Using an SMT Solver for Interactive Requirements Prioritization

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

ABSTRACT

The prioritization of requirements is a crucial activity in the early phases of the software development process. It consists of finding an order relation among requirements, considering several requirements characteristics, such as stakeholder preferences, technical constraints, implementation costs and user perceived value.

We propose an interactive approach to the problem of prioritization based on Satisfiability Modulo Theory (SMT) techniques and pairwise comparisons. Our approach resorts to interactive knowledge acquisition whenever the relative priority among requirements cannot be determined based on the available information. Synthesis of the final ranking is obtained via SMT constraint solving.

The approach has been evaluated on a set of requirements from a real healthcare project. Results show that it overcomes other interactive state-of-the art prioritization approaches in terms of effectiveness, efficiency and robustness to decision maker errors.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

Keywords

Requirements prioritization, constraint solving

1. INTRODUCTION

Requirements prioritization is the process of specifying a total or partial order relation on a set of requirements under analysis. This process is of paramount importance in the first phases of the software development process when the scheduling of the software releases is planned and requirements are assigned to the releases, considering the stakeholder expectations, implementation costs, time and technical constraints.

Several approaches to requirements prioritization have been proposed in the last few years [5, 6, 7, 8, 9, 10, 13, 15]. Some approaches [8, 9, 15] rely on the definition of models that weight the attributes of the requirements (e.g., development costs, value for users, implementation effort) *a priori*, without any reference to the instances of requirements under analysis. Other approaches [5, 6, 7, 10, 13] allow the decision maker to formulate the ordering *ex-post*, on the basis of the characteristics of the specific set of requirements that is considered, without relying on predefined schemes.

Among the prioritization techniques used in the latter category, the Analytical Hierarchy Process (AHP) [12] is widely referenced. It exploits pairwise comparisons to extract the user[1] knowledge with respect to the ranking of the requirements. The AHP method suffers scalability problems, since it defines the prioritization through user assessment of all the possible pairs of requirements. Hence, the number of pairwise comparisons asked to the user is quadratic with the number of requirements.

In order to address the scalability problem of AHP, some techniques have been proposed [2, 4, 14]. They sub-sample the exhaustive set of pairwise comparisons, while trying to achieve comparable performance as AHP. IAHP (Incomplete Analytic Hierarchy Process) [4] and IGA (Interactive Genetic Algorithm) [14] are two of the most successful attempts made in this direction.

In this paper we propose an *ex-post* approach based on the use of Satisfiability Modulo Theory (SMT) techniques for the synthesis of a ranking from the constraints. We rely on pairwise preference elicitation, to extract relevant knowledge from the user, only when the available knowledge does not allow the disambiguation of the relative priority of a set of requirements. We use an SMT solver to compose the elicited preferences with the relevant ordering criteria induced by the attributes describing the requirements and encoded as constraints. The final objective is that of minimizing the user decision-making effort, increasing as much as possible the accuracy of the final requirements ranking.

Specifically, pairs elicited from the user and initial constraints on the relative ordering of requirements define the set of constraints to be satisfied. Elicitation of stakeholders' preferences and optimization are conducted at the same time and influence each other. In fact, the set of stakeholder constraints is specified incrementally, based on the unresolved priorities under the available constraints. Elicited constraints are iteratively composed with the constraints

[1]The user of the prioritized requirements is the development team (system architect, requirements analysts, etc.) and should not be confused with the final user of the system under development.

coming from other sources.

We evaluated our SMT approach comparing it with other state-of-the-art interactive prioritization techniques (IAHP and IGA in particular). Results indicate that SMT substantially outperforms them, while keeping the user effort (in terms of number of elicited pairs) acceptable. Moreover, we compared interactive and non-interactive prioritization, observing an increased performance when interaction with the decision maker takes place. We also evaluated the robustness of our method with respect to decision maker's errors.

This paper is structured as follows: in Section 2 we describe the background for our work, Section 3 presents the proposed approach and algorithm. In Section 4 we comment a set of experimental evaluations on effectiveness, robustness and efficiency of the approach. The empirical assessment was conducted on a set of requirements from a real healthcare project. Relevant related works are discussed in Section 5. Conclusions and future work are presented in Section 6.

2. BACKGROUND

Here we briefly introduce two quite successful, state-of-the-art methods for interactive prioritization of requirements: Incomplete Analytic Hierarchy Process (IAHP) [4] and Interactive Genetic Algorithm (IGA) [14]. Both methods have the objective of synthesizing an approximation of the ranking of a set of N requirements, minimizing the effort associated with the interactive input required from the decision maker (consisting of pairwise comparisons).

2.1 Incomplete AHP

IAHP [4] is a variant of the Analytic Hierarchy Process (AHP) [12] 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).

In particular, AHP implements a pairwise comparison iterative process in which the decision maker is required to specify a preference as an integer value $p_{ij} \in [1 \dots 9]$, between two requirements R_i and R_j. The value can be mapped to a qualitative measure of the preference relation, as summarized in Table 1. After assigning a preference value to one of the two requirements (say, R_i) compared to the other one (R_j), the preference value for the least preferred requirement (R_j) is the reciprocal of p_{ij}, so, $p_{ji} = 1/p_{ij}$. When all the pairs have been elicited, a ranking is synthesized through the computation of the principal eigenvector of the matrix p_{ij}, the components of which determine the rank of each requirement.

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 more important than R_j
9	R_i Extremely more important than R_j
2, 4, 6, 8	For compromise between the above values

Table 1: A possible fundamental scale in the interval $[1 \dots 9]$ used with AHP

IAHP overcomes the AHP scalability problem by minimizing the number of pairs 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 to the decision maker in order to find a stable and good approximation of the target ranking, early before eliciting all the $N(N-1)/2$ pairs for the set of N requirements.

