Using Interactive Genetic Algorithm for Requirements Prioritization

(Uso di un Algoritmo Genetico Interattivo per la Priorizzazione dei Requisiti)

Relatore: 
Prof. Paolo Giorgini

Laureando: 
Francis Palma

Correlatore: 
Dott. Paolo Tonella
Dott. Angelo Susi

Anno Accademico 2009-10
Dedication

"Every big thing in world only comes true, when somebody does more than he has to do."

It is my great pleasure to dedicate this work to Dr. Hermann Gmeiner (1919-1986), an Austrian philanthropist, the founder father of SOS Children’s Villages and most importantly my beloved Grand father (in Bengali, Dadu).

Born to a big family of farmers in Vorarlberg (Austria), Dr. Hermann Gmeiner was a talented child and won scholarship to attend grammar school. His mother died while he was too young, his eldest sister Elsa took on the task of caring for the smallest of the children. Having experienced the horrors of war himself as a soldier in Russia, he was then confronted with the isolation and sufferings of the many war orphans & homeless children as a child welfare worker after the end of the World War-II. In his conviction that help can never be effective as long as the children have to grow up without a home of their own, he set about implementing his idea for SOS Children’s Villages.

With just 600 Austrian Schillings (approx. 40 US dollars) in his pocket, Dr. Hermann Gmeiner established the SOS Children’s Village Association in 1949, and in the same year the foundation stone was laid for the first SOS Children’s Village in Imst, in the Austrian state of Tyrol. His work with the children and development of the SOS Children’s Village organization kept Dr. Hermann Gmeiner so busy that finally decided to discontinue his medical degree course. In the following decades his life was inseparably linked with his commitment to a family-centred child-care concept based on the four pillars of a mother, a house, brothers & sisters and a village. He served as Village Director in Imst, organized the construction of further SOS Children’s Villages in Austria, and helped to set up SOS Children’s Villages in many other countries of Europe.

In 1960, SOS-Kinderdorf International was established in Strasbourg as the umbrella organization for SOS Children’s Villages with Dr. Hermann Gmeiner as the first president. In the following years, the activities of SOS Children’s Villages spread beyond Europe. The sensational, grain of rice campaign raised enough funds to permit the first non-European SOS Children’s Village to be built in Daegu, Korea in 1963 and SOS Children’s Villages on the American and African continents followed. By 1985 the result of Dr. Hermann Gmeiner’s work was a total of 233 SOS Children’s Villages in 85 countries. In recognition of his services to orphaned and abandoned children he received numerous awards and was nominated several times for the Nobel Peace Prize.

Dr. Hermann Gmeiner died in Innsbruck in 1986 and was buried at SOS Children’s Village Imst.

SOS Children’s Villages is currently active in 132 countries & territories. 438 SOS Children’s Villages & 346 SOS Youth Facilities provide more than 60,000 children and youths in need with a new home.
Acknowledgments

It is my pleasure to thank some people around me who made this thesis possible.

It is difficult to overstate my gratitude to my supervisor, Prof. Paolo Giorgini for the enthusiasm and inspiration, which were always there for me.

This thesis would not have been possible without guidance and support from the initial to the final level from Dott. Paolo Tonella. Throughout my working and thesis-writing period, he provided encouragement, sound advice, good teaching, good company and lots of good ideas. I would have been lost without him.

I am very specially thankful to Dott. Angelo Susi who was abundantly helpful and offered invaluable assistance and guided me in every possible way during my thesis works and writing.

I also want to let my gratefulness to Opera Universitariá, Provincia Autonoma di Trento, for the financial support they provided during my thesis works.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of this thesis works.
# Contents

List of Figures 4

List of Tables 6

1 Introduction 7

1.1 Background and Motivation ........................................ 7
1.1.1 The Problem ............................................. 8
1.1.2 The Business and IT Context ................................ 8
1.2 Goal of the work - benefits and results .......................... 8
1.2.1 Benefits ................................................ 9
1.2.2 Results ................................................. 9
1.3 Followed Approach ................................................ 9
1.4 Structure of the thesis ............................................ 10
1.5 Novel Contributions ................................................ 12

2 State of the Art 13

2.1 Pairwise Comparison based approaches ............................ 13
2.1.1 Analytic Hierarchy Process (AHP) ............................ 14
2.1.2 Incomplete Analytic Hierarchy Process (IAHP) ............... 17
2.1.3 Minimal Spanning Tree ....................................... 18
2.1.4 Bubble Sort .............................................. 19
2.1.5 Cost-Value Approach ........................................ 19
2.1.6 CBRank .................................................. 20
2.2 Non-Pairwise comparison based approaches ....................... 22
2.2.1 Nominal Scale ............................................ 22
2.2.2 Ordinal Scale ............................................ 24
2.2.3 Hundred Dollar Method ..................................... 25
2.3 Combined Techniques ............................................... 26
2.3.1 Planning Game ............................................ 26
2.4 Domain Knowledge based approaches ............................. 27
2.4.1 Priority Groups ........................................... 27
2.4.2 Genetic Algorithm (GA) .................................... 28
2.5 Summary: State of the Art ......................................... 30
3 Genetic Algorithm

3.1 Genetic Algorithm Operators ........................................... 32
  3.1.1 Crossover ......................................................... 32
  3.1.2 Mutation ......................................................... 36
  3.1.3 Selection ......................................................... 37

3.2 Genetic Parameters .................................................... 40
  3.2.1 Population Size .................................................. 40
  3.2.2 Probability of Crossover ........................................ 40
  3.2.3 Probability of Mutation ......................................... 40

3.3 Genetic Algorithm pseudo-code ..................................... 41

4 Algorithm Description .................................................. 42

4.1 Overall Proposed Process .......................................... 42
  4.1.1 High Level Overview ............................................. 42

