individual with minimum disagreement w.r.t. the initial partial orders and the partial order elicited from the user.

Inside the main evolutionary iteration (steps 11–22), the first operation to be performed is *selection* (step 11). While any selection mechanism could be used in principle with this algorithm, after several preliminary tests with "Roulette Wheel Selection" and "Elitism", we experimentally achieved the best performance when using "Tournament selection". When evolutionary optimization gets stuck for G generations (i.e., a locally minimum disagreement with available constraints is reached), the resulting population is sorted by decreasing disagreement and ties are determined for the set of the best individuals in the population (i.e., the set of individuals having lowest disagreement) or for a fraction of this set. If there are ties, the user is consulted in order to resolve them. Specifically, the pairs in the disagreement between equally scored individuals are submitted to the user for pairwise comparison. We take care to ensure that each pair is not presented to the user multiple times during the process. The result of the comparison is added to the elicited precedence graph (*eliOrd*).

After the selection and the optional interactive step, the population is evolved through mutation and crossover. For mutation, we use the *requirement-pair-swap* operator, which consists of selecting two requirements and swapping their position in the mutated individual. Selection of the two requirements to swap can be done randomly and may either involve neighboring or non-neighboring requirements. For example, if we mutate individual Pr_1 in Table 1 and select R_1, R_4 for swap, we get the new individual $Pr'_1 = \langle R_3, R_2, R_4, R_1, R_5\rangle$. More sophisticated heuristics for the selection of the two individuals to swap may be employed as well (e.g., based on the disagreement of the individual with the available precedence graphs). In this work we considered only random selection of the positions to swap.

For crossover we use the *cut-head/fill-in-tail* and the *cut-tail/fill-in-head* operators, which select a cut point in the chromosome of the first individual, keep either the head or the tail, and fill-in the tail (head) with the missing requirements, ordered according to the order found in the second individual to be crossed over. For example, if we cross over Pr_2 and Pr_3 using *cut-head/fill-in-tail* and selecting the separation between positions 2–3 as cut point, we get $Pr'_2 = \langle R_3, R_2, R_1, R_4, R_5\rangle$ and $Pr'_3 = \langle R_1, R_3, R_2, R_5, R_4\rangle$, i.e., we keep the head in both chromosomes ($\langle R_3, R_2\rangle$ and $\langle R_1, R_3\rangle$) and we fill-in the tail with the missing requirements in the order in which they appear respectively in Pr_3 and Pr_2. Selection of the cut point can be done randomly, but again there is room for more sophisticated heuristics, such as choosing a cut point associated with a requirement pair on which the individual is in disagreement with some precedence graph.

The mutation and crossover operators described above may, from time to time, generate chromosomes that are already part of the new population being formed. In general, this is not a problem (best individuals are represented multiple times), but it may become a problem in degenerate cases where most of the population has only a single or a few chromosomes. To overcome such a problem, it is possible to introduce a measure of population diversity and use it to limit the generation of chromosomes already present in the population being evolved. Mutation and crossover may be applied repeatedly, until the population diversity exceeds a predefined threshold.

The last steps of the algorithm (19–22) determine the fitness measure (disagreement) to be used during the next selection of the best individuals. This computation of the disagreement takes into account the initial partial orders as well as the elicited precedences obtained through successive user interactions.

The most distinguishing property of this algorithm is that it resorts to user input only when the available information is insufficient and at the same time availability of more information allows for a better fitness estimation. Hence, the requests made to the user are limited and the information provided by the user is expected to be most beneficial to finding a good prioritization. Fitness function computation is not entirely delegated to the user, which would be an unacceptable burden. Rather, it is only when the fitness landscape becomes flat and the search algorithm gets stuck that user interaction becomes necessary.

5. Case study

We applied the IGA algorithm to prioritize the requirements for a real software system, as part of the project ACube (Ambient Aware Assistance) [27]. ACube is a large research project funded by the local government of the Autonomous Province of Trento, in Italy, aiming at designing a highly technological smart environment to be deployed in nursing homes to support medical and assistance staff. In such context, an activity of paramount importance has been the analysis of the system requirements, to obtain the best trade off between costs and quality improvement of services in specialized centers for people with severe motor or cognitive impairments. From the technical point of view, the project envisages a network of sensors distributed in the environment or embedded in the end-users' clothes. This technology should allow monitoring the nursing home guests unobtrusively, that is, without influencing their usual daily life activities. By means of advanced automatic reasoning algorithms, the data acquired through the sensor network are going to be used to promptly recognize emergency situations and to prevent possible dangers or threats for the guests themselves. The ACube project consortium has a multidisciplinary nature, involving software engineers, sociologists and requirements analysts, and is characterized by the presence of professionals representing end-users directly engaged in design activities.

As a product of the end-user requirements analysis phase, 60 end-user requirements (49 technical requirements[4]) and three macro-scenarios have been identified. Specifically, the three macro scenarios are: (i) "localization and tracking to detect falls of patients", (ii) "localization and tracking to detect patients escaping from the nursing home", (iii) "identification of dangerous behavior of patients"; plus (iv) a comprehensive scenario that involves the simultaneous presence of the previous three scenarios.

Out of these macro-scenarios, detailed scenarios have been analyzed together with the 49 technical requirements. Examples of such technical requirements are:

> *"TR16: The system identifies the distance between the patient and the nearest healthcare operator"*

or

> *"TR31: The system infers the kind of event based on the available information"*

Table 4 summarizes the number of technical requirements for each macro-scenario. Together with the set of technical requirements, two sets of technical constraints have been collected during requirements elicitation: *Priority* and *Dependency*, representing respectively the priorities among requirements and their dependencies. In particular, the *Priority* constraint has been built by the software architect on the basis of the end-users' needs and it is de-

[4] We consider only functional requirements.

Table 4
The four macro-scenarios and the number of technical requirements associated with them.

Id	Macro-scenario	Number of requirements
FALL	Monitoring falls	26
ESC	Monitoring escapes	23
MON	Monitoring dangerous behavior	21
ALL	The three scenarios	49

fined as a function that associates each technical requirement to a number (in the range 1–500), indicating the priority of the technical requirement with respect to the priority of the end-user requirements it is intended to address. The *Dependency* feature is defined on the basis of the dependencies between requirements and is a function that links a requirement to the set of other requirements it depends on.

Finally, for each of the four macro-scenarios, we obtained the *Gold Standard* (GS) prioritization from the software architect of the ACube project. The GS prioritization is the ordering given by the software architect to the requirements when he planned their implementation during the ACube project. We take advantage of the availability of GS in the experimental evaluation of the proposed algorithm, in that we are able to compare the final ordering produced by the algorithm with the one defined by the software architect.