The high level steps of the IAHP process are: *(i)* the decision maker provides $N-1$ judgments which form a connected graph; *(ii)* using the available pairwise comparisons, the missing comparisons are estimated by taking the geometric mean of the intensities over a spanning tree (this results in a weight matrix); *(iii)* the derivatives of the weight matrix with respect to the missing matrix elements are computed and the next pairwise comparison to ask is determined based on such derivatives; *(iv)* if the stopping criteria are met (e.g., maximum number of elicitations or convergence metrics below threshold), then the computation stops and a rank is produced by calculating the principal eigenvector of the estimated matrix; otherwise the selected comparison is elicited and the algorithm iterates.

2.2 Interactive GA

In [14] we presented an Interactive Genetic Algorithm (IGA) to achieve the minimization of the disagreement between a total order of prioritized requirements and the various constraints coming from the domain knowledge, that are either encoded with the requirements or expressed iteratively by the user during the prioritization process.

In the approach we exploit the 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 by the genetic algorithm represents an alternative prioritization Pr_i of the N requirements. Two examples of such individuals are: $Pr_1 = \langle R_1, R_5, R_4, R_3, R_2 \rangle$, or $Pr_2 = \langle R_1, R_4, R_5, R_3, R_2 \rangle$. 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, in the form of preferences between pairs of requirements, so specifying if R_i is before or after R_j in the user ideal ranking, so as to make the fitness function landscape better suited for further minimization. In this way, knowledge about the constraints in the domain can help the method to minimize the amount of knowledge elicited from the decision maker, so decreasing her/his effort. The prioritization process terminates when a low disagreement is reached, a time out is reached or the allocated elicitation budget is over.

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 SMT solvers to achieve such a minimization, taking advantage of interactive input from the requirement engineer whenever the solver produces more than one prioritized list of requirements having the same disagreement with the available constraints. The prioritization process terminates when a unique solution at minimum disagreement is found or the maximum allowed number of pairwise comparisons is reached.

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	

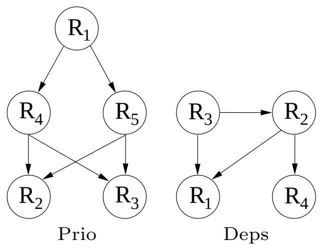


Table 2: Requirements with priority and dependencies

Let us consider the five requirements listed in Table 2. For each requirement we consider the priority expressed by the requirement engineer who collected them and the dependencies the requirement may have with other requirements. For conciseness, in Table 2 we omit other important elements of a requirement (e.g., the textual description). Constraints upon requirements can be represented by means of a *constraint* graph. In a constraint graph, an edge between two requirements indicates that, according to the related dependency or priority, 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 each constraint. When encoded in the input language of the SMT solver, such constraints will be retractable constraints which are given the weight labeling the edge (or 1 if no weight is given). An infinite weight is used for constraints that must necessarily hold in the final ordering of the requirements. These will be non-retractable constraints for the SMT solver.

At the right of Table 2, we show the constraint graph induced by the *priority* (Prio) property of the requirements and the constraint 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 constraint 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.

It is possible to encode the requirements prioritization problem as a MAX-SAT problem. An SMT solver will find an integer assignment which maximizes the weight of the retractable constraints that are satisfied by the solution (minimum cost of unsatisfied constraints). The encoding is described formally in the next section. Intuitively, it consists of an assignment of positions (integers between 1 and N, for N requirements) to the components of an integer array x of size N. The assignment must define a permutation of the positions, hence there should be no repetition of positions. This is easily encoded as a set of inequalities between pairs of positions ($x[i] \neq x[j] \quad \forall\ i \neq j$). The other constraints (from the constraint graphs) are encoded as inequalities (e.g., $x[1] < x[4]$ and $x[1] < x[5]$ for the two edges at the top of the constraint graph Prio). The solutions to the MAX-SAT problem produced by the SMT solver are all

```
(set-evidence! true)
(define x::(-> nat nat))
(define N::nat 5)

;; Only permutations of positions are allowed
(assert (forall (i::(subrange 1 N)) (and (>= (x i) 1) (<= (x i) N))))
(assert (forall (i::(subrange 1 (- N 1)))
          (forall (j::(subrange (+ i 1) N)) (/= (x i) (x j)))))

;; Prio
(assert+ (< (x 1) (x 4)) 1)
(assert+ (< (x 1) (x 5)) 1)
(assert+ (< (x 1) (x 2)) 1)
(assert+ (< (x 1) (x 3)) 1)
(assert+ (< (x 4) (x 2)) 1)
(assert+ (< (x 4) (x 3)) 1)
(assert+ (< (x 5) (x 2)) 1)
(assert+ (< (x 5) (x 3)) 1)

;;Deps
(assert+ (< (x 3) (x 2)) 1)
(assert+ (< (x 3) (x 1)) 1)
(assert+ (< (x 3) (x 4)) 1)
(assert+ (< (x 2) (x 1)) 1)
(assert+ (< (x 2) (x 4)) 1)

(max-sat)
```

Figure 1: Encoding of the constraints in Table 2 for the SMT solver Yices

requirement orderings that violate the minimum number of retractable constraints (e.g., Deps and Prio), or have minimum cost of violation, in case different weights are given to different constraints. The encoding of the requirement prioritization problem in Table 2 for the SMT solver Yices[2] is shown in Figure 1.