4.2 Operators: IGA ....................................................... 49
  4.2.1 Crossover Operator ............................................. 50
  4.2.2 Mutation Operator ............................................. 50
  4.2.3 Selection Operator ............................................. 51

4.3 The Algorithm ....................................................... 52

4.4 Algorithm Steps Details ............................................. 54

5 The Case Study .......................................................... 56

5.1 ACube Project: ....................................................... 56
  5.1.1 Innovative Aspects .............................................. 56
  5.1.2 Technology ...................................................... 57
  5.1.3 Technical Objectives ........................................... 57
  5.1.4 System requirement details ................................... 57

6 Experiments & Results .................................................. 60

6.1 Experiments .......................................................... 60
  6.1.1 Design .......................................................... 60

6.2 Research Questions ................................................... 63
  6.2.1 Research Question 1: [Convergence] ........................ 63
  6.2.2 Research Question 2: [Role of Interaction] ................ 67
  6.2.3 Research Question 3: [Role of Initial Precedence Constraints] 75
  6.2.4 Research Question 4: [Robustness] .......................... 81

6.3 General Discussion on Results ..................................... 86
  6.3.1 Comparison with AHP ......................................... 86

6.4 Threats to Validity ................................................... 86

7 Conclusion & Future Works .......................................... 88

7.1 Conclusion .......................................................... 88

7.2 Future Works ........................................................ 90
List of Figures

1.1 Chapter dependency of the thesis work ........................................... 11
2.1 Basic steps of the CBRanking prioritization process ............................ 21
3.1 Example: An One-Point Crossover Operator (1-PX) .......................... 33
3.2 Example: A 3-Point Crossover Operator (3-PX) .............................. 34
3.3 Example: A Uniform Crossover Operator ....................................... 34
3.4 Example: A Diagonal Crossover Operator ...................................... 35
3.5 Example: An Arithmetical Crossover Operator ............................... 35
3.6 Example: A Mid-Point Crossover Operator .................................... 36
3.7 Example: A Simplex Crossover Operator ...................................... 36
3.8 Example: A simple mutation method in Genetic Algorithm .................. 37
3.9 An Example: Roulette Wheel Selection ....................................... 38
3.10 An Example: Situation before ranking (graph of fitnesses) ................ 39
3.11 An Example: Situation after ranking (graph of order numbers) ........... 39
4.1 A high level overview of the proposed IGA approach .......................... 43
4.2 Prio: The priority graph for the requirements ................................. 44
4.3 Deps: The dependency graph for the requirements .......................... 45
4.4 Eli: The elicited pairs, representing edges in the third precedence graph ... 48
4.5 Example: Cut-head/fill-in-tail operator of the proposed IGA ............... 50
4.6 Example: Requirement-pair-swap operator of the proposed IGA .......... 51
4.7 Example: Tournament Selection operator of the proposed IGA ............ 52
6.1 Disagreement of the best individual in each population across generations in ALL scenario & 25 Elicited pair (with timeOut 600 seconds) ....................... 65
6.2 Disagreement of the best individual in each population across generations in ALL scenario & 100 Elicited pair (with timeOut 1080 seconds) .................. 66
6.3 Box-Plot of Disagreement with respect to GS for 25, 50 & 100 Elicited pairs for ALL (49 req) ................................................................. 69
6.4 Box-Plot of Average Distance with respect to GS for 25, 50 & 100 Elicited pairs for ALL (49 req) ................................................................. 70
6.5 Box-Plot of Disagreement with respect to GS for 25, 50 & 100 Elicited pairs for MON (21 req) ................................................................. 72
6.6 Box-Plot of Average Distance with respect to GS for 25, 50 & 100 Elicited pairs for MON (21 req) ................................................................. 73
6.7 Plots of minimum final disagreement against GS in each run for IGA, GA & RAND 74
6.8 Box-Plot of Disagreement w.r.t. GS for 100 Elicited Pairs & 49 Reqs. for 0% Error with different constraints ................................. 77
6.9 Box-Plot of Disagreement w.r.t. GS for 50 Elicited Pairs & 21 Reqs. for 0% Error with different constraints ........................................... 78
6.10 Box-Plot of Disagreement w.r.t. GS for 50 Elicited Pairs & 26 Reqs. for 0% Error with different constraints ........................................... 79
6.11 Box-Plot of Disagreement w.r.t. GS for 100 Elicited Pairs & 26 Reqs. for 0% Error with different constraints ........................................... 80
6.12 Box-Plot of Disagreement w.r.t. GS for 50 Elicited Pairs & 49 Reqs. with error rates 5% to 20% ............................................................. 83
6.13 Box-Plot of Disagreement w.r.t. GS for 100 Elicited Pairs & 49 Reqs. with error rates 5% to 20% ............................................................. 84
6.14 Box-Plot of Disagreement w.r.t. GS for 100 Elicited Pairs & 26 Reqs. with error rates 5% to 20% ............................................................. 85

A.1 An event for MON macro scenario in ACube .................................. 91
A.2 An event for ESC macro scenario in ACube .................................. 94
A.3 An event for FAL macro scenario in ACube .................................. 96

B.1 Class diagram for IGA Implementation of the algorithm part (in Java) .... 103
B.2 Class diagram for IGA Implementation of the analysis part (in Java) .... 106
B.3 Program model for IGA Implementation: with classes & functions ........ 112
# List of Tables