5.1. Research questions

The experiments we conducted aim at answering the following research questions:

RQ1 (Convergence) *Can we observe convergence with respect to the finally elicited fitness function?*

Since the fitness function is constructed incrementally during the interactive elicitation process, convergence is not obvious. In fact, initially IGA optimizes the ordering so as to minimize the disagreement with the available precedence graphs (*Priority* and *Dependency* in our case study). Then, constraints are added by the user (in our case the ideal user simulated by sampling the elicited pairs from the Gold Standard) and a new precedence graph (*eliOrd*) appears. Hence, the target of optimization is slightly and gradually changed. The question is whether the overall optimization process converges, once we consider (a posteriori) all precedence graphs, including the elicited one in its final form. We answer this research question by measuring the final fitness function values —i.e., disagreement (to be minimized) with all final precedence graphs— over each generation after the evolutionary process is over (hence, *eliOrd* has been gathered completely). It should be noticed that the final fitness function values are not available during the evolutionary optimization process, when they are approximated as the disagreement with the initial precedence graphs and the partially constructed *eliOrd*.

RQ2 (Comparison) *Does IGA produce improved prioritizations compared to IAHP?*

In our previous work [10] we showed that IGA outperforms non-interactive requirement ordering. In this paper, we compare IGA with another interactive requirement prioritization technique, IAHP [9]. IAHP (Incomplete AHP) is a variant of AHP [8] that can deal with incomplete information (pairwise comparisons) from the user. It was designed to support scalability of AHP to requirements whose size make AHP prohibitively expensive (the number of pairs that must be elicited to apply AHP is quadratic with the number of requirements). It represents the state of the art in interactive requirements prioritization. Appendix A provides a brief introduction to the IAHP algorithm.

In this research question we compare the output of IGA with the output of the IAHP algorithm, at equal number of pairs elicited from the user. We vary this number from low to high values, in order to investigate the regions where one approach is superior to the other (i.e., the degree of information incompleteness each approach can accommodate).

To assess the performance of IGA and IAHP we use two main metrics: number of disagreements between the rank and GS and average distance of the position of a requirement in the rank from the position of the same requirement in the GS. The latter metric is highly correlated with the former, but it has the advantage of being more easily interpretable than disagreement. In fact, disagreement involves a quadratic number of comparisons (each pair of requirements w.r.t. GS order), hence its absolute value is not straightforward to understand and compare. On the contrary, the distance between the position of each requirement in the prioritization produced by our algorithm and the position of the same requirement in the GS gives a direct clue to the number of requirements that are incorrectly positioned before or after the requirement being considered.

RQ3 (Role of weights) *How does the specification of weights affect the performance of IGA?*

The fitness function used by the IGA algorithm may be improved by specifying the relative importance of the various precedence graphs used for prioritization (including the elicited precedence graph). In Section 4 we have described how the disagreement computation, used as fitness measure by IGA, can be adjusted in the presence of weights, which can be specified for entire graphs or even for individual edges. This research question deals with the role of such weights. We want to understand whether the specification of weights can speed up the convergence to the final solution in a significant way. To this aim, we consider some alternative settings for the weights to be given to *Prio*, *Dep* and *Eli*. We discussed the considered settings with the system architect and the requirements analysts of Acube, so as to investigate those configurations which better match the relative importance of the available information, as perceived by the people who actually worked on the project.

RQ4 (Robustness) *Is IGA more robust than IAHP with respect to errors committed by the user during the elicitation of pairwise comparisons?*

In order to test the robustness of the IGA in comparison with IAHP at increasing user error rates, we simulate such errors by means of a simple stochastic model. We fix the probability of committing an elicitation error (p_e). Then, during the execution of the IGA and IAHP algorithms, whenever a pairwise comparison is elicited from the user, we generate a response in agreement with the GS with probability $1 - p_e$ and we generate an error (i.e., a response in disagreement with the GS) with probability p_e. We varied the probability of user error p_e from 5% to 20%.

5.2. Experiments performed

We executed several experiments to answer the research questions listed above. In this section we summarize the experimental settings used in each experiment.

5.2.1. Experimental design

The factors to be tested through our experimentation include: (a) *Applied Algorithm/Method*, for the experimental design we used our approach, i.e., IGA, along with Incomplete-AHP (IAHP). We compare resulting disagreement/distance of the final ordering produced as the outcome of the experimented approaches w.r.t. GS; (b) *User Model*, in our experiments we consider both the case where there is an ideal user (e.g. user makes no error) and a user that sometimes makes wrong decisions; to simulate such a user we resort to the

Table 5
Levels for the factors used in IGA and IAHP.

Factor	Levels
Algorithm	IGA (Interactive, Domain Info, User Knowledge through pairwise comparison) IAHP (Interactive, User Knowledge through pairwise comparison)

Table 6
Parameters of the user model used in the experiments.

Factor	Levels		
	Tot. eli. pairs	Err. rate (%)	Wrong eli. pairs
User model	25	0	0
		5	1
		10	2
		20	5
	50	0	0
		5	2
		10	5
		20	11
	100	0	0
		5	5
		10	10
		20	20

Table 7
Relative weights given to Prio, Dep and Eli constraint graphs.

Weight model	Prio	Dep	Eli
Base model (BM)	1	1	1
Weight model 1 (WM1)	2	1	4
Weight model 2 (WM2)	2	1	2
Weight model 3 (WM3)	2	1	1

GS ordering of pairs (or its opposite, if the simulated user is supposed to make an error); and, (c) *Available Knowledge*, the possibility of taking advantage of different sources of knowledge (i.e. user knowledge or domain knowledge, coming from the requirements analysis phase) and to give them different weights.

5.2.2. Experimental preparation: the levels of the factors

(a) *Applied Algorithm/Method*: The algorithm or method that we apply is the main factor (i.e. we expect the use of different algorithms or methods will produce different levels of disagreement/distance; see Table 5).

(b) *User Model*: In our experiment the user is an artificial user agent that replies to the requests of pairwise comparisons automatically, either in agreement with the GS (with probability $1 - p_e$), or in disagreement with it (with probability p_e). We have conducted experiments with all four macro scenarios (see Table 4) with a different number of pairs elicited from the user and with variations in user error rates. More specifically, for all macro scenarios we experimented with 25, 50 and 100 elicited pairs and error rates 0% (no errors), 5% 10% and 20%. Table 6 reflects the user and error model we used in our experiments (numbers remain same for all macro scenarios i.e. ALL, FALL, ESC and MON).

Table 9
Different experimental settings used for experimentation with IGA and IAHP.