Id	Reqs	Disagree
Pr_1	$< R_1, R_5, R_4, R_3, R_2 >$	4
Pr_2	$< R_1, R_4, R_5, R_3, R_2 >$	4
Pr_3	$< R_1, R_5, R_3, R_4, R_2 >$	4
Pr_4	$< R_1, R_5, R_3, R_2, R_4 >$	4

Table 3: Prioritized requirements and related minimum disagreement

After encoding the constraint graphs in Table 2 as retractable assertions (`assert+` in Figure 1) and running an SMT solver (e.g., Yices) on them, we obtain the four prioritized lists of requirements shown in Table 3 (the solver must be run four times to obtain the complete set of prioritizations, each time negating the previously found solutions). These are all prioritized lists having the same, minimum cost of retracted assertions (i.e., 4). Such a cost is called *disagreement* and it represents the number of constraints that are not respected by the resulting requirement positions. The first two prioritizations Pr_1 and Pr_2 are in complete agreement with the constraint graph Prio, while they violate some of the edges of Dep (namely, (R_3, R_1), (R_3, R_2), (R_2, R_4) and the transitive constraint (R_3, R_4)). Pr_3 violates (R_4, R_3) from Prio and (R_2, R_4), (R_2, R_1), (R_3, R_1) from Dep. Pr_4 violates (R_4, R_2), (R_4, R_3) from Prio and (R_3, R_1), (R_2, R_1) from Dep.

In order to choose among the alternative prioritizations at minimum disagreement, we compare them and determine the pairs of requirements that are ordered differently in dif-

[2]http://yices.csl.sri.com/

Prios	Disagreement pairs
(Pr_1, Pr_2)	(R_4, R_5)
(Pr_1, Pr_3)	(R_3, R_4)
(Pr_1, Pr_4)	(R_3, R_4), (R_2, R_4)
(Pr_2, Pr_3)	(R_4, R_5), (R_3, R_4)
(Pr_2, Pr_4)	(R_4, R_5), (R_3, R_4) , (R_2, R_4)
(Pr_3, Pr_4)	(R_2, R_4)

Table 4: Pairwise comparisons to resolve ties

ferent prioritized lists. For example, if we compare Pr_1 and Pr_2 from Table 3, we can notice that they differ on the relative ordering of R_4 and R_5. We say that these two prioritizations are in disagreement with respect to the requirement pair (R_4, R_5). Table 4 shows the disagreement pairs for each comparison of prioritizations at minimum cost (taken from Table 3). For example, Pr_2 and Pr_3 differ on the relative ordering of two requirement pairs: (R_4, R_5) and (R_3, R_4). If we take the union of the requirement pairs on which the four minimum cost prioritizations differ, we get three pairs: (R_2, R_4), (R_3, R_4), (R_4, R_5) (see Table 4). These are the causes of the ties between the four prioritizations determined by the SMT solver. We can discriminate among such four prioritizations if we can decide on the precedence holding for the disagreement pairs. Hence, the information to be elicited from the user consists of pairwise comparisons between requirements that are ordered differently in equally scored prioritizations.

The user is requested to express a precedence relationship between each pair in the disagreement computed for prioritizations in a tie. Given a disagreement pair of requirements (e.g., R_4, R_5), the elicited ranking may state that one requirement should have precedence over the other one (e.g., $R_4 \to R_5$; or, $R_5 \to R_4$) or the user may answer *don't know*. In the first case an edge is introduced in the elicited constraint graph Eli, a third constraint graph that adds to Prio and Dep and is obtained by gathering interactive user input. In the second case no edge is introduced. In our running example, the user would be requested to compare (R_2, R_4), (R_3, R_4) 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_4, while Deps gives precedence to R_2, R_3. Hence, it is entirely justified to request additional user input, to resolve an inconsistency which is already present in the requirement documents, when considering different properties or 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 SMT solver to distinguish, e.g., between Pr_1 and Pr_2, whose only difference consist 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 constraint graph Eli (initially empty) is augmented with the new edges. The appearance of cycles in the elicited graph indicates the existence of contradictory information. This is managed automatically by the SMT solver, which will satisfy some constraints in the cycle and will retract some others.

When the new elicited constraint graph is available, the SMT solutions at minimum cost are recomputed. Such solutions are expected to be much more discriminative than the previous ones, thanks to the additional information gathered through interaction. Only pairs that actually make a difference in the optimal orderings are submitted to the user for assessment. After a number of iterations, the algorithm is expected to have successfully discriminated all prioritizations at minimum cost, thanks to the input elicited from the user, leading to the final selection of the prioritization with lowest disagreement w.r.t. all precedence graphs (including the elicited one).

3.1 Algorithm

In this section, we describe the interactive 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 and we formalize the SMT problem to be solved. We define disagreement 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 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 $dis(ord_1, ord_2)$.

The requirement prioritization problem can be formulated as the problem of assigning the integer values taken from the set of requirement positions $P = \{1, \ldots, N\}$, where N is the number of requirements, to the components of an integer array x, such that no two different array components hold the same value (i.e., the assignment defines a permutation of P):

$$\forall i \in P, j \in P : x_i \in P, x_j \in P, i \neq j \Rightarrow x_i \neq x_j \qquad (2)$$

Among all permutations defined by equation (2), we want to determine those that minimize the number of unsatisfied constraints. This is achieved by encoding the constraint graphs as retractable inequality constraints: $x_i < x_j$ whenever there is a path between R_i and R_j in the constraint graphs. The minimum number of such constraints that are retracted in the MAX-SAT solution found by the SMT solver gives the disagreement between the found solution and the constraint graphs. The SMT solver can be iterated so as to enumerate all solutions at minimum disagreement (i.e., at minimum constraint violation cost).