2.1 Fundamental scales used for AHP (Karlsson and Ryan, 1997, p.70) . . . . . . . . 16  
2.2 Comparison of methods that require pairwise comparison . . . . . . . . . . . . . . . 30  
2.3 Comparison of methods that do not require pairwise comparison: part-I . . . . . . 31  
2.4 Comparison of methods that do not require pairwise comparison: part-II . . . . . 31  
3.1 Crossover: The Classes & subdivisions . . . . . . . . . . . . . . . . . . . . . . . . 33  
4.1 Nine requirements with priorities and dependencies . . . . . . . . . . . . . . . . . . 44  
4.2 Prioritized candidate requirements and their related disagreements . . . . . . . . 46  
4.3 Pairwise comparison to resolve ties between individuals . . . . . . . . . . . . . . . 48  
5.1 Four macro-scenarios and the number of technical associated requirements . . . . 58  
6.1 Levels for the factors used in IGA . . . . . . . . . . . . . . . . . . . . . . . . . . . 61  
6.2 Numerical representation of user error model used for simulation in IGA . . . . . 62  
6.3 Different degrees of domain knowledge i.e. precedence graphs in three experiments 62  
6.4 Experimental settings for the RQ1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 64  
6.5 Experimental settings for the RQ2 (part-I) . . . . . . . . . . . . . . . . . . . . . 68  
6.6 Experimental settings for the RQ2 (part-II) . . . . . . . . . . . . . . . . . . . . . 68  
6.7 Analysis of variance (ANOVA) comparing Disagreement of IGA, GA and RAND 71  
6.8 Analysis of variance (ANOVA) comparing Average Distance of IGA, GA and RAND 71  
6.9 Experimental settings for the RQ3: (part-I) . . . . . . . . . . . . . . . . . . . . . 76  
6.10 Experimental settings for the RQ3: (part-II) . . . . . . . . . . . . . . . . . . . . . 76  
6.11 Experimental settings for the RQ4 . . . . . . . . . . . . . . . . . . . . . . . . . . 82  
A.1 Dependency table for MON scenario . . . . . . . . . . . . . . . . . . . . . . . . . 93  
A.2 Dependency table for ESC scenario . . . . . . . . . . . . . . . . . . . . . . . . . . 95  
A.3 Dependency table for FALL scenario . . . . . . . . . . . . . . . . . . . . . . . . . 98  
A.4 Dependency table for ALL scenario . . . . . . . . . . . . . . . . . . . . . . . . . 101
Chapter 1

Introduction

1.1 Background and Motivation

In the software development cycle, during the requirement analysis and requirement specification phase, requirements prioritization plays an important role. During planning for a set of requirements to implement in the successive system releases, several considerations are made based on: available budget, time constraints, stakeholder’s expectations and possible run-time technical constraints. The process of requirements prioritization can be viewed as the process of producing a total order on the set of requirements. The process of prioritizing requirements can be designed as an *a priori* or as an *a posteriori* process. In the first type of process, before doing the specification of the requirements, preferences are formulated via models, such as ranking criteria expressed in requirements attributes and values. In the *a posteriori* approach, the ranking of the set of requirements is formulated on the basis of the characteristics present in the elicited requirements, for example via the process of pair-wise comparison (i.e. based experiences and knowledge), in which formal or informal reasoning allows selecting one from a pair of requirements.

In our proposed method, based on the Interactive Genetic Algorithm (IGA) search-based technique and on pair-wise preference elicitation, we focus on the *a posteriori* approach. The reason for choosing the *a posteriori* approach is to extract relevant knowledge from the user, while considering also relevant ordering criteria captured from the description of the requirements attributes. The final objective is to minimize the *user decision-making* effort, hence increasing the accuracy of the final requirements ordering.

Several approaches for prioritizing requirements have been proposed in the last few decades [5], [20], [26], [32], [40]. Among the prioritization techniques used in these methods, *Analytical Hierarchy Process (AHP)* [40] exploits a pair-wise comparison techniques to extract the stakeholder (decision maker’s) knowledge w.r.t. to the ranking of the requirements. AHP defines the prioritization criteria through a priority assessment of all the possible pairs of requirements. In general, *a posteriori* prioritization methods, including AHP, face huge challenges regarding *Scalability*. 
1.1.1 The Problem

The order of the implementation of requirements in a project affects the quality and value of the delivered system and of course the end-users’ satisfaction. The main goal of requirements prioritization is to rank the requirements in an order which trades-off between users’ needs and development constraints. Some typical development or technical constraints can be: dependencies among the requirements, limited resources available and proper allocation of that limited resources etc. The goal of our research is to prioritize requirements in a suitable order that satisfies both end users’ and development constraints.

Requirements prioritization aims at identifying the most important requirements for a system. This is a crucial step to decide which requirements to implement (and deliver) in each release, taking into account resources, budget and time constraints as well as customer expectations. Existing methods, to help this decision making process have limitations w.r.t. their usability with a large number of requirements and other technical issues like requirements interdependencies. We developed an interaction based framework for requirements prioritization, called Interactive Genetic Algorithm (IGA), which exploits domain and specially user knowledge to overcome scalability problem that is very common in state-of-the-art approaches. We performed a wide range of experimentations with a real case study (ACube, see Chapter 5).

1.1.2 The Business and IT Context

Requirements prioritization techniques are used in a to-be-built system for determining which candidate requirements should be included in a certain release. Care is also taken, so that the most important or high risk requirements are implemented first. In the initial phase of the software development life cycle, requirements are elicited from stakeholders and analyzed. In the release planning, requirements are then prioritized and selected for a release; hence, one of the key steps in release planning is requirements prioritization. The prioritization process helps the project team to resolve conflicts, plan for stepped deliveries and to make necessary trade-off.