Parameters	Values
targetAlgorithm	IGA, IAHP
populationSize (IGA)	50
mutationRate (IGA)	10%
maxElicitedPairs	25, 50, 100
executionTime (for IGA)	600 s, 840 s & 1080 s
userErrorRate	0%, 5%, 10%, 20%
targetMeasure	DIS, AD

In Table 6, the second column represents the total number of elicited pairs we experimented with. The third column represents the user error rates we used and the fourth column represents the calculated number of elicited pairs that are answered wrongly by the artificial user agent at runtime.

(c) *Weights*: We investigated three weight models that were regarded as interesting variants by the system architect and by the requirements analysts of the Acube project. They are listed in Table 7. They all give higher importance to the end-user's preferences (Prio), compared to the implementation constraints (Dep), which makes sense for a user-centered system such as Acube. The first one, which gives the highest importance (weight = 4) to the elicited constraints, is indeed the one suggested by the system architect and by most requirements analysts. The other two models, which gradually reduce the importance attributed to the elicited constraints while keeping a 2:1 ratio between Prio and Dep, have been suggested as variants of the first model that are worthy of investigation.

5.2.3. Experimental measures

To evaluate the outcome (i.e., the prioritized list of requirements) of the experimented methods, we considered the measurement scales reported in Table 8. The disagreement measure has already been specified in detail in Section 4. The average distance *AD* is computed by considering the difference between the positions of each requirement R_i in two total orders P_1, P_2, and taking the average over the requirements:

$$AD(P_1, P_2) = \frac{1}{n}\sum_{i=1}^{n} |pos(P_1, R_i), pos(P_2, R_i)| \quad (3)$$

5.2.4. Experimental settings

The experimental settings used to answer the research questions in the previous section are reported in Table 9. These parameters were used in all the scenarios considered in the experiments (ALL, FALL, ESC, MON). GA parameters are based on values commonly used in the literature and on a few preliminary calibration runs of the algorithm, in accordance with the offline parameter tuning method [28].

5.2.5. Experimental executions

Since IGA is non deterministic, we replicated each experiment a minimum number of 10 to maximum of 20 times with a time-out between 600 s and 1080 s. We varied the maximum execution-time in accordance with the maximum number of elicited pairs. The more pairs we elicit, the more operational time is assumed

Table 8
Different types of measurements used in the experiments with IGA and IAHP.

Measurement	Type	Description
Disagreement (DIS)	Count	Total count of mismatched pairs w.r.t. GS
Average Distance (AD)	Position	Position distance of each requirement w.r.t. its position in the GS

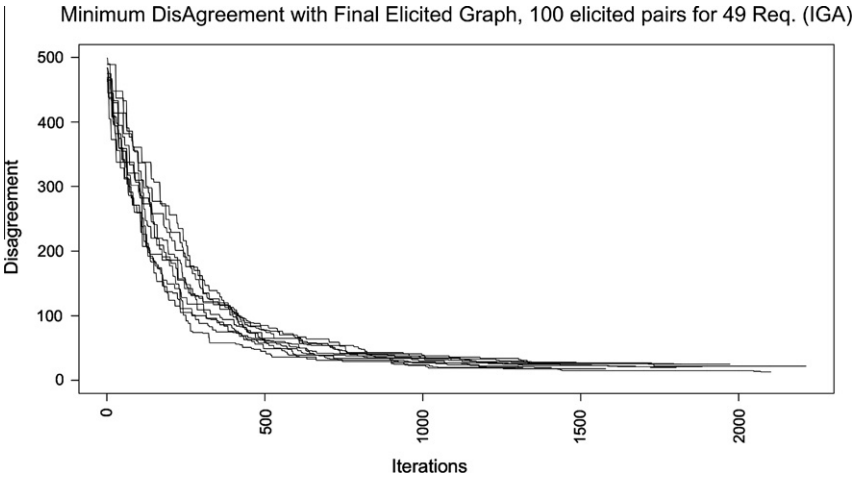


Fig. 1. Disagreement of the best individual in each population across generations in the ALL scenario (the *timeOut* is 1080 s)

to be available. So, we assumed a range of 600 s, 840 s and 1080 s as execution time while eliciting 25, 50 and 100 pairs respectively. The maximum number of elicited pairs is set to 100 based on our previous experience with pairwise comparison experiments involving humans. Above this threshold, results become unreliable because of fatigue, distraction and boredom. Our experience is corroborated by available cognitive psychology studies on user attention and performance [29]. For IAHP, we also performed experiments on 10 random initial orderings to observe its consistency w.r.t. the initialization and we terminate the prioritization process when we reach *maxElicitedPairs*. For both algorithms, we simulate the artificial user who automatically responds according to the GS both in error-free setting and in erroneous settings. After the optimization process terminates, we computed disagreement and average distance for the returned prioritized order w.r.t. GS. Finally, we produced box-plots for all the results we obtained, to be able to visually compare IGA and IAHP (or the other factors being investigated). We also executed statistical tests (e.g., ANOVA) to check whether the visually observed differences were also statistically significant.

5.3. Results

RQ1 *(Convergence)* Fig. 1 shows how the best individual in each population converges toward a low value of the final fitness function (i.e., disagreement with the final precedence graphs, including all elicited constraints). Thirty executions of the IGA algorithm are considered, on all the 49 requirements of the case study. While different runs exhibit slightly different behaviors and the final value obtained for the minimum disagreement differs from run to run, we can observe that the trend is always a steep decrease, which indicates the algorithm is indeed optimizing (minimizing) the final fitness function value, even though the fitness function is only partially known during the optimization iterations. Similar plots have been obtained for the three separate scenarios, FALL, ESC, MON.[5]

RQ2 *(Comparison)* Fig. 2 (top) shows that IGA can accommodate lack of information better than IAHP when the number of elicited pairs is small (in the range 25–125). In fact, the disagreement of the final requirement ordering produced by IGA w.r.t. the GS is substantially lower than that produced by IAHP, when provided with the same number of elicited pairs. Like AHP, IAHP is known to converge asymptotically to disagreement zero when all possible pairs are elicited. In fact, AHP is ensured to produce a disagreement equal to zero, but it requires a quadratic number of pairs to be elicited, which is usually not affordable. In the ALL scenario, since there are 49 requirements, AHP would need $49 \times 48/2 = 1176$ pairwise comparisons elicited from the user in order to be applied. In Fig. 2, we can notice that actually when the number of elicited pairs gets close to 1000, the disagreement of IAHP is converging toward zero. What's interesting in this plot is that up to 125 elicited pairs, IGA is superior (i.e., can make better use and can accommodate better information incompleteness) than IAHP. This is in part due to the capability of IGA of taking advantage of additional information sources (Prio and Dep precedence graphs). On the contrary, AHP and IAHP make use only of information elicited from the user. Fig. 2 (middle, bottom) show the boxplots of disagreement and distance at 25, 50 and 100 elicited pairs, which represent the interesting range of pairwise comparisons (i.e., number of comparisons that can be elicited in a reasonable time, with reasonable effort and acceptable user fatigue [29]) as also pointed out in the study described in [26]. It can be noticed that the results produced by IGA have substantially less variance than those produced by IAHP. This means that best and worst case executions are very similar in IGA, while when IAHP is used there is a non negligible risk of obtaining a performance substantially lower than the expected one. We can conclude that in the range of the number of pairs that can be reasonably elicited from the user (25–100), IGA has better performance than IAHP. Statistical significance of the observed differences was tested using ANOVA (see Table 10 for the ALL scenario; similar results have been obtained for the other scenarios).