Algorithm 1 contains the pseudocode of the interactive 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.). Step 3 sets an important parameter of the algorithm. Namely, the maximum number of pairwise comparisons that can be reasonably requested to the user. The default value for it is 100.

The algorithm iterates until a unique solution to the prioritization problem is found or the maximum number of pairwise comparisons has been requested (see condition of *while* loop at step 5). In the body of the loop, the SMT solver is invoked at step 6. The solver receives in input a set of retractable assertions, associated with the constraints graphs (including the elicited constraint graph). Such graphs are turned into inequality assertions as explained above. In addition to such retractable assertions, the SMT solver adds

Algorithm 1 Compute prioritized requirements

```
Input  R: set of requirements
Input  $ord_1, ..., ord_k$: partial orders defining constraints
       upon R ($ord_i \subseteq R \times R$)
Output $\langle R_1, ..., R_n \rangle$: ordered list of requirements
 1: Solutions := $\emptyset$
 2: elicitedPairs := 0
 3: maxElicitedPairs := MAX_ELI_PAIRS (default = 100)
 4: eliOrd := $\emptyset$
 5: while $|Solutions| \neq 1 \wedge elicitedPairs < maxElicitedPairs$ do
 6:    Solutions := MAX_SAT($ord_1, ..., ord_k, eliOrd$)
 7:    if $|Solutions| \neq 1$ then
 8:       eliOrd := eliOrd $\cup$ elicit pairwise comparisons from
          user for disagreement pairs up to maxElicitedPairs
 9:       increment elicitedPairs by the number of elicited
          pairwise comparisons
10:    end if
11: end while
12: return $Pr_{min}$, a requirement list from Solutions with
    minimum disagreement
```

always the constraints expressed in equation (2) as non retractable assertions. The solver is repeatedly executed, so as to enumerate all solutions having the same minimum cost. Technically, this is achieved by adding an assertion that negates the previously found solution and asking for another solution to the new formulation of the problem. MAX_SAT return the set of all such minimum cost solutions.

If more than one solution is produced by the solver, the disagreement pairs for them is determined and the user is requested input to disambiguate them (steps 7-10). The user input is encoded as an additional constraint graph, *eliOrd*, which introduces new retractable assertions in the next iteration of the algorithm.

If the algorithm terminates because the size of *Solutions* is one, then the unique solution found is returned as Pr_{min}. If the algorithm terminates because the maximum number of elicitations has been reached, but no unique solution has been found, any randomly selected solution from the set *Solutions* is returned.

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 requirement ordering. 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.

4. EXPERIMENTAL RESULTS

We applied the IGA algorithm to prioritize the requirements of a real software system, as part of the project ACube (Ambient Aware Assistance) [1]. ACube is a large research project funded by the local government of the Autonomous Province of Trento, in Italy, aiming at designing a complex system to be deployed in nursing homes to support medical and assistance staff. In such context, an important activity 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 monitors the nursing home guests and operators without influencing their usual daily life activities. Through advanced automatic reasoning algorithms, the data acquired through the sensor network are used to promptly recognize emergency situations and to prevent possible threats for the guests themselves.

The user requirements analysis phase produced 60 user requirements, 49 technical requirements[3] and four macro-scenarios. Specifically, the 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: *"TR7: The system identifies the presence of a person in a given area"* or *"TR9: The system identifies the identity of the persons in the scene"*.

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 5: The four macro-scenarios and the number of technical requirements associated with them.

Table 5 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. Specifically, 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 technical dependencies between requirements and is a function that links a requirement to the set of other requirements it depends on.

Finally, the ACube software architect specified the *Gold Standard* (GS) prioritization for each of the four macro-scenarios. 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 (that we consider the ideal one).

[3]We consider here only functional requirements.

		SMT		SMT ($p_e = 5\%$)			SMT ($p_e = 10\%$)			SMT ($p_e = 20\%$)			IGA		IAHP	
Eli.pairs	Actual	Dis	AD	Err	Dis	AD	Err	Dis	AD	Err	Dis	AD	Dis	AD	Dis	AD
25	25	92	3.18	1	102	3.43	3	102	3.42	5	101	3.39	124	4.06	478	13.60
50	50	90	3.14	3	92	3.16	5	95	3.22	10	98	3.31	120	3.82	208	6.30
100	84	73	2.78	4	75	2.86	8	79	2.94	17	85	3.10	114	3.69	187	5.75

Table 6: Comparison table of disagreement and average distance among SMT, IGA & IAHP at different user error rates for ALL scenario

4.1 Research questions

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

RQ1 (Comparison) *Does the SMT-based method produce improved prioritizations compared to IAHP and IGA?*

We compare SMT-based interactive prioritization with the state of the art interactive requirement prioritization technique IAHP [4] and with our previous work IGA [14]. The output of SMT-based prioritization is compared with the output of the IAHP and IGA algorithms, at equal number of pairs elicited from the user. We vary such 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: 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.

RQ2 (Role of interaction) *Does SMT-based interactive prioritization produce improved prioritizations compared to non-interactive SMT-based prioritization?*

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 SMT before any pair is elicited from the user with the output of the same method after completing the elicitation of pairwise comparisons from the user to resolve ties.

RQ3 (Robustness) *Is the SMT-based method more robust than IAHP and IGA with respect to errors committed by the user during the elicitation of pairwise comparisons?*

In order to test the robustness of SMT in comparison with IAHP and IGA 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 SMT, IAHP and IGA 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%.

