Using Interactive GA for Requirements Prioritization

Paolo Tonella, Angelo Susi
Fondazione Bruno Kessler
Software Engineering Research Unit
Trento, Italy
tonella, susi@fbk.eu

Francis Palma
University of Trento
Department of Inf. Eng. and Computer Science
Trento, Italy
francis.palma@studenti.unitn.it

***Abstract*—The order in which requirements are implemented in a system affects the value delivered to the final users in the successive releases of the system. Requirements prioritization aims at ranking the requirements so as to trade off user priorities and implementation constraints, such as technical dependencies among requirements and necessarily limited resources allocated to the project.**

Requirement analysts possess relevant knowledge about the relative importance of requirements. We use an Interactive Genetic Algorithm to produce a requirement ordering which complies with the existing priorities, satisfies the technical constraints and takes into account the relative preferences elicited from the user. On a real case study, we show that this approach improves non interactive optimization, ignoring the elicited preferences, and that it can handle a number of requirements which is otherwise problematic for state of the art techniques.

***Keywords*-requirements prioritization; interactive genetic algorithms; search based software engineering.**

I. Introduction

The role of requirements prioritization is of extreme importance during the software development lifecycle, when planning for the set of requirements to implement in the successive system releases, according to information concerning the available budget, time constraints as well as stakeholder expectations and technical constraints.

The process of requirements prioritization can be viewed as the process of finding an order relation on the set of requirements under analysis. This process can be designed as an *a priori* or as an *a posteriori* process. In the former case, the preferences are formulated before the specification of the set of requirements via predefined models, for example, based on ranking criteria induced by the requirements attributes and their values, independently of the current set of requirements that are to be evaluated. In the *a posteriori* approaches, the ranking is formulated on the basis of the characteristics of the set of requirements under analysis, e.g., via a process of pairwise comparison allowing to define at the same time which requirement and why it has to be preferred between two alternatives.

In our work, we focus on an *a posteriori* approach, based on the Interactive Genetic Algorithm (IGA) search-based technique and on pairwise preference elicitation, with the objective of extracting relevant knowledge from the user and of composing it with the relevant ordering criteria induced by the attributes describing the requirements. The final objective is that of minimizing the user decision-making effort, increasing as much as possible the accuracy of the final requirements ranking.

Several approaches to requirements prioritization have been proposed in the last years [1], [2], [3], [4], [5]. Among the prioritization techniques used in these methods, Analytical Hierarchy Process (AHP) [6] exploits a pairwise comparison technique to extract the user knowledge with respect to the ranking of the requirements. AHP defines the prioritization criteria through a priority assessment of all the possible pairs of requirements. In general, available *a posteriori* prioritization methods (including AHP) suffer scalability problems.

Our solution belongs to the class of pairwise comparison methods and exploits an IGA approach to minimize the number of pairs to be elicited from the user. Elicited pairs and initial constraints on the relative ordering of requirements define the fitness function, which consists of the disagreement between the requirements ordering encoded in an individual and the initial and elicited constraints. Since elicitation and optimization are conducted at the same time and they influence each other, a peculiar trait of our algorithm is that the input to the fitness function is constructed incrementally, being only partially known at the beginning. This makes convergence a non trivial issue. We assessed the effectiveness of our 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. Results indicate that IGA converges and improves the performance of GA (without interaction) in a substantial way, while keeping the user effort (number of elicited pairs) acceptable.

In this paper, we first describe some relevant related works (Section II), then we give an intuitive description of the IGA approach (Section III), followed by a formal presentation of the algorithm (Section IV). Then, we describe a set of experimental evaluations about convergence, effectiveness and robustness of IGA (Section V). The empirical assessment was conducted on a real set of requirements. Conclusions

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.17

IEEE computer society

and future work are presented in Section VI.

II. Related Work

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* [7]. Other, alternative techniques consist of assigning each requirement to one specific class among a set of predefined priority classes, as for instance in *Numerical Assignment* [7], [8] and in *Top-ten requirements* [9].

Among the pairwise techniques, CBRank [10] 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).

The Analytical Hierarchy Process (AHP) [6] can be considered one of the reference methods which adopt a pairwise comparison strategy, allowing to define the prioritization criteria through a priority assessment of all the possible pairs of requirements. This method becomes impractical as soon as the number of requirements increases, thus inducing scalability problems in using this technique, only partially addressed by the introduction of heuristic stopping rules.