1.2 Goal of the work - benefits and results

Given typically high customer expectations, but limited time-frame and resources, requirement engineers need to make sure the product contains the most important functionalities. Developers do not always know which requirements are most important to the customers, and customers cannot judge the cost and technical constraints associated with the requirements. So, through our work, we try to do a trade-off among process performance of the prioritization
1.2.1 Benefits

With our work, different personnel will be benefited in several ways, directly or indirectly. The primary and first beneficiaries are the end users or customers, whose satisfaction is the ultimate goal in any project. The requirement engineers will be supported by the tool that implements the IGA framework for prioritizing requirements. On the other hand, the researchers in this arena can also perform further investigation based on or using the ideas of our new methodology for requirements prioritization. So, the direct benefited parties are stakeholders (i.e. end users), requirement engineers and researchers.

1.2.2 Results

The concrete results that we deliver are the performance difference between the Interactive Genetic Algorithm (IGA) and the other non-interactive approaches (i.e. state-of-the-art approaches), hence saving time and effort. So, the benefits are propagated towards all the stakeholders of the new methodology. We applied our method to a large system (ACube), with the requirements set extracted by the system analyst of ACube, and then we prioritized them using both IGA approach and other non-interactive approaches. In all the empirical assessments, IGA outperforms those approaches (see Chapter 6).

1.3 Followed Approach

Often requirement analysts possess relevant knowledge about the relative importance and attributes of requirements. We use an Interactive Genetic Algorithm (IGA) to produce a prioritized final requirements ordering which complies with user priorities, satisfies the technical constraints and takes into account the relative preferences elicited from the analyst. On a real case study (ACube, see Chapter 5), we show that this approach improves non-interactive optimization that does not include the facility of elicited preferences, and that IGA can handle a number of requirements which is rather problematic for the state-of-the-art techniques.

Our solution belongs to the class of methods that is principally based on pair-wise comparison and exploits an IGA approach to minimize the number of pairs to be elicited from the stakeholder, whereas other state-of-the-art approaches do not have this advantage of extended interaction. Elicited pairs and initial constraints on the relative ordering of requirements define the complete fitness function. This function consists both 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 during the evolutionary process, certainly they influence each other and a more specific uncommon characteristic of IGA is that the fitness function is constructed incrementally, being only partially known or even null at the beginning. Thus convergence is not trivially ensured by the optimization process. We have evaluated our IGA algorithm against a real case study, with a high number of requirements, which makes state-of-the-art techniques impractical decision-making or prioritizing approaches. Our results indicate that IGA converges and improves the performance of GA (without interaction) by a considerable amount, in terms of disagreement with the Gold Standard (see Section 4.1), while keeping the user efforts (number of elicited pairs) within an acceptable upper limit.

1.4 Structure of the thesis

In this work, we first point out the problem we are dealing with in Chapter 1. Then we describe a list of state-of-the-art methods for prioritizing requirements and finally at the end of the chapter, we present a summarized framework to compare all the presented methods. The use of a Genetic Algorithm gives an important functional contribution to IGA, so in a separate chapter, we give a detailed description about Genetic Algorithm (Chapter 3). In Chapter 4, we present our approach and the proposed algorithm. We introduce the ACube case study in Chapter 5; the project from which we extract the requirements set and the macro scenarios for our empirical assessment.

In Chapter 6, one of the most important part is presented, devoted to Experiments & Results. Chapter 7 includes conclusion and future work. In Appendix A, we provide a more details about the requirements of the ACube project. Finally, the implementation level detail, using class diagrams and functions definitions, is presented in Appendix B as an extended reference.
Figure 1.1: Chapter dependency of the thesis work

Figure 1.1 is the pictorial presentation of the chapter dependencies. Here, an arrow from A to
B denotes that A is dependent on B or referring information from B.

1.5 Novel Contributions

A research paper was derived from this work and was presented at SSBSE 2010 by Dott. Angelo Susi.

Paolo Tonella¹, Angelo Susi¹, and Francis Palma², "Using Interactive GA for Requirements Prioritization" in 2nd International Symposium on Search Based Software Engineering 2010. ¹Fondazione Bruno Kessler, Software Engineering Research Unit; ²Department of Inf. Eng. and Computer Science, University of Trento, (see also [43])
Chapter 2

State of the Art

Requirements Prioritization is part of Requirement Engineering; whereas requirement engineering is a branch of Software Engineering. Over the years, several approaches for requirement prioritization have been proposed by different SE researchers. Most of the state-of-the-art techniques are focused on minimizing effort and time, and maximizing the benefits through an optimized final ordering of the requirements, as an outcome of the prioritization process. State-of-the-art approaches usually mediate to negotiate cost and effort, considering existing or available resources. We present here some such state-of-the-art approaches [5], [20], [26], [32], [40] to describe their positive aspects as well as shortcomings in terms of performance and flexibility.

We can classify the approaches mainly into three classes.

- Approaches using User Knowledge either performing pairwise comparison or not; i.e. AHP, IAHP, Bubble sort etc.
- Approaches using Domain Knowledge; i.e. GA.
- Approaches using Both; i.e. CBRank.

A detailed classification of the approaches is presented in Sections [2.1], [2.2], [2.3] and [2.4].

2.1 Pairwise Comparison based approaches

Pairwise Comparison is widely used and accepted methodology as a requirements prioritization process in which a pair of requirement is presented to the user or expert; then she/he decides the order between them in an ordinal scale or assuming a relative value or even in a very simple way stating one is more important than the other. The outcome of the user’s decision can be used as an edge of graph (in IAHP) or as a cell value of a matrix (in AHP) or as a simple entry in a prioritized listing. Among many pairwise comparison based approaches, Analytic Hierarchy Process in Section 2.1.1, Incomplete Analytic Hierarchy Process in Section 2.1.2,
Hundred Dollar Method in Section 2.2.3, Minimal Spanning Tree in Section 2.1.3, Bubble Sort in Section 2.1.4 and Cost-Value Approach in Section 2.1.5 are discussed briefly.