RQ4 (Performance) *What is the machine time necessary to execute the constraint solver during the prioritization process, compared to the execution time of the optimization algorithms involved in IAHP and IGA?*

This research question deals with the time performance of the three methods being compared: SMT, IAHP and IGA. Low execution time of the constraint solving/optimization step is fundamental to allow online usage of the proposed method. In fact, the time between an elicitation and the next one is determined by the execution time of the algorithm that performs the required constraint solving/optimization steps and then determines which pairwise comparisons are to be elicited next. Such time should be acceptable from the point of view of the decision maker who answers the pairwise comparison questions.

4.2 Results

We implemented the algorithm described in Section 3 in C++, using the C SMT solver library provided with the SMT tool Yices. IAHP and IGA have been implemented in Java. The software and the experimental data are available online at `http://selab.fbk.eu/smt-req-prio`. For space reasons, we hereby report only a subset of the experimental results. The interested reader can find the complete set of results in a technical report available online at `http://se.fbk.eu/en/techreps`.

Since the algorithms used in the experiments involve some degree of randomness at various steps (e.g., IGA selects between mutation and crossover operators randomly, according to a predefined probability), whenever not stated differently, results are based on 20 replications of each experiment. We report the average and the distribution of the data values (using boxplots). Statistical significance of the differences observed among the evaluated techniques was assessed by means of the ANOVA statistical test. 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.

Table 6 shows a summary of the main results obtained in the experiments. Details by research question are reported below. The results reported in Table 6 have been obtained for the ALL macro scenario with a different number of elicited pairs and variations in user error rate, p_e. We obtained similar tables for the other macro scenarios (i.e. MON, FALL, ESC). The first column represents the number of elicited pairs that was used as user knowledge. In the first two cases (i.e. maximum 25 & 50 pairwise comparisons) SMT made use of the maximum allowed number of elicited pairs. In the third case (i.e. maximum 100 pairwise comparisons) SMT elicited only 84 pairs and reached the minimized prioritization before consuming all allowed comparisons. Columns 3 and 4 shows the disagreement (Dis) and average distance (AD) with GS for each setting. Columns 5 to 7, 8 to 10 and 11 to 13 show the different outcomes for user error rate 5%, 10% and 20%, where for each case, the first column (Err) represents the number of pairs that were answered by the user wrongly. Columns 14-15 & 16-17 represent average Dis and AD for IGA and IAHP respectively. With 100 elicited pairs, SMT has a disagreement of 73 (with user error probability p_e equal to 0), 75 (p_e equal to 5%), 79 (p_e equal to 10%) and 85 (p_e equal to 20%), all of which are better than IGA (114) and IAHP (187) with no user error (p_e equal to 0%).

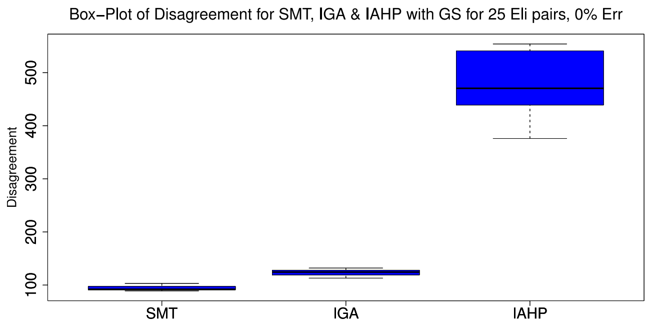
RQ1 *(Comparison).*

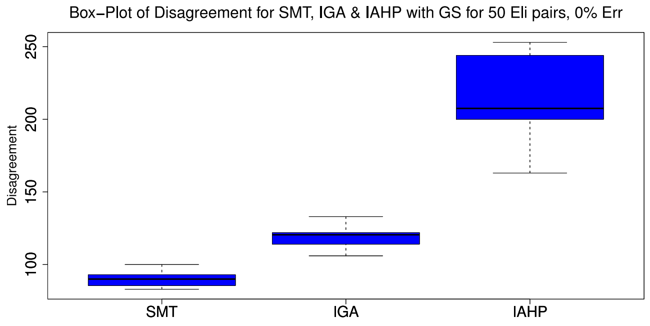
Figure 2 (top) shows the performance of the SMT-based approach, compared to IGA and IAHP, by considering the difference between the prioritization produced by the algorithm and the GS for the ALL macro scenario after eliciting 25 pairs. Similar plots have been obtained for the other scenarios (MON, FALL and ESC). We can observe that SMT outperforms both IGA and IAHP. After eliciting only 25 pairs from the user, SMT produced a prioritization with an average disagreement of 92, whereas IGA and IAHP have an average disagreement of 124 and 478 respectively. Even though IGA is more consistent than IAHP and produces much improvement with little user knowledge, the SMT-based approach produces an even better prioritization. In this case, the performance of IAHP is definitely worse than IGA, but SMT still shows improvements over IGA, even when little knowledge from the user is available.

Figure 2 (middle) shows the results obtained for the ALL macro scenario after eliciting 50 pairwise comparisons. SMT produced a prioritization with an average disagreement of 90, whereas IGA and IAHP have an average disagreement of 120 and 208 respectively. IGA outperformed IAHP, but the SMT-based approach further improves the prioritization produced by IGA. At this stage we are assuming the user is error-free, thus making no mistake in answering the pairwise comparisons.

Figure 2 (bottom) shows the results obtained with a maximum of 100 pairwise comparisons. SMT produced the best prioritization, with an average disagreement of 73, whereas IGA and IAHP have an average disagreement of 114 and 187 respectively. The degree of improvement of IGA with respect to IAHP and that of SMT with respect to IGA is almost equal and significantly large. Actually, as indicated in Table 6, the SMT-based interactive approach converged to the optimal solution after just 84 pairwise comparisons (out of the 100 available in this setting), hence reducing the elicitation effort with respect to the maximum allowed.