Among the methods exploiting genetic algorithms for requirements management, in [11] the EVOLVE method supports continuous planning for incremental software development. This approach is based on an iterative optimization method supported by a genetic algorithm. In [12], the authors focus on the specific Multi-Objective Next Release Problem (MONRP) in Requirements Engineering and present the results of an empirical study on the suitability of weighted and Pareto optimal genetic algorithms, together with the non-dominated sorting genetic algorithm (NSGA-II), presenting evidence to support the claim that NSGA-II is well suited to the MONRP. These two works 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.

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, the 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 having ambiguous or unreliable relative ordering. It resorts to elicitation only for those pairs which are expected to affect the final ordering to a major extent.

III. 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 is over.

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	

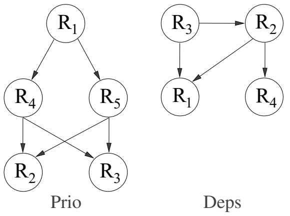

Table I
Requirements with priority and dependencies

Let us consider the five requirements listed in Table I. 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 I 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 (directly acyclic graphs). In fact, cyclic (sub-)graphs provide no useful information, since they represent precedences that are compatible with any requirements ordering. 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 implemented before the target requirement. Edges may be weighted, to actually quantify the strength or importance of such a precedence and an infinite weight may be used for precedence relations that must necessarily hold in the final ordering of the requirements. For simplicity, in this section we assume that each edge has a weight equal to one, but all the arguments, as well as the algorithm described in the next section, can be applied almost unchanged to weighted precedence graphs. Weights could be given to individual edges or, alternatively, to the whole precedence graph associated with a constraint or property.

At the right of Table I, 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 raise to the edges (R_2, R_1) and (R_3, R_1) in Deps.

Id	Reqs	Disagree
Pr_1	$< R_3, R_2, R_1, R_4, R_5 >$	6
Pr_2	$< R_3, R_2, R_1, R_5, R_4 >$	6
Pr_3	$< R_1, R_3, R_2, R_4, R_5 >$	6
Pr_4	$< R_2, R_3, R_1, R_4, R_5 >$	7
Pr_5	$< R_2, R_3, R_4, R_5, R_1 >$	9
Pr_6	$< R_2, R_3, R_5, R_4, R_1 >$	9

Table II
PRIORITIZED REQUIREMENTS AND RELATED DISAGREEMENT

The interactive genetic algorithm we use for requirements prioritization evolves a population of individuals, each representing a candidate prioritization. Table II shows 6 individuals (i.e., 6 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 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 II, 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 II and the precedence graphs.

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)

Table III
PAIRWISE COMPARISONS TO RESOLVE TIES

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. 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 II, 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.

We ask the user for information that allows us to rank the prioritizations in a tie. 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). 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 III. 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 *don't 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.

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 a cycle is compatible with any ordering of requirements. Hence, whenever a cycle is introduced in the elicited graph, we ask the user to break it.

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 discriminative 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).

IV. 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 [10]), 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.

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

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[1] 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 $dis(ord_1, ord_2)$.

Algorithm 1 Compute prioritized requirements

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

[1] 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 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 ($ord_1, ..., ord_k$), 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 at, and the fraction of individuals with highest fitness that are considered for possible ties, 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 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, we experimented 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 best (top fraction) individuals in the population. If there are ties, the user is resorted to in order to resolve them. Specifically, the pairs in the disagreement between equally scored individuals are submitted to the user for pairwise comparison; in this work we take care 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 I and select R_1, R_4 for swap, we get the new individual $Pr'_1 =< R_3, R_2, R_4, R_1, R_5 >$. 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.

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 $Pr'_2 =< R_3, R_2, R_1, R_4, R_5 >$ and $Pr'_3 =< R_1, R_3, R_2, R_5, R_4 >$, i.e., we keep the head in both chromosomes ($< R_3, R_2 >$ and $< R_1, R_3 >$) 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 of 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 generated. Mutation and crossover are 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.

V. 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) [13]. 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 users' clothes. This technology should allow monitoring the nursing home guests unobtrusively, that is, without influencing their usual daily life activities. Through 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 analysts, and is characterized by the presence of professionals representing end users directly engaged in design activities.