RQ3 *(Role of weights)* Figs. 3, 4 show respectively disagreement and average position distance obtained by running IGA with 25/50/100 elicited pairs (top to bottom in both figures), on the ALL scenario (49 requirements), using different weight models that have been suggested by the software architect of the ACube project (see Table 7). It is apparent from both figures that weights play some role only when a small number of pairs is elicited (i.e., 25). In fact, the box plots for disagreement and distance at 50/100 elicited pairs do not reveal any significant difference. Even though the ANOVA test cannot find any statistically significant difference for weighted models compared to the base model, when a small number of pairs is elicited (25) a direction can be observed, especially for WM1 compared to BM. Weights seem also somewhat beneficial when weight model WM3 is adopted, while with weight model WM2 the benefits are not apparent.

In conclusion, weights play a minor role and some (statistically non-significant) effect can be observed only in the presence of

[5] See http://se.fbk.eu/sites/se.fbk.eu/files/TR-FBK-SE-2011-7.pdf for a detailed description of the results.

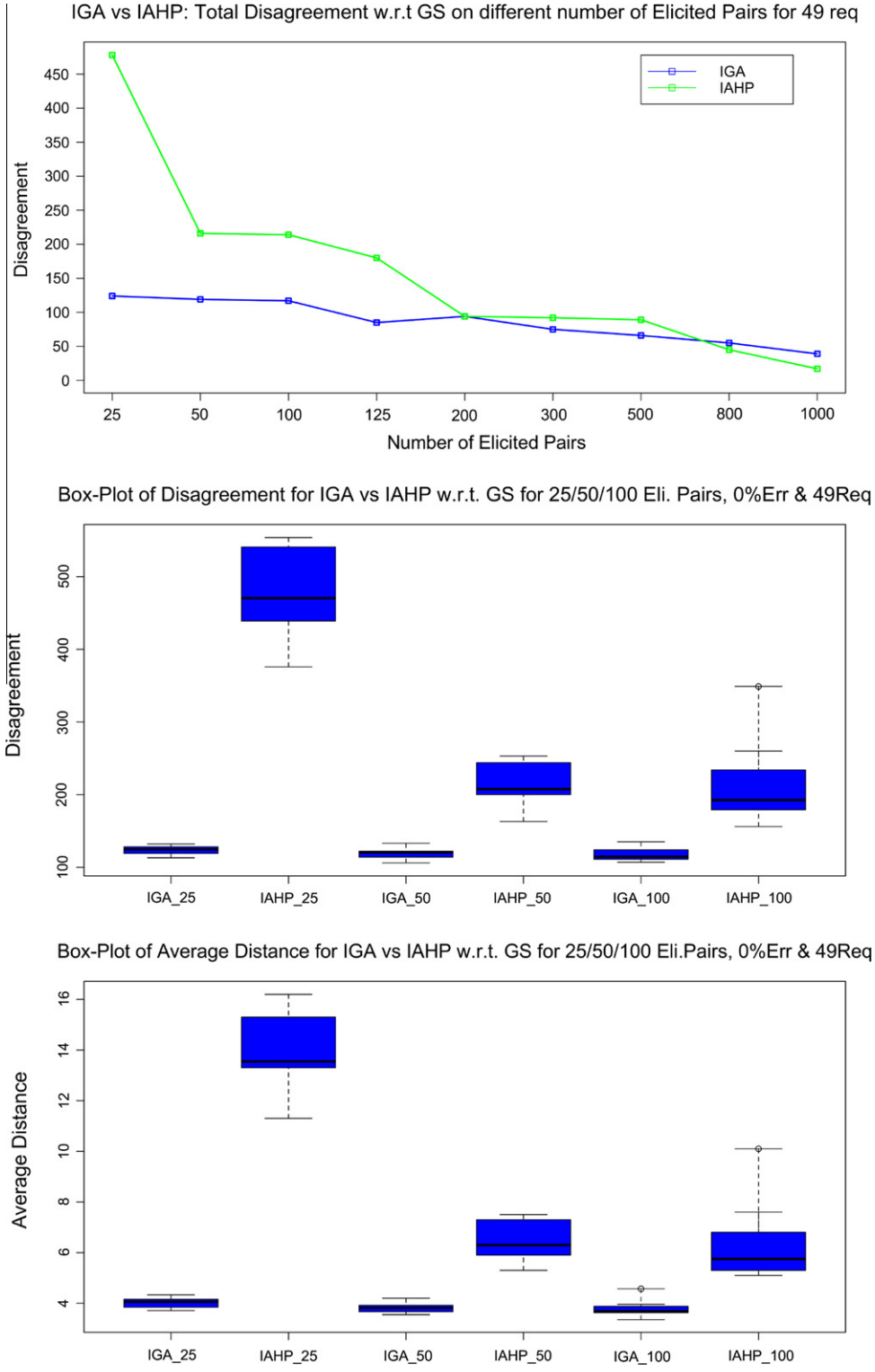


Fig. 2. Disagreement of IGA vs. IAHP at increasing elicited pairs (top); boxplots of disagreement with (middle) and distance from (bottom) GS after eliciting 25/50/100 pairs (ALL scenario).

strict limits on the number of pairs that can be elicited from the user. If such limits are low, a good weighting can improve (but to a limited extent) the final requirements ordering produced by IGA.

RQ4 *(Robustness)* Fig. 5 shows how the performance of the IGA and IAHP algorithms degrade at increasing user error rates. We show the results obtained for ALL and 50 elicited pairs, but similar plots are available for the smaller, separate scenarios and a different number of elicited pairs (25/50/100). As expected, disagreement (top) and distance (bottom) tend to increase as long as the user makes errors at increasing rate. However, such increase is much steeper for IAHP than for IGA, indicating that the IGA algorithm is quite robust with respect to the errors possibly affecting the user responses to pairwise comparisons, while IAHP is very sensitive to such errors. Even at a user error probability p_e as high as 20%, IGA has good performance (which remains better than non interactive algorithms, as shown in our previous work [10]), while

Table 10
Analysis of variance (ANOVA) comparing IGA and IAHP.