Statistical significance of the observed differences among





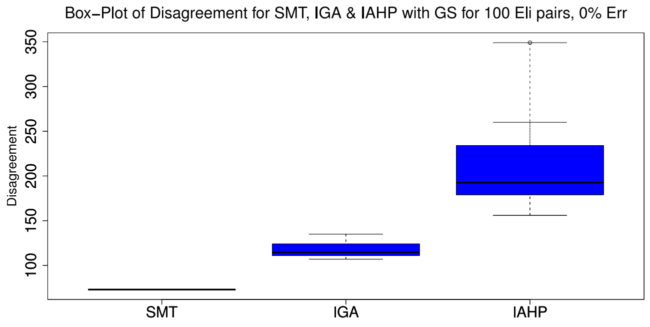


Figure 2: Disagreement with GS after eliciting at most 25 (top), 50 (middle) and 100 (bottom) pairs from the user in the ALL scenario

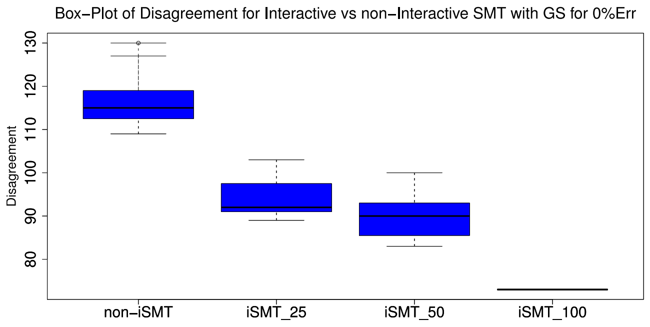
the three methods was tested using ANOVA; the results are summarized in Table 7. In particular, we report the p-values for the Disagreement and Avarage Distance (see Table 6 for a summary of the values for Avarage Distance) for the comparison of the three mehods for each of the three numbers of elicited pairs (25, 50, 100).

RQ2 *(Role of interaction).*

Figure 3 shows the comparison between the performance of interactive and non-interactive SMT-based prioritization, by considering the difference between the prioritization produced by the algorithm and the GS. In RQ2, we experimented with the same settings as RQ1, by assigning the value of *MAX_ELI_PAIRS* to 0, 25, 50, 100. In Figure 3 (top), non_iSMT represents the disagreement for non-interactive SMT, while iSMT_25, iSMT_50 and iSMT_100 represent the disagreement for interactive SMT while eliciting 25, 50 and 100 pairs from the user respectively. From Figure 3 it is clearly visible that while we are eliciting more pairs form the user, the fitness is getting better in terms of disagreement with GS. The non-interactive SMT-based approach produces

Methods	**Disagreement p-value**	**Aver. Distance p-value**
SMT25, IGA25, IAHP25	<2.2e-16	<2.2e-16
SMT50, IGA50, IAHP50	<2.2e-16	<2.2e-16
SMT100, IGA100, IAHP100	9.8e-05	10.31e-05

Table 7: Analysis of variance (ANOVA) comparing SMT, IGA and IAHP (while 0% error)



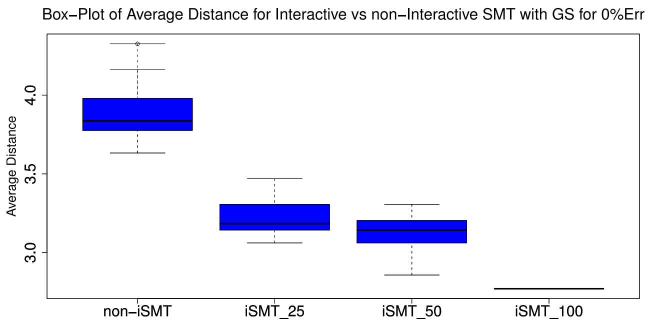


Figure 3: Disagreement (top) and average distance (bottom) with GS for interactive and non-interactive SMT after eliciting 25, 50, 100 pairs from the user in the ALL scenario

a prioritization having disagreement 115. After eliciting 25, 50 and 100 pairs form the user, the SMT-based interactive approach produces prioritizations having a disagreement of 92, 90 and 73 respectively. The improvement of interactive SMT, compared to non interactive SMT, after eliciting 100 pairs is almost 50%.

Figure 3 (bottom) shows the comparison on average distance between interactive and non-interactive SMT, by considering the positional difference between the prioritization produced by the algorithm and the GS. While eliciting more pairs form the user, the fitness is getting better in terms of average distance. Non-interactive SMT produces a prioritization with average distance of 3.84. After eliciting 25, 50 and 100 pairs form the user, interactive SMT produces a prioritization with average distance of 3.18, 3.14 and 2.78 respectively. The improvement of iSMT_100 over non_iSMT is around 50% also in terms of average distance.

RQ3 *(Robustness).*

In Figure 4, we show that even if the performance of the SMT-based interactive approach degrades at increasing user error rates, it always produces an improved prioritization compared to IGA and IAHP with a user who does not make any mistake. We show this for 50 elicited pairs, but similar results are available for a different number of elicited pairs. With no error SMT produces a prioritization with a disagreement of 90. At 5%, 10% and 20% error rate, the disagreement becomes 92, 95 and 98 respectively. On the other hand, even if the user makes no error (i.e. error-free user), the disagreement of the prioritizations produced by IGA and IAHP is 120 and 208. We obtained similar plots for the other three macro scenarios (MON, FALL, ESC).