As a product of the user requirements analysis phase, 60 user requirements (49 technical requirements[2]) 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 behaviors 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 IV 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 on the basis of the users' needs and it is defined 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 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.

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

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

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 planning 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.

A. 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 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 (i.e., disagreement with all final precedence graphs) over generations 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*.

[2] We consider only functional requirements.

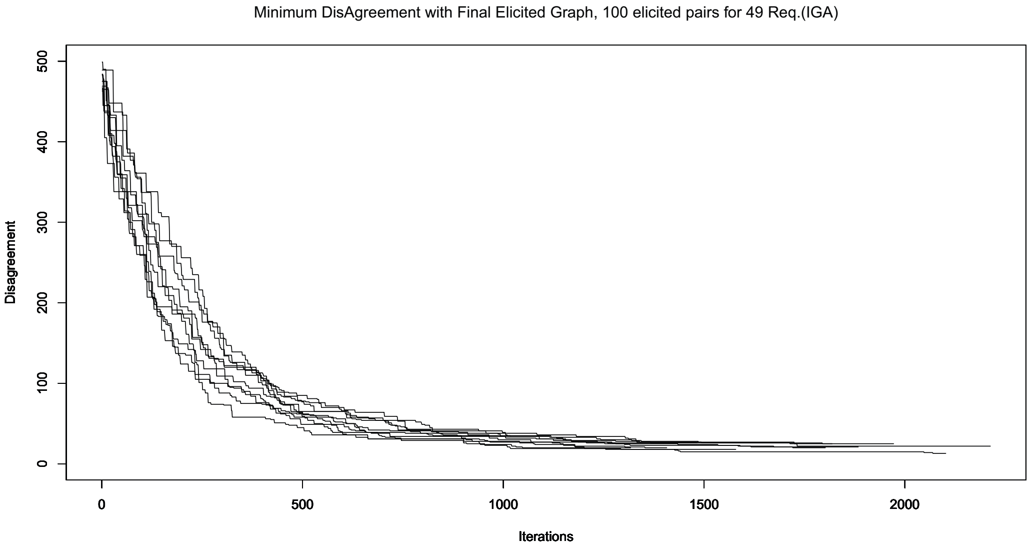

Figure 1. Disagreement of the best individual in each population across generations in the ALL scenario (the $timeOut$ is 1080 seconds)

RQ2 (Role of interaction) *Does IGA produce improved prioritizations compared to non-interactive requirement ordering?*

Our research hypothesis is that user knowledge plays an important role in requirement prioritization. RQ2 is designed to test such hypothesis. To assess the importance of the information elicited from the user, we compare the output of IGA with the output of algorithms which do not take into account any user provided knowledge. As a sanity check, we consider random requirement orderings (RAN). Moreover, in addition to IGA we apply GA, i.e., a local minimum is searched which minimizes the disagreement with the initially available precedence graphs, without eliciting any pair from the user (this is trivially achieved by setting *maxElicitedPairs* to 0 in Algorithm 1).

To assess the performance of our algorithm we use two main metrics: disagreement with GS and average distance from the position of each requirement in the GS. The latter metrics 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 on the number of requirements that are incorrectly positioned before or after the requirement being considered.

RQ3 (Role of initial precedence constraints) *How does initial availability of precedence constraints affect the performance of IGA?*

IGA is expected to be particularly effective when little information is available upfront, in terms of precedence constraints produced during the requirements elicitation phase (e.g., priorities and dependencies). This means that IGA would be particularly recommended in contexts in which requirement documents do not provide complete information about stakeholders' priorities and mutual dependencies. In order to test whether this is true in our case study, we conducted experiments in which only a subset of the available information was used as the initial precedence graphs. Specifically, we prioritized the requirements by means of IGA using only Prio or only Dep as the initial precedence graph, and we compared the improvement margin in these two situations with respect to the improvement achieved when both precedence graphs are available.

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

In order to test the robustness of the proposed algorithm at increasing user error rates, we simulate such errors by means of a simple stochastic model. We fix a probability of committing an elicitation error, say p_e. Then, during the execution of the IGA algorithm, 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%.