### 2.1.1 Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) was proposed firstly by Saaty. It is one of the most accepted decision making methods [40]. Using AHP to prioritize software requirements involves comparing all unique pairs of requirements to determine which of the two is of higher priority, and to what extent. In a software project, comprising N user requirements, \(\frac{N \times (N-1)}{2}\) pair-wise comparisons are asked to the decision maker. On one hand, AHP is a demanding method due to the dramatically increasing number of required pair-wise comparisons when the number of requirements grows; on the other hand, AHP is very trustworthy since the huge amount of redundancy in the pair-wise comparisons makes the process fairly insensitive to judgmental errors. Another advantage is that the resulting priorities are relative and based on a ratio scale, which allows for useful assessments of requirements.

**Hierarchy & Criteria:** The first step in AHP is to model the problem as a hierarchy. A hierarchy is a layering process where each element (except the top one) is subordinate to other elements. Similarly, for a complex decision problem, it is suitable to use a hierarchy to integrate large amounts of information. Building this structure means forming a better picture of the problem as a whole. Hierarchy consists of an overall goal, a group of options or alternatives for reaching the goal, and a group of factors or criteria that relate the alternatives to the goal. The criteria can be further broken down into sub-criteria.

For example, having a problem of comparing N alternatives according to C criteria, the total pair-wise comparisons to be made in the first phase is,

\[
\text{Total Comparison} = C \times (C-1) + C \times \frac{N \times (N-1)}{2} \tag{2.1}
\]

To simplify the above equation, let's discuss an example. Suppose, a person needs to buy a car. He has three criteria (C = Style, Reliability, Fuel Economy) to select the best one from different alternatives (N = Civic, Saturn, Escort, Clio). First he prioritizes the criteria through pairwise comparison (i.e. C×(C-1) comparisons). Then for each criteria, he will have a prioritized order of alternatives also through pairwise comparison (i.e. \(C \times (N \times (N-1))/2\) comparisons). All these operations are performed through matrix calculations. Finally, through the matrix multiplication between the ranking of the criterion (C×1 matrix) and the ranking of each alternative according to those criterion (N×C matrix), the final prioritization will be found (resulting matrix N×1).

Prioritizing software requirements using AHP involves all three stages of a prioritizing session (for a comprehensive description of AHP, see [40]):
1. In initialization step: outline all unique pairs of requirements.

2. In execution step: compare all outlined pairs of requirements using the scale in Table 2.1.

3. In presentation step: use the *Averaging Over Normalized Columns* method (based on the pair-wise comparisons) to estimate the relative priority of each requirement. Calculate the consistency ratio of the pair-wise comparisons using methods provided by AHP. The consistency ratio is an indicator of the reliability of the resulting priorities, and thus also an estimate of the judgmental errors in the pair-wise comparisons [24].

To make a decision one needs various kinds of knowledge, information, and technical details. These basically concern the following: [42]

- details about the problem for which a decision is needed
- the people or actors involved
- their objectives and policies
- the influences affecting the outcomes and
- the time horizons, scenarios, and constraints

Decision making can also be considered as a process that involves the following steps: [42]

1. Structure a problem with a model that shows the problem’s key
2. Elicit judgments that reflect knowledge, feelings, or emotions
3. Represent those judgments with meaningful numbers
4. Use these numbers to calculate the priorities of the elements
5. Synthesize the results to determine an overall outcome
6. Analyze sensitivity to changes in judgment

*Judgments and Comparisons:* A judgment or comparison is the numerical representation of a relationship between two elements that shares a common parent. The set of all such judgments can be represented in a square matrix in which the set of elements is compared within itself. Each judgment represents the dominance of an element in the column on the left over an element in the row on top. It reflects the answers to two questions: which of the two elements is more important with respect to a higher level criterion and how strongly, using the 1-9 scale shown in Table 2.1. It is important to note that the lowest element is always used as the unit and the highest one is a multiple of that unit. From all the paired comparisons, we calculate the priorities and exhibit them on the upper right of the matrix. For a set of N elements in a matrix one needs $\frac{N \times (N-1)}{2}$ comparisons because there are N 1’s on the diagonal for comparing elements with themselves and of the remaining judgments, half are reciprocals. Thus we have
Table 2.1: Fundamental scales used for AHP (Karlsson and Ryan, 1997, p.70)

<table>
<thead>
<tr>
<th>Intensity of Importance</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Equal importance</td>
<td>Two activities contribute equally to the objective</td>
</tr>
<tr>
<td>3</td>
<td>Moderate importance</td>
<td>Experience and judgment slightly favor one activity over another</td>
</tr>
<tr>
<td>5</td>
<td>Strong importance</td>
<td>Experience and judgment strongly favor one activity over another</td>
</tr>
<tr>
<td>7</td>
<td>Very strong or demonstrated importance</td>
<td>An activity is favored very strongly over another; its dominance demonstrated in practice</td>
</tr>
<tr>
<td>9</td>
<td>Extreme importance</td>
<td>The evidence favoring one activity over another is of the highest possible order of affirmation</td>
</tr>
<tr>
<td>2, 4, 6, 8</td>
<td>For compromise between the above values</td>
<td>Sometimes one needs to interpolate a compromise judgment numerically because there is no good word to describe it</td>
</tr>
</tbody>
</table>

The foundation of the AHP is a set of axioms that delimits the scope of the problem environment (Saaty 1986). It is based on the well-defined mathematical structure of consistent matrices and their associated right eigenvector’s ability to generate true or approximate weights, Merkin (1979). The AHP methodology compares criteria, or alternatives with respect to a criterion, in a natural, pair-wise mode. To do so, AHP uses a fundamental scale of absolute numbers that has been proven in practice and validated by physical and decision problem experiments. The fundamental scale has been shown to be a scale that captures individual preferences with respect to quantitative and qualitative attributes just as well or better than other scales (Saaty 1980, 1994) [16].