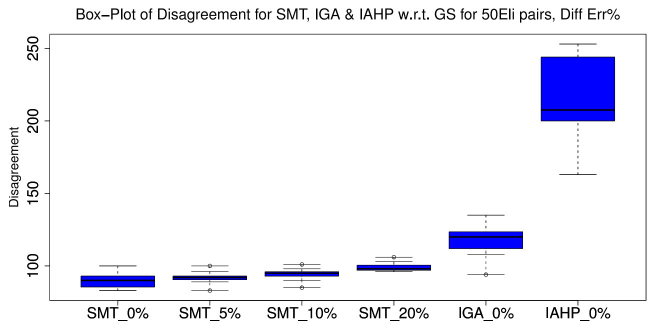


Figure 4: Robustness of SMT: Disagreement for SMT at different user error rates; IGA and IAHP with error-free user, after eliciting 50 pairs in the ALL scenario

In Figure 5 we compare the robustness of SMT with the two other approaches considered in this work, IGA and IAHP. Here we show the results obtained for ALL. Similar results have been obtained for the other macro scenarios. The SMT-based interactive approach is substantially more robust than the others. At increasing user error rates, the performance of IGA and IAHP is degrading to a higher degree. SMT has also a degrading performance at increased error rates, but it outperforms IGA and IAHP to a large extent. For an error rate of 5% SMT, IGA and IAHP have a disagreement of 92, 120 and 236 respectively. For a 10% error rate, these are 95, 119 and 277. For an error rate of 20%, the disagreements with GS become 98, 123 and 331 respectively. So, clearly the SMT-based interactive approach is more robust then IGA and IAHP.

RQ4 *(Performance).*

Pairs	SMT	IGA	IAHP
25	46s	640s	4s
50	51s	840s	6s
100	50s	1080s	128s

Table 8: Prioritization time (system time) required by SMT, IGA and IAHP for the ALL scenario

Table 8 shows the comparison among execution times of SMT, IGA and IAHP. All execution times in the table are obtained from the system time of each execution and are measured in seconds. The experiments were performed on an Intel Core2 Duo 2.10GHz machine with 3GB of memory, running the 32bit Windows Vista OS. The overall prioritization process time is much lower when using SMT as compared to IGA. IAHP is more efficient than SMT at 25 and 50 elicited pairs, but its performance seems to suffer a non-linear degradation when the number of elicitations in-

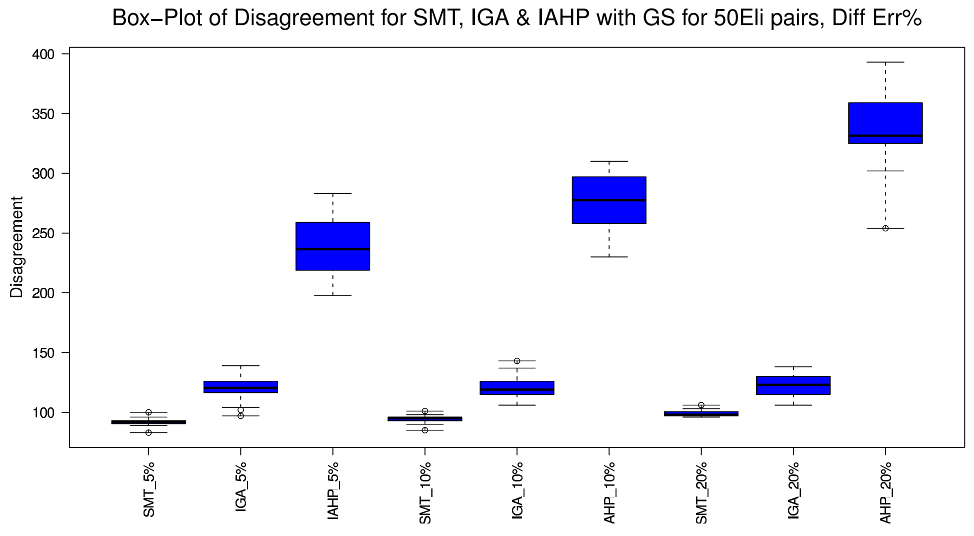


Figure 5: Robustness of SMT: Disagreement for SMT, IGA and IAHP after eliciting 50 pairs at different user error rates in the ALL scenario

creases. In fact, at 100 elicited pairs, the execution time of IAHP jumps from 4s to 128s. On the contrary, SMT seems to have little dependence on the number of elicitations. At 100 elicited pairs it remains almost constant and it is substantially inferior than IAHP.

If we approximate the time distance between two consecutive pairwise comparisons with the execution time per elicited pair, we observed that SMT leaves the user waiting for the next comparison for a time between 0.5s and 2s approximately. IGA introduces longer intervals between elicitations (between 10s and 15s), while IAHP has very short intervals (around 0.15s) when at most 25 or 50 pairs are elicited. This interval increases to 1.2s for 100 elicited pairs. We can conclude that the execution times of SMT and IAHP are compatible with an interactive, online usage of the algorithm, while IGA introduces longer waiting times, which may be annoying for the user. Moreover, IAHP seems to have a non linear time behavior with respect to the maximum number of elicited pairs.

4.3 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. When comparing the prioritizations produced by the considered algorithms with GS, both in terms of disagreement and of position distance, interactive SMT outperforms substantially IGA and IAHP (RQ1). Moreover, interactive SMT outperforms non-interactive SMT (RQ2), hence showing that user input plays a key role in producing a prioritization of requirements which gets close to the reference, gold standard one. The behavior of the algorithm is robust with respect to the presence of elicitation errors committed by the user (RQ3). Up to 20% user error rate, SMT outperforms IGA and IAHP, even when these are run at 0% user error rate. Finally, our proposed SMT-based interactive approach consumes much less system-time compared to IGA. This is true in comparison to IAHP only at a high number of elicited pairs (e.g., 100). The estimated time between consecutive pairwise elicitations (0.001s–0.95s) is compatible with online usage of the tool.