B. Results

Since the IGA algorithm involves non deterministic steps (e.g., when applying mutation or crossover), we replicated each experiment, typically 30 times, with a $timeOut$ of 1080 seconds, and computed average and box plots over such runs. When the algorithm involves the user interactively, we simulate the user response by means of an artificial user which replies automatically. In the default setting, the artificial user makes no elicitation error, i.e., it always replies to a pairwise comparison with a relative ordering of the two requirements being compared which is consistent with the GS. When p_e, the probability of user error, is set to a value greater than zero, the artificial user occasionally makes errors: with error probability p_e, it replies to a pairwise comparison with a relative ordering of the two requirements which is the opposite of that found in the GS. We used mutation rate = 10%. Population size was 50 for ALL (22, 27 and 24 for MON, FALL and ESC respectively).

Figure 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). 30 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 this is only partially known during the optimization iterations. Similar plots have been obtained for the 3 separate scenarios, FALL, ESC, MON.

Figure 2 shows the performance of IGA, compared to GA and RAN, by considering the difference between the prioritization produced by the algorithm and the GS. Such difference is measured both by disagreement with GS (top of Figure 2) and average distance from the requirements position in the GS (bottom of Figure 2). Figure 2 shows the results obtained for a particular scenario (MON) after eliciting 25, 50 and 100 pairwise comparisons. Similar plots have been obtained for the other two scenarios (FALL and ESC), as well as for ALL. The average distance from the requirements position in GS for ALL (after eliciting 25/50/100 pairs) is shown in Figure 3.

We can observe that the sanity check is passed (i.e., both GA and IGA outperform RAN by a large degree). We can also see how interaction improves the performance of non-interactive GA. While the improvement is minor with 25 elicited pairs, it gets quite substantial with 50 and it is definitely a major one with 100 elicited pairs. It should be noticed that state of the art algorithms for requirement prioritization, such as AHP, require exhaustive pairwise comparisons, which in the ALL scenario means 49 * 48 / 2 = 1176 comparisons (210 in MON). Hence, even as many as 100 elicited pairs represent a small fraction (around 50% for MON; 10% for ALL) of the comparisons elicited by AHP.

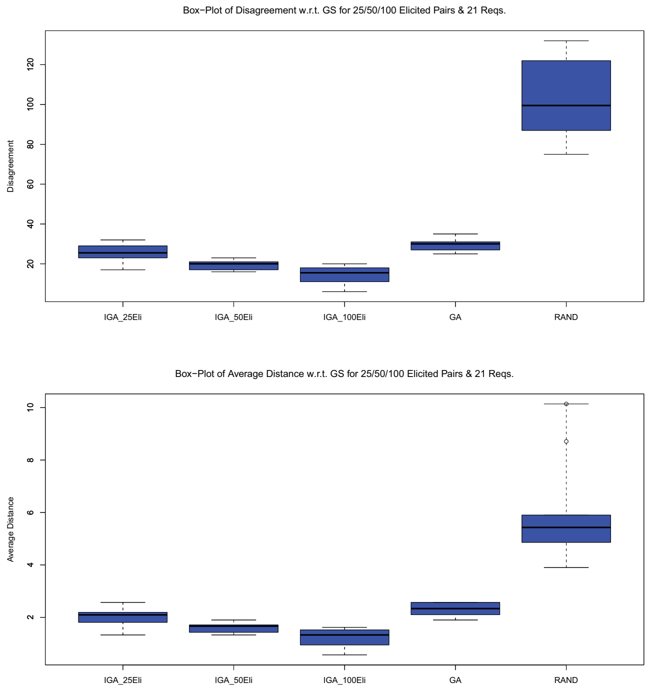

Figure 2. Disagreement with (top) and distance from (bottom) GS after eliciting 25/50/100 pairs from the user in the MON scenario

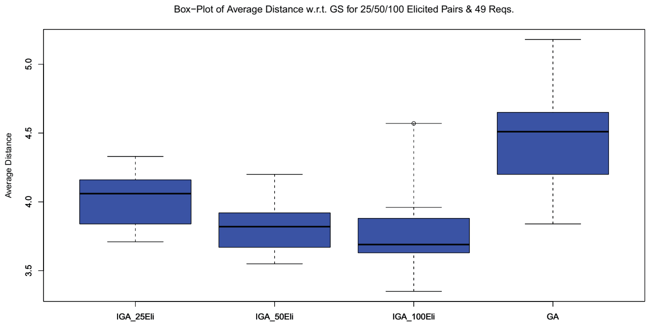

Figure 3. Average distance of each requirement position from the respective GS position after eliciting 25/50/100 pairs in the ALL scenario