Disagreement	*p*-Value	Distance	*p*-Value
(IGA25, IAHP25)	$< 1.9e-13$	(IGA25, IAHP25)	$< 4.7e-14$
(IGA50, IAHP50)	$< 4.2e-9$	(IGA50, IAHP50)	$< 4.8e-9$
(IGA100, IAHP100)	$< 5.5e-5$	(IGA100, IAHP100)	$< 4.9e-5$

IAHP exhibits a major performance degradation, with the disagreement jumping above 300 and the distance above 9. Statistical significance of the better performance of IGA w.r.t. IAHP at increasing user error rate is confirmed by the outcome of the ANOVA test, reported in Table 11. We think the higher sensitivity of IAHP to user errors, as compared to IGA, is due to the computation of matrix derivatives, which are extremely sensitive to incorrect values. On the contrary, the fitness function used by IGA requires a pretty sim-

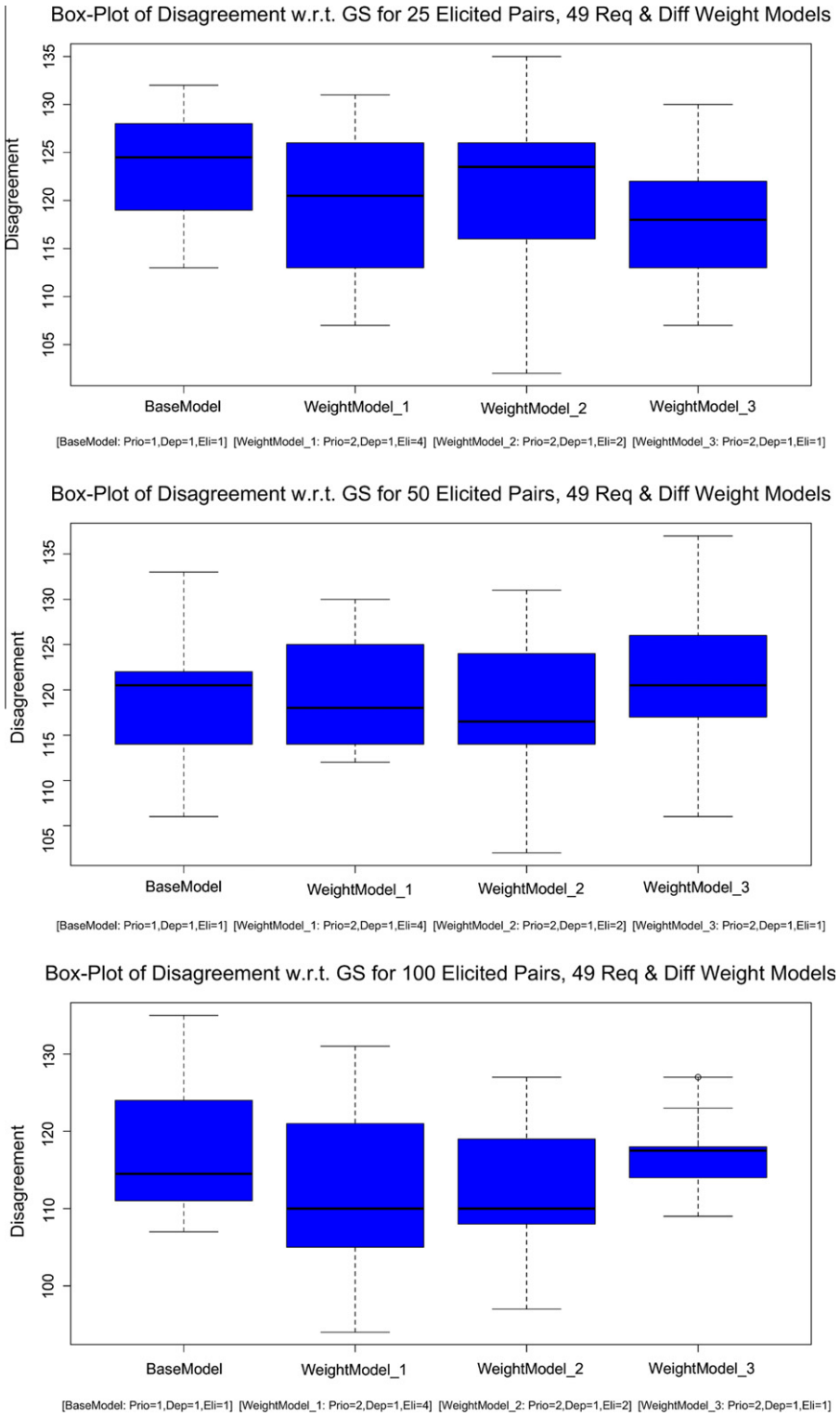


Fig. 3. Performance (disagreement) of IGA under different weighting of the precedence graphs in the ALL scenario at 25/50/100 elicited pairs (top to bottom).