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 (2–3 positions). This indicates that the approximate solution produced by our method is close enough to the ideal solution to be usable in practice. Of course, further studies are needed to evaluate the acceptability of the approximate solution by the users.

In several cases, even though we set the maximum number of pair elicitations to 50 or 100, the SMT based approach used a lower number of pairs (i.e., 31 for MON, 34 for ESC, 42 for FALL and 84 for ALL), converging to a final requirement ordering which at the same time improves IGA and IAHP, and requires less user effort (pairwise comparisons). Hence, we think the cost/benefit trade off offered by the SMT-based interactive approach as compared to IGA and IAHP, as well as AHP, which requires exhaustive pairwise comparisons, is extremely advantageous. With an elicitation effort reduced to less than 10% of the one required by AHP (in the ALL scenario, 49×48 / $2 = 1{,}176$ pairwise comparisons), SMT produces an approximate ordering which has a quite low average distance from the requirement positions in the GS, substantially lower than IGA and IAHP. More empirical investigation is necessary to assess the actual, practical viability and acceptability of the trade off offered by SMT, as compared to AHP, IGA and IAHP. We plan to conduct such empirical studies, involving real (vs. artificial) users, as part of our future work.

4.4 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 three macro scenarios in addition to the complete one, which enlarges a bit the scope of generalizability of the results.

Other threats to validity concern 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. For the time performance, we measured system time. 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. 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 report a difference to exist between two datasets, we checked our claim using statistical tests.

5. RELATED WORKS

Several techniques are currently exploited in prioritization approaches. Many of them consist of assigning a rank to each requirement in a set according to a specific decision making criterion. 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* [9]. Other techniques consist of assigning each requirement to one specific class among a set of predefined priority classes, as for instance in *Numerical Assignment* [9, 15] and in *Top-ten requirements* [8].

Some approaches share a prioritization process that consists of a selection of one or more prioritization criteria including business goals and technical features, the acquisition of requirements priorities on the basis of the criteria, eliciting such priorities from the stakeholders, and the integration of the acquired orderings into the final one [6, 13, 5, 10, 11].

Among them, CBRank [2] 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). Differently from our method, CBRank has no way to encode constraints such as dependencies, which play a key role in requirements prioritization.

The Analytical Hierarchy Process (AHP) [12] can be considered the reference method for prioritizing alternatives. It adopts a pairwise comparison strategy, allowing to define the prioritization criteria through a priority assessment of all the possible pairs of requirements. As highlighted above, this method becomes impractical as soon as the number of requirements increases. Harker [4] proposed the method incomplete AHP (IAHP), to minimize the number of pairs elicited from the decision maker maintaining a good tradeoff between the precision of the final solution and the effort of the decision maker.

Among the methods exploiting search based techniques, in particular genetic algorithms, the EVOLVE method [3] supports continuous planning for incremental software development. This approach is based on an iterative optimization method supported by a genetic algorithm. Zhang et al. [16] focus on the specific Multi-Objective Next Release Problem (MONRP) and present the results of an empirical study on the suitability of weighted and Pareto optimal genetic algorithms (NSGA-II in particular), providing 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.

In our previous work [14], we proposed an approach based on Interactive Genetic Algorithm (IGA) to minimize the amount of knowledge, in terms of pairwise evaluations, that has to be elicited from the user. The present work represents a substantial advancement of IGA in terms of faster convergence, improved final solution, increased robustness and diminished user effort, as apparent from the experimental results.

6. CONCLUSIONS AND FUTURE WORK

In this paper we proposed an approach to the requirements prioritization problem relying on Satisfiability Modulo Theory techniques. The approach is interactive and is based on a pairwise preference elicitation process to extract relevant knowledge from the decision makers that is encoded as constraints. This knowledge is then combined with the knowledge related to the ordering criteria induced by requirements characteristics, such as implementation costs, values for the users, implementation dependencies, that are also encoded as constraints.

The aim is that of minimizing the user decision-making effort, in terms of number of pairwise comparison to be specified, increasing as much as possible the accuracy of the final requirements ranking.

To validate the approach we conducted experiments on a set of requirements from a real healthcare project. Results show that our approach overcomes other state-of-the-art interactive approaches such as IAHP and IGA, both in terms of disagreement between the ideal solution and the solution obtained via the methods and in terms of execution time. Moreover, we verified the importance of the interaction with the decision maker, which increases substantially the performance of the SMT approach with respect to a non interactive version of the same algorithm. The method is also robust with respect to user errors, maintaining good performance even when the percentage of errors is around 20%.

In our future work we plan to conduct empirical studies involving human subjects (requirement analysts) to assess the usability of the method and the acceptability of the approximate solution produced at the end of the pairwise elicitation.

7. REFERENCES

[1] 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. Technical report, Fondazione Bruno Kessler - IRST, 2010.

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

[3] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[4] P. T. Harker. Incomplete Pairwise Comparisons in the Analytic Hierarchy Process. *Math. Modelling*, 9(11):837 – 848, 1987.

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

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

[7] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997.

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

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

[10] F. Moisiadis. Prioritising software requirements. In *SERP 2002*, June 2002.

[11] A. Ngo-The and G. Ruhe. Requirements negotiation under incompleteness and uncertainty. In *Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering, SEKE*, pages 586–593, San Francisco Bay, CA, USA, July 2003.

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

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

[14] P. Tonella, A. Susi, and F. Palma. Using Interactive GA for Requirements Prioritization. In *2nd International Symposium on Search Based Software Engineering*, pages 57–66. IEEE, 2010.

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

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