The Benefits of Analytic Decision Making: Many excellent decision makers do not rely on a theory to make their decisions. Are their good decisions accidental or are there implicit logical principles that guide them in the process of making a decision? There can arise a question if these principles are complete and consistent or not. Analytic decision making is of tremendous value, but it must be simple and accessible to the target user [42].

Problems with Analytic Decision Making: Assume that an individual has expressed preference among a set of alternatives, and that as a result, he or she has developed a ranking for them. There is a concern, that individual’s preferences and the resulting rank order of the alternatives is affected if alternatives are added to the set or deleted from it and if no criteria are added or deleted. Mathematically, the number and quality of newly added alternatives are known to affect preference among the original alternatives. Most people make each decision separately, and they are not very concerned with rank reversal unless they are forced for some...
reason to refer to their earlier conclusions. According to Saaty, it is essential to understand and
to deal with this phenomenon [42].

Scalability is another problem for AHP. If the number of criteria is high and the number
of alternatives is also high, the decision making process becomes complex and results in an
enormous number of comparisons, which is not only tiresome but also inefficient from the user’s
perspective.

2.1.2 Incomplete Analytic Hierarchy Process (IAHP)

As described in Section 2.1.1, the Analytical Hierarchy Process (AHP) was developed in
the 1970s by T. L. Saaty [40] and over the years, it has been proven to be a very effective
decision-analysis tool. The two advantages which the AHP has over other multi-criteria
methods are: (i) ease of use and (ii) the ability to handle inconsistencies in judgments. The
experience with AHP supports Saaty’s [40] claim that pairwise comparisons are somewhat
natural i.e. individuals or groups quickly become comfortable with the pairwise comparison
mechanism and find it easy to use. By not forcing consistency of preferences, AHP leads to a
useful and usable decision-analysis and decision making tool [20].

The major drawback stays in individual or group decision process requiring significant
amount of effort to complete all pair-wise comparisons. If we have 9 alternatives and 5 criteria,
the total comparisons will be 190. In real-life problems these number of alternatives and criteria
can be higher. This concern is handled by an almost functionally equivalent approach that
requires less effort, called Incomplete Analytic Hierarchy Process (IAHP).

As P. T. Harker described it in [20], the steps used in the Incomplete Pairwise Compari-
son method are:

- Step 0: The decision maker provides N-1 judgments which form a connected graph if there
  are N alternatives.
- Step 1: Using the completed pairwise comparisons, derive the missing comparison by
taking the geometric mean of intensities of a sample of random spanning trees. Calculate
the weight matrix.
- Step 2: Calculate the derivatives of the weight matrix with respect to the missing matrix
  elements and select the next question by means of some mathematical derivations.
- Step 3: If this question meets the appropriate stopping criteria then stop; else elicit this
  comparison and return to step 1.

There are three possible ways by which a decision maker can stop the comparison process:

- The decision maker decides whether or not to continue with the further questioning; or,
• If the maximum absolute difference in the attribute weights from one question to another is \( \leq \alpha \% \), where \( \alpha \) is a given constant; or,
• The comparisons will continue to be made until one is sure that the ordinal rank will not be reversed.

To explain the last stopping criteria, we can state that the weights are cardinal rankings of the alternatives, which creates an ordinal ranking for them as well. But if the decision maker answers too many or more questions, the cardinal ranking in weights may be slightly altered but the ordinal ranking would remain the same. So, the third stopping rule states that the next question is going to be asked only if the ordinal ranking could be reversed. Using IAHP, substantial effort (i.e. time) saving can be achieved w.r.t. classical AHP but still inefficient to handle large set of requirements.

2.1.3 Minimal Spanning Tree

*Minimal Spanning Tree* is another prioritization technique which was introduced by Karlsson (1998) [25]. The idea of the minimal spanning tree method is that if the decisions are perfectly consistent, the redundancy won’t exist and in this case the number of comparisons will reduce to only \( N-1 \) comparisons (\( N := \) number of requirements). A minimal spanning tree constructs unique pairs of requirements. It is a directed graph which is minimally connected.

Using the minimal spanning tree approach involves all three stages of a prioritizing session:

1. In initialization step: outline \( N-1 \) unique pairs of requirements so that a minimal spanning tree can be constructed.
2. In execution step: compare all outlined pairs of requirements using the scale (there is no fixed or standard scale).
3. In presentation step: compute the missing intensities of importance by taking the geometric mean of the existing intensities of all possible ways in which they are connected. Then use AHP as usual [24].

As, only \( N-1 \) comparisons would be enough to calculate the relative intensity of the remaining comparisons. This implies that the least effort required by a decision maker is to create a minimal spanning tree (i.e. a directed graph) [24]. Minimal spanning tree can reduce the number of pairwise comparisons dramatically compared with AHP. However, the ability to handle inconsistent judgments is low.
2.1.4 Bubble Sort

Bubble sort was introduced by Aho, Hopcroft and Ullman (1983) [2]. It is a method for sorting elements using pairwise comparison. Karlsson (1998) [25] introduced this technique to the requirements prioritization area for ranking purpose. The idea of the bubble sort method for sorting requirements is that the users compare two requirements at a time and swap them if the two requirements are in the wrong order. The comparisons continue until no more swaps are needed. The result of bubble sort is a list of ranked requirements. The average and worst case complexity for bubble sort is $O(N^2)$.