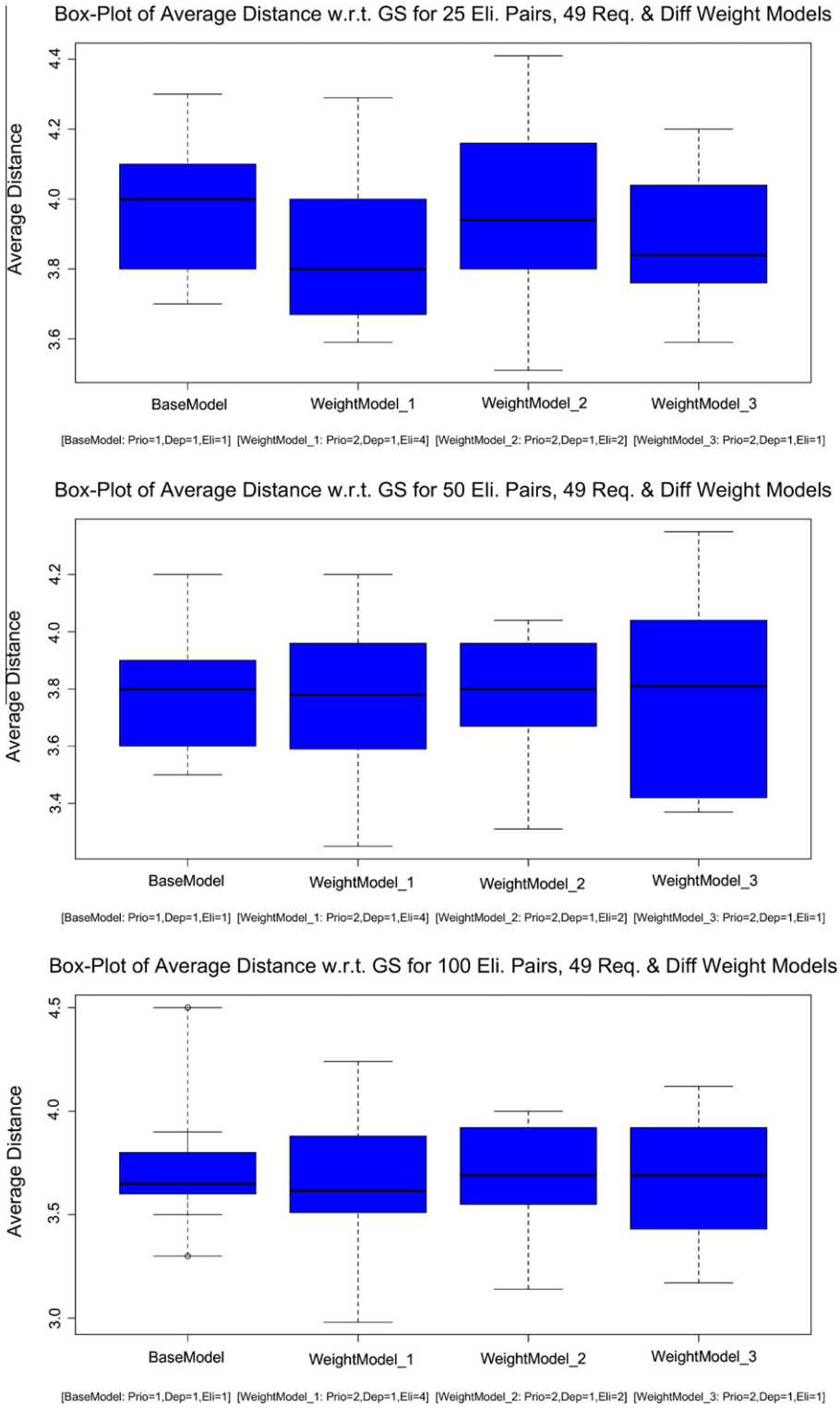


Fig. 4. Performance (distance) of IGA under different weighting of the precedence graphs in the ALL scenario at 25/50/100 elicited pairs (top to bottom).

ple disagreement computation, giving an overall measurement for the quality of the prioritization, which is only marginally affected by incorrect values.

5.4. Discussion

Based on the data collected from our experiments, we can answer positively to three out of four research questions (i.e., RQ1, RQ2 and RQ4). IGA converges even though the fitness function is known in its complete form only at the end of the elicitation process (RQ1). When comparing the prioritizations produced by the considered algorithms (IGA vs. IAHP) with GS, both in terms of disagreement and of position distance, IGA outperforms substantially IAHP, especially when a small number of pairwise comparisons can be carried out (RQ2). In the range 25–100 comparisons, the level of disagreement and the distance of requirements from

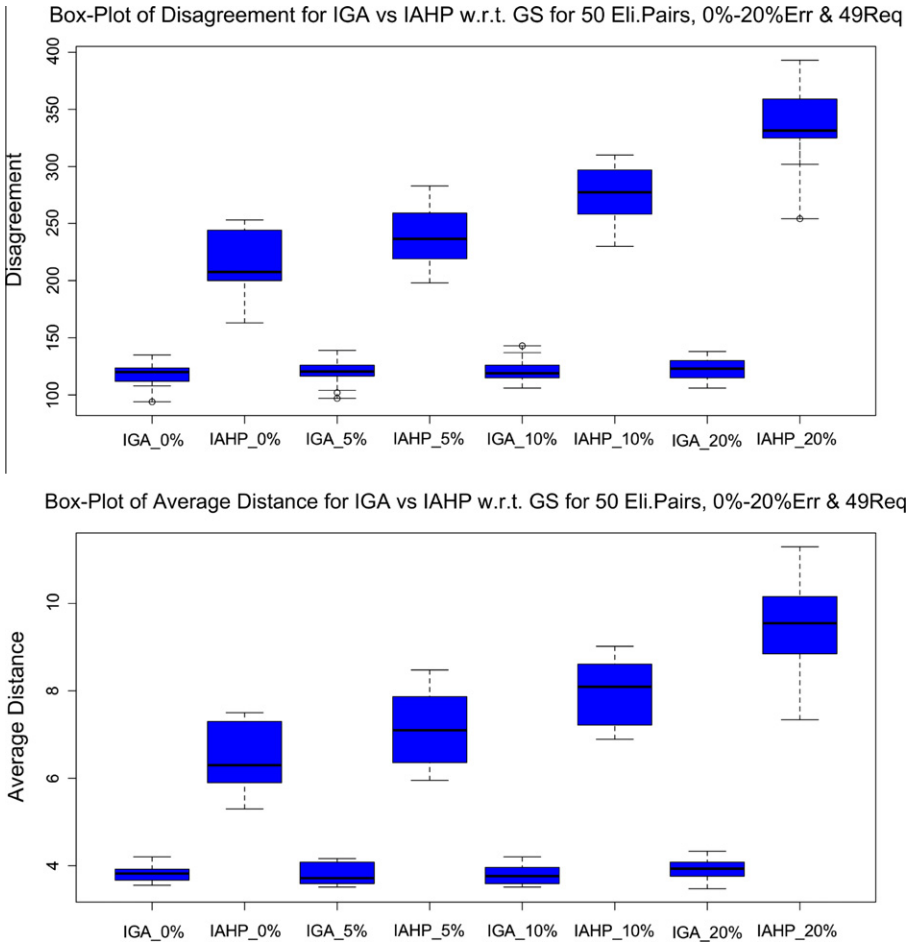


Fig. 5. Performance of IGA and IAHP at increasing user error rates in the ALL scenario.

the GS positions of IGA are significantly better than those of IAHP, as shown by the statistical test ANOVA and as apparent from the boxplots of the results. When only few pairwise comparisons can be elicited (e.g., 25) adoption of proper weighting for the available precedence information can amplify the benefits of the IGA approach a little. The behavior of the IGA algorithm is substantially more robust than IAHP in the presence of elicitation errors committed by the user (RQ4).

AHP requires exhaustive pairwise comparison, which is often impractical. It is definitely so in our case study, where it would be impossible to elicit 1176 comparisons from the user for the ALL scenario (in the subscenarios exhaustive elicitation is also impractical, requiring between 210 and 338 pairwise comparisons). In the ALL scenario, with an artificial user which makes no error, AHP would produce a final ordering which has no disagreement with GS. By eliciting only a small fraction (at most 10%) of the pairs required by AHP, we cannot expect to be equally good in terms of disagreement. In fact, the final disagreement produced by IGA is not zero. However, IGA can make better use of the elicited information than the incomplete variant of AHP, IAHP. In fact, by combining the information coming from the user with other precedence graphs (Prio and Dep), IGA can converge more quickly to an ordering which is very close to the GS. If weights are specified by the user, convergence is even faster. Moreover, in the presence of user errors, IGA has a substantially more robust behavior than IAHP.