In the MON scenario, the average distance of each requirement from the GS position is around 2 after eliciting 25 pairs, between 1 and 2 after 50 and around 1 after 100 pairs. For comparison, GA has an average distance between 2 and 3, while RAN is between 5 and 6 (see Figure 2, bottom). In the ALL scenario we get similar improvements (see

Disagreement	p-value	Distance	p-value
(IGA25, GA, RAN)	$<$ 2.2e-16	(IGA25, GA, RAN)	$<$ 2.2e-16
(IGA50, GA, RAN)	$<$ 2.2e-16	(IGA50, GA, RAN)	$<$ 2.2e-16
(IGA100, GA, RAN)	$<$ 2.2e-16	(IGA100, GA, RAN)	$<$ 2.2e-16

Table V
ANALYSIS OF VARIANCE (ANOVA) COMPARING IGA, GA AND RAN

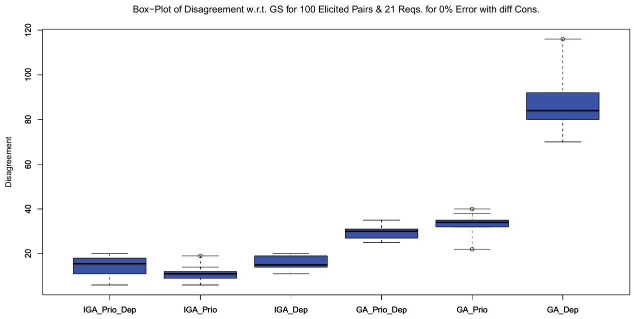

Figure 4. Performance of IGA and GA under different availability of precedence constraints (either Prio, Dep or both) in the MON scenario

Figure 3). IGA reduces the distance from the GS position, going from around 4.5 (GA) to 3.5–4 (IGA/100). RAN is definitely worse (14–17) in the ALL scenario. The final distance that we get after applying IGA indicates that the prioritization produced by our algorithm is quite close to the GS. Moreover, it consistently improves GA by a sensible degree and it outperforms RAN by a large degree. Statistical significance of the observed differences was tested using ANOVA (see Table V for the ALL scenario).

Figure 4 shows the final disagreement with the GS obtained respectively when (1) both Prio and Dep are available; (2) only Prio is available; and, (3) only Dep is available. We report the results for MON after eliciting 100 pairs, but similar plots have been obtained in all the other scenarios, including ALL. When Dep is dropped (compare IGA_Prio vs. GA_Prio), the improvement obtained by acquiring user knowledge through pairwise comparison is higher than the improvement obtained when both precedence graphs are available (compare IGA_Prio_Dep vs. GA_Prio_Dep). The effect of pair elicitation becomes particularly evident when only Dep is available (compare IGA_Dep vs. GA_Dep). In this case, the disagreement drops down from around 85 to 15, thanks to the information elicited from the user by means of the IGA algorithm. Overall, results indicate that IGA is particularly suitable and appropriate when scarce ranking attributes are associated with the requirements collected by the analysts. Applying IGA instead of GA in such cases improves the final results by a huge degree.

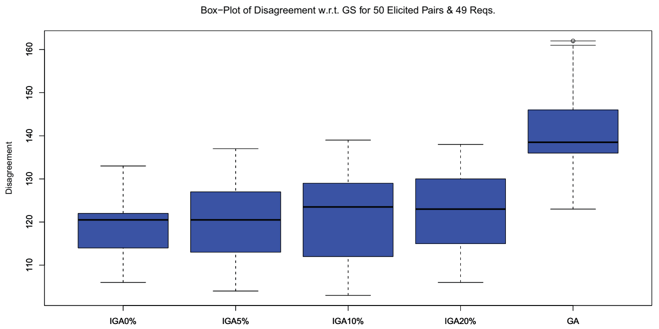

Figure 5. Performance of IGA at increasing user error rates in the ALL scenario

Figure 5 shows how the performance of the IGA algorithm degrade at increasing user error rates. We show the results obtained for ALL, but similar plots are available for the smaller, separate scenarios. As expected, the disagreement tends to increase as long as the user makes errors at increasing rate. However, such increase is not steep, indicating that the algorithm is quite robust with respect to the errors possibly affecting the user responses to pairwise comparisons. Even at a user error probability p_e as high as 20%, IGA keeps some margin of improvement over GA.