Using the bubble sort approach involves the following stages of a prioritizing session:

1. In initialization step: outline the requirements in a list.

2. In execution step: start to compare the requirements at the top with the requirement at the position below the top. If the requirement in the above position is considered of higher priority, swap their positions. Continue unless $N \times (N - 1)$ comparisons are reached.

3. In presentation step: outline the sorted list. The result of the process is a list where the original order of the requirements has changed. The least important requirement is at the top of the list, and the most important requirement is at the bottom of the list. The result of bubble sort are requirements ranked according to their priority on an ordinal scale [24].

Bubble sort is one of the simplest and most basic methods for sorting elements with respect to a criterion i.e. according to their priorities. Interestingly, bubble sort is closely related to AHP. As with AHP, the required number of pair-wise comparisons in bubble sort is $\frac{N \times (N - 1)}{2}$. But, the decision maker has only to determine which of the two requirements is of higher priority, not to what extent [24].

2.1.5 Cost-Value Approach

Karlsson and Ryan (1997) [26] introduced an approach which is called the Cost-Value Approach for prioritizing requirements. The basic idea of this approach is that each individual requirement is determined on two aspects: the value to the users and the cost of implementing the requirements. It uses the AHP technique to compare requirements pair-wise according to the relative values and costs.

A process for prioritizing software requirements must be simple and fast and yield accurate and trustworthy results. If both of these conditions are not met, the process is unlikely to be used in commercial software systems development. According to the group of Shoji Shiba and his colleagues, there are three main factors in stakeholder satisfaction: Quality, Cost, and Delivery. For a software system to succeed, quality must be maximized, cost minimized, and time-to-delivery be as short as possible. The cost-value approach prioritizes requirements
according to their relative value and cost. Based on this information, software managers can make decisions such as which requirements to be excluded from the first release to keep the time-to-market at minimum.

Here, quality can be interpreted as candidate requirement’s potential contribution to customer satisfaction with the resulting system and cost is the cost of successfully implementing the candidate requirement. In practice, software developers often calculate costs purely in terms of money. However, Joachim Karlsson and Kevin Ryan found that prioritizing based on relative rather than absolute assignments is faster, more accurate, and more trustworthy [25, 26].

There are five steps to prioritizing requirements using the Cost-Value Approach.

1. Requirements engineers carefully review candidate requirements for completeness and to ensure that they are stated in an unambiguous way.

2. Customers and users apply the AHP pairwise comparison method to assess the relative value of the candidate requirements.

3. Experienced software engineers use AHP pairwise comparison to estimate the relative cost of implementing each candidate requirement.

4. A software engineer uses AHP to calculate each candidate requirement’s relative value and implementation cost, and plots these on a cost-value diagram.

5. The stakeholders use the cost-value diagram as a conceptual map for analyzing and discussing the candidate requirements. Based on this discussion, software managers prioritize the requirements and decide which will actually be implemented. They can also use the information to develop strategies for release planning [26].

Empirical studies performed by Karlsson and Ryan [26] showed that the Cost-Value Approach is time consuming.

2.1.6 CBRank

The Case-Based Ranking (CBRank) technique [6] exploits a machine learning algorithm to guide the elicitation of user preferences during the prioritization process. The framework is based on an iterative process that can handle single and multiple decision makers (stakeholders or users) and different criteria (both business goals and technical parameters). The main input to the process is the collection of requirements that have to be ranked and the final output is an approximation of the target ranking.

The Pair Sampling action is an automatic procedure which selects a pair (or a sample of pairs) of requirements on the basis of a predefined selection policy which may take into account information on the currently available rankings. The user performs the evaluation of
the requirements pairs, by iterating the following steps till all the pairs in the sample have been evaluated: select a pair from the sample; evaluate the relative importance of the requirements in the pair. That is, given a pair of requirements, the user is asked to specify which one is the more important requirement between them with respect to the given criterion. Differently from AHP, here there is no range of values, the preference is strict. The output of this step is a set of ordered pairs.

A high level CBRank action process is presented below to show how it performs:

![Diagram of the CBRanking prioritization process](image)

Figure 2.1: Basic steps of the CBRanking prioritization process

If the ranking produced by the ranking learning activity can be considered a good approximation (e.g. the error measure exploited in the technique is low) it is given in output, otherwise it becomes the input to a further iteration of the process [37].

The CBRank method is supported by a web-based tool named **SCORE (Supporting Case-Based**
Oriented Rank Elicitation) [5] which allows for a distributed use of the framework, to support the pair-wise priority elicitation by distributed stakeholders. The system supports the whole evaluation process. In particular, SCORE presents the user an agenda of comparisons. The user can analyze each one of the pairs specifying the preferred requirement in the pairs, by indicating which one of the requirements is more important than the other. Finally, once all the evaluations have been performed, the system computes the rank and, in the case of a further iteration, it presents to the user the set of new pairs of requirements to be evaluated.

CBRank adopts a preference elicitation process that combines sets of preferences elicited from human decision makers with the 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) [43]. So, it can be summarized that CBRank uses both user knowledge and domain knowledge along with pairwise comparisons as shown in Figure 2.1.

Even though it is better w.r.t other state-of-the-art approaches and more flexible to use (i.e. not tiresome), it has still the problem of scalability, which was not completely handled in the proposed method.

2.2 Non-Pairwise comparison based approaches

Non-pairwise comparison is another main stream methodological class for prioritizing requirements using a formal technique. Non-pairwise comparison can be categorized in two classes: Nominal Scale and Ordinal Scale as described in Section 2.2.1 and 2.2.2 respectively.