5.5. Threats to validity

The main threat to the validity of our case study concerns the *external validity*, i.e., the possibility to generalize our findings to requirements collected for other systems and having different features. Since we conducted only one case study, the natural way to corroborate our findings and make them applicable to other systems is by replicating this study on other, different cases. Although considering just one case, we did our best to exploit it as much as possible. Specifically, we considered four macro scenarios in addi-

Table 11
Analysis of variance (ANOVA) comparing IGA and IAHP at increasing user error rates, for 50/100 elicited pairs.

Disagreement	p_e (%)	p-Value	Distance	p_e (%)	p-Value
(IGA50, IAHP50)	5	$< 1.7e-10$	(IGA50, IAHP50)	5	$< 6.6e-10$
(IGA100, IAHP100)	5	$< 8.6e-7$	(IGA100, IAHP100)	5	$< 2.3e-6$
(IGA50, IAHP50)	10	$< 1.3e-12$	(IGA50, IAHP50)	10	$< 3.6e-12$
(IGA100, IAHP100)	10	$< 1.2e-5$	(IGA100, IAHP100)	10	$< 2.5e-5$
(IGA50, IAHP50)	20	$< 2.4e-12$	(IGA50, IAHP50)	20	$< 4.1e-12$
(IGA100, IAHP100)	20	$< 5.0e-13$	(IGA100, IAHP100)	20	$< 2.5e-13$

tion to the complete one, and we considered the same set of precedence constraints, but with different weights associated with them. This provides a little more evidence for generalisation of the results.

Other threats to validity regard the *construct validity*, i.e., the observations we made to test our research hypotheses. Specifically, we used disagreement and requirement position distance as the metrics that determine the algorithm's performance. Other metrics may be more meaningful or more appropriate. On the other hand, disagreement is widely used in the related literature and position distance is a better interpretable alternative. A construct threat to validity is the choice of the gold standard ordering of the requirements, which determines the performance of the methods being compared. We decided to use the system architect's ordering as the gold standard, because this is the order in which requirements have been actually implemented, but of course we have no warranty that this is indeed the optimal order. Another construct validity threat might be related to the simple user error model we used to simulate a user who occasionally makes errors. We will experiment with more sophisticated models in the future. To minimize the *conclusion validity* threats (concerning the possibility to reject the null hypothesis), whenever we reported a difference to exist between two datasets, we checked our claim using statistical tests (ANOVA).

6. Conclusions and future work

We used a genetic algorithm to incrementally acquire user knowledge about the relative importance of pairs of requirements and to combine such information with dependencies and priorities available from requirements documents. The proposed approach was validated on a real case study, consisting of a number of requirements which make prioritization methods based on exhaustive elicitation (AHP) inapplicable. We compared our approach with IAHP (a variant of AHP which admits incomplete requirement elicitation, hence avoiding the scalability problems of AHP). Results indicate that especially when the number of elicited pairs is low, but in any case between 25 and 125 elicited pairs, IGA outperforms IAHP. We verified also the robustness of the IGA algorithm in the presence of user errors, in comparison with IAHP. IGA outperforms IAHP also in contexts where the input from the user is only partially reliable, since it is substantially more robust with respect to user errors.

We think the combination of knowledge elicitation and a search heuristic (such as GA) is potentially useful in several software engineering activities, going beyond the problem of prioritizing the requirements. Whenever the objective of the activity can be quantified (approximately) by a fitness function and the user has relevant knowledge that cannot be made fully explicit (because of the huge cost involved), an IGA-like approach can be very effective for the selective elicitation of the relevant information. Examples of software engineering activities that potentially have such characteristics include: project planning, test plan definition, assessment of design and architectural alternatives.

As part of our future work, we will conduct more experiments on other case studies to corroborate our findings. We will carry out additional comparisons with methods that have an internal stopping criterion (e.g., CBRank). These methods may not ensure that the number of elicitations is within the allocated budget, while with IGA the number of elicited pairs is a configurable parameter of the algorithm. However, we could carry out the comparison by configuring IGA to stop after eliciting the same number of pairs as the alternative method. We intend also to design and conduct an empirical study with human subjects in the role of requirement analysts. Moreover, we aim to investigate possible stopping rules for IGA, for example, adopting ideas from the stopping rule defined in IAHP.

Appendix A. AHP and incomplete AHP

Here we briefly introduce the Analytical Hierarchy Process (AHP) [8] method and its incomplete version defined by Harker [9], that represent the state-of-the-art in interactive requirements prioritization.

A.1. AHP method

AHP was developed by Thomas Saaty and was first applied to requirement prioritization by Karlsson and Ryan [4]. AHP can be considered one of the reference methods adopting a pairwise comparison strategy, allowing to define the prioritization criteria through a priority assessment of all the possible pairs of requirements.

Given a set of n requirements $R = \{R_1, \ldots, R_i, R_j, \ldots, R_n\}$, the first step in AHP is the construction of an $n \times n$ matrix whose rows and columns represent the candidate requirements. In this step a pairwise comparison iterative process is used in which the decision maker is required to give an integer value for preference $p_{ij} \in [1 \cdots 9]$, between two requirements, for each possible pairs of requirements. The value represents a qualitative measure of the preference relation, so, given two requirements R_i and R_j if the requirement R_i is "equally important" as requirement R_j with respect to the given criterion, $p_{ij} = 1$ is given, if the requirement R_i is "moderately more important" than requirement R_j the value $p_{ij} = 3$ is given, if the requirement R_i is "strongly more important" than requirement R_j the value $p_{ij} = 5$ is given, and so on; a set of possible values are presented in Table A.12. The elicited value p_{ij} is inserted in the corresponding cell of the matrix (R_i, R_j), while the cell (R_j, R_i) is filled with the reciprocal of the value $p_{ji} = 1/p_{ij}$. When all the pairs have been evaluated, a ranking is synthesized through the computation of the principal eigenvector of the matrix (i.e., the eigenvector with the highest eigenvalue norm). Each component of the principal eigenvector represents the rank of each requirement.

AHP has the important property of detecting the inconsistencies of the decision maker while eliciting priorities, providing a way to measure the confidence in the prioritization results. On the other hand, this method becomes of difficult application as soon as the number of requirements increases, thus inducing scalability problems. In fact, the user has to elicit $n(n-1)/2$ pair-wise comparisons to compute the solution.

A.2. IAHP method

To overcome the AHP scalability issues, Harker [9] proposed an incomplete version of AHP (IAHP). This method relies on the key idea of minimizing the number of pairs elicited from the decision maker, maintaining, at the same time, a good trade-off between the precision of the final solution and the effort of the decision maker. This is done by calculating, at each iteration of the elicitation process, a prediction of the next most promising pair to be asked to the decision

Table A.12
A possible fundamental scale in the interval $[1 \cdots 9]$ used for AHP.