C. Discussion

Based on the data collected from our experiments (due to space limitations, we showed only a portion of them in this paper), we can answer positively to all our research questions. 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 with GS, both in terms of disagreement and of position distance, IGA outperforms substantially GA (and RAN), especially when a higher number of pairwise comparisons can be carried out (RQ2). The improvement of IGA over GA is even higher when limited ranking information is available a-priori (RQ3). Moreover, the behavior of the algorithm is robust with respect to the presence of elicitation errors committed by the user (RQ4).

AHP, which represents the state of the art for requirement prioritization, requires exhaustive pairwise comparison. This 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 zero 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, if we look at the average distance of each requirement from the position it has in the GS, we can see that such a distance is quite low (3.5–4). Hence, we think the cost/benefit trade off offered by IGA as compared to AHP is extremely interesting. With an elicitation effort reduced to 10% of the one required by AHP, IGA produces an approximate ordering which has a quite low average distance from the requirement positions in the GS. Of course, more empirical investigation is necessary to assess the actual, practical viability and acceptability of the trade off offered by IGA, as compared to AHP. We plan to conduct such empirical studies, involving real (vs. artificial) users, as part of our future work.

D. 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 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 addition to the complete one, and we considered the same set of requirements, but with different precedence constraints associated. This enlarges a bit the scope of generalizability 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 looked like an interpretable alternative. Another construct validity threat might be related with to simple user error model we used to simulate a user who occasionally makes errors. We will experiment with more sophisticated models in the future.

VI. Conclusions and future work

We have proposed an interactive genetic algorithm to collect pairwise information useful to prioritize the requirements for a software system. We have applied our algorithm to a real case study, consisting of a non trivial number of requirements, which makes AHP (the state of the art prioritization method) hardly applicable. Our approach scaled to the size of the considered case study and produced a result that outperforms GA (i.e., a genetic algorithm which optimizes satisfaction of the initial constraints, without gathering further constraints from the user). Specifically, by eliciting between 50 and 100 pairwise comparisons from the user it was possible to obtain a substantially better ordering of the prioritized requirements. Such gain gets amplified if initially poor or scarce ranking information is associated with the requirements. In fact, in such cases it is fundamental to ask the user for further ranking information, without overwhelming her/him. We achieved a good compromise between elicitation effort and performance of the algorithm. We verified also the robustness of the algorithm in the presence of user errors, which makes the algorithm applicable in contexts where the input from the user is only partially reliable.

In our future work we will conduct more experiments in alternative settings and on other case studies to corroborate our findings. We plan also to design and conduct an empirical study with human subjects in the role of requirement analysts, to test the approach in the field.

References

[1] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *Software IEEE*, vol. 14, no. 5, pp. 67–74, 1997.

[2] J. Karlsson, "Software Requirements Prioritizing," in *Proceedings of 2nd International Conference on Requirements Engineering (ICRE '96)* , April 1996, pp. 110–116.

[3] S. Sivzittian and B. Nuseibeh, "Linking the Selection of Requirements to Market Value: A Portfolio - Based Approach," in *REFSQ 2001*, 2001.

[4] H. P. In, D. Olson, and T. Rodgers, "Multi-criteria preference analysis for systematic requirements negotiation," in *COMPSAC 2002*, 2002, pp. 887–892.

[5] F. Moisiadis, "Prioritising software requirements," in *SERP 2002*, June 2002.

[6] T. L. Saaty and L. G. Vargas, *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Kluwer Academic, 2000.

[7] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach*. Addison-Wesley Longman Inc., 2000.

[8] K. E. Wiegers, *Software Requirements. Best Practices*. Microsoft Press, 1999.

[9] S. Lauesen, *Software requirements: styles and techniques*. Addison Wesley, 2002.

[10] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," in *RE 2005*, 2005, pp. 297–306.

[11] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[12] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *GECCO '07*. ACM, 2007, pp. 1129–1137.

[13] R. Andrich, F. Botto, V. Gower, C. Leonardi, O. Mayora, L. Pigini, V. Revolti, L. Sabatucci, A. Susi, and M. Zancanaro, "ACube: User-Centred and Goal-Oriented techniques," Fondazione Bruno Kessler - IRST, Tech. Rep., 2010.