Preference p_{ij}	Definition
1	R_i Equally important to R_j
3	R_i Moderately more important than R_j
5	R_i Strongly more important than R_j
7	R_i Very strongly or demonstrated more important than R_j
9	R_i Extremely more important than R_j
2, 4, 6, 8	For compromise between the above values

maker in order to find a stable and good approximation of the target ranking, early before eliciting all the $n(n-1)/2$ pairs.

In particular, the steps used in IAHP are:

1. The decision maker provides $n-1$ judgments which form a spanning tree. The n nodes of the tree are the requirements. The $n-1$ edges represent the pairwise comparisons elicited from the user. Each edge has an intensity taken from the same scale as the one used by AHP and shown in Table 12.
2. Using the available pairwise comparisons, the missing comparisons are derived by taking the geometric mean of the intensities, computed on a sample of the paths that connect start and end node (for each pair of nodes not connected by an edge). With these estimated intensities, the weight matrix is complete.
3. The derivatives of the weight matrix are computed with respect to the missing matrix elements and the next question (pairwise comparison to elicit) is selected as the one which maximizes the norm of the derivative (hence having the largest impact on the estimated intensities).
4. If the stopping criterion is met (e.g., maximum number of pairs has been elicited), then the algorithm stops and the final rank is calculated from the principal eigenvector of the matrix (as in AHP), else the selected pairwise comparison is elicited and the algorithm continues with step 2.

Other stopping criteria may be adopted, instead of the maximum number of pairwise comparisons. For instance, if the maximum absolute difference in the estimated weights from one question to another is below a given threshold, we can assume that convergence has already been reached. Another convergence indicator is that the ordinal ranking given by the principal eigenvector components remains unchanged.

References

[1] P. Avesani, C. Bazzanella, A. Perini, A. Susi, Facing scalability issues in requirements prioritization with machine learning techniques, in: RE 2005, pp. 297–306.
[2] J. Karlsson, S. Olsson, K. Ryan, Improved practical support for large scale requirements prioritizing, Journal of Requirements Engineering 2 (1997) 51–67.
[3] J. Karlsson, Software requirements prioritizing, in: Proceedings of 2nd International Conference on Requirements Engineering (ICRE '96), pp. 110–116.
[4] J. Karlsson, K. Ryan, A cost-value approach for prioritizing requirements, IEEE Software 14 (1997) 67–74.
[5] S. Sivzittian, B. Nuseibeh, Linking the Selection of Requirements to Market Value: A Portfolio – Based Approach, in: REFSQ 2001.
[6] H.P. In, D. Olson, T. Rodgers, Multi-criteria preference analysis for systematic requirements negotiation, in: COMPSAC 2002, pp. 887–892.
[7] F. Moisiadis, Prioritising software requirements, in: SERP 2002.
[8] T.L. Saaty, L.G. Vargas, Models, Methods Concepts & Applications of the Analytic Hierarchy Process, Kluwer Academic, 2000.
[9] P.T. Harker, Incomplete pairwise comparisons in the analytic hierarchy process, Mathematical Modelling 9 (1987) 837–848.
[10] P. Tonella, A. Susi, F. Palma, Using interactive GA for requirements prioritization, in: 2nd International Symposium on Search Based Software Engineering, IEEE, pp. 57–66.
[11] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering (2010) 1.
[12] J. Quiroz, S. Louis, A. Shankar, S. Dascalu, Interactive genetic algorithms for user interface design, in: IEEE Congress on Evolutionary Computation, pp. 1366–1373.
[13] A. Herrmann, M. Daneva, Requirements prioritization based on benefit and cost prediction: an agenda for future research, in: RE, IEEE Computer Society, 2008, pp. 125–134.
[14] M. Daneva, A. Herrmann, Requirements prioritization based on benefit and cost prediction: A method classification framework, in: EUROMICRO-SEAA, IEEE, 2008, pp. 240–247.
[15] J. del Sagrado, I. del Águila, F. Orellana, Ant colony optimization for the next release problem: a comparative study, in: 2nd International Symposium on Search Based Software Engineering, IEEE, pp. 67–76.
[16] Y. Zhang, M. Harman, Search based optimization of requirements interaction management, in: 2nd International Symposium on Search Based Software Engineering, IEEE, pp. 47–56.
[17] D. Greer, G. Ruhe, Software release planning: an evolutionary and iterative approach, Information and Software Technology 46 (2004) 243–253.
[18] Y. Zhang, M. Harman, S.A. Mansouri, The multi-objective next release problem, in: GECCO '07, ACM, 2007, pp. 1129–1137.
[19] J.T. de Souza, C.L.B. Maia, T. do Nascimento Ferreira, R.A.F. do Carmo, M.M.A. Brasil, An ant colony optimization approach to the software release planning with dependent requirements, in: M.B. Cohen, M.Ó. Cinnéide (Eds.), SSBSE, Lecture Notes in Computer Science, vol. 6956, Springer, 2011, pp. 142–157.
[20] A. Finkelstein, M. Harman, S. Mansouri, J. Ren, Y. Zhang, A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making, Requirements Engineering 14 (2009) 231–245.
[21] D. Leffingwell, D. Widrig, Managing Software Requirements: A Unified Approach, Addison-Wesley Longman Inc., 2000.
[22] K.E. Wiegers, Software Requirements Best Practices, Microsoft Press, 1999.
[23] S. Lauesen, Software Requirements: Styles and Techniques, Addison Wesley, 2002.
[24] A. Cockburn, Agile Software Development, Addison Wesley, 2001.
[25] A. Ngo-The, G. Ruhe, Requirements negotiation under incompleteness and uncertainty, in: Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering, SEKE, San Francisco Bay, CA, USA, pp. 586–593.
[26] A. Perini, F. Ricca, A. Susi, Tool-supported requirements prioritization. Comparing the AHP and CBRank method, Information and Software Technology 51 (2009) 1021–1032.
[27] R. Andrich, F. Botto, V. Gower, C. Leonardi, O. Mayora, L. Pigini, V. Revolti, L. Sabatucci, A. Susi, M. Zancanaro, ACube: User-Centred and Goal-Oriented techniques, Technical Report, Fondazione Bruno Kessler – IRST, 2010.
[28] A.E. Eiben, S.K. Smit, Evolutionary algorithm parameters and methods to tune them, in: E.M.Y. Hamadi, F. Saubion (Eds.), Autonomous Search, Springer, 2011.
[29] D.A. Allport, Attention and performance, in: G. Claxton (Ed.), Cognitive Psychology: New Directions, Routledge & Kegan Paul, London, UK, 1980.