2.2.1 Nominal Scale

Nominal Scale is the lowest measurement level that can be used (from a statistical point of view). A nominal scale, as the name implies, is simply some placing of data into categories, without any order or structure. In research activities a yes/no scale is nominal. It has no order and there is no distance between yes and no. Thus, nominal measurement consists of assigning items to groups or categories. No quantitative information is conveyed and no ordering of the items is implied. Nominal scales are therefore qualitative rather than quantitative. Religious preference, race and sex are all examples of nominal scales.

For nominal scale methods, requirements are assigned to different priority groups, with all requirements in one priority group being of equal priority. The nominal scale is the most primitive of the four scale types and includes some kind of categorization or classification. All objects are grouped into subgroups and each subgroup is assigned a certain name or number. Requirements grouped according to which sub systems they concern is an example of nominal
classification. The only statistics to be gathered on this scale is frequency, i.e. the number of objects in each group. The mode can be calculated, but not the median or mean [30].

Numerical Assignment

Numerical Assignment is mentioned by a number of studies such as Berander and Andrews (2005) [8], Bradner (1997) [11], IEEE-STD 830-1998 (1998) [1], Karlsson, Host and Regnell (2006) [27], Leffingwell and Widrig (2000) [29] and Sommerville and Sawyer (1997) [23]. It is a simple requirements prioritization technique based on grouping requirements into different priority groups. The results of numerical assignment are on a nominal scale. All requirements contained in one priority group represent equal priority. No further information shows that one requirement is of higher or lower priority than another requirement within one priority group.

The Numerical Assignment technique is based on the principle that each requirement is assigned a symbol representing the requirement’s perceived importance. Several variants based on the Numerical Assignment technique exist, e.g. classifying requirements as mandatory, desirable or inessential [25]. Another way to classify requirements is to divide them into essential, conditional or optional requirements, as suggested by IEEE [1]. Furthermore, it would be possible to give each requirement a number e.g. between 1 and 5, where requirements with a 5 are the most important ones [25]. Classifying requirements according to Numerical Assignment does not give us information about the relation between the requirements in each class, thus several requirements may appear equally valuable [30].

MoScoW

MoScoW is a kind of numerical assignment and it is mentioned by DSDM Consortium (2009) [13], Hatton (2007, 2008) [38], [39] and Tudor and Walter (2006) [14]. MoScoW currently incorporates the methodology called Dynamic Systems Development Method into the software development. The idea of MoScoW is that it groups all requirements into four priority groups MUST have, SHOULD have, COULD have, and WON’T have.

- **MUST have** means that requirements in this group must be contained in the project. Failure to deliver these requirements means the entire project would be a failure.

- **SHOULD have** means that the project would be nice if it contains the requirements in this group.

- **COULD have** also means that the project would be nice if it contains these requirements. But these requirements are less important than the requirements in the **SHOULD have** group.

- **WON’T have** is like a wish list. It means that the requirements in this group are good requirements but they will not be implemented in the current stage. They may be implemented in the next release.
The results of MoScow are on a nominal scale and all requirements contained in one priority
group represent equal priority. Further information is unavailable about whether one require-
ment is of higher/lower priority than another within the same priority group.

2.2.2 Ordinal Scale

The simplest Ordinal Scale is a ranking. When a market researcher asks us to rank 5 types of beer from most flavorful to least flavorful, he/she is actually asking us to create an ordinal scale of preference. There is no objective distance between any two points on this subjective scale. To one the top beer may be far superior to the second preferred beer, but to others, with the same top and second beer, the distance may be subjectively small. Measurements with ordinal scales are ordered in the sense that higher numbers represent higher values.

Ordinal scale methods result in an ordered list of requirements. The ordinal scale can be used to enhance the nominal scale with information about the ordering of classes or categories. This is the case in Numerical Assignment in Section 2.2.1, when each requirement is classified according to its value and assigned to e.g. the mandatory, desirable, or inessential group. Priorities can also be measured using numbers such as 1, 2 or 3 where the requirements with highest priority are assigned 1. In addition, requirements within the groups can be ranked so that an ordered list of requirements is received. This scheme is also used for the Planning Game described in Section 2.3.1. The numbers associated with the requirements represent ranking only, so arithmetic operations, such as addition and multiplication, have no meaning [30].

Simple Ranking

Ranking elements or a collection of items is a quite intuitive notion for most people as it is a very common solution approach to some daily life problems. Berander and Andrews (2005) [8] and Hatton (2008) [39] mention Simple Ranking in the way that N (N:= total number of requirements for a customer) requirements are simply ranked from integer 1 to N, with the most important requirement ranked as 1 and the least important requirement as N. This is a common requirements prioritization technique based on an ordinal scale.

Binary Search Tree

Another method for sorting elements, mentioned by Aho (1983) [2] is Binary Search Tree. A binary search tree is a tree in which each node contains at most two children. Karlsson (1998) [24] introduce this technique to the requirements prioritization area for ranking requirements. The idea of the binary search tree method for ranking requirements is that each node represents a requirement, all requirements placed in the left subtree of a node are of lower priority than the node priority, and all requirements placed in the right subtree of a node are of higher priority.
than that node priority. For using this method, one first chooses one requirement to be the top node. Then one unsorted requirement to compare with the top node is selected. If that requirement is of lower priority than the top node, the left subtree is searched, but if that requirement is of higher priority than the top node, the right subtree is searched. The process is repeated until no further node needs to be compared and at that time the requirement can be inserted into the right position. The average complexity for binary search tree is $O(N \times \log(N))$.

Prioritizing $N$ software requirements using the binary search tree approach involves constructing a binary search tree consisting of $N$ nodes. Using the binary search tree approach involves all three stages of a prioritizing session:

1. In initialization step: outline the candidate requirements
2. In execution step: select the requirements one at a time and create a binary search tree
3. In presentation step: do the inorder traversal the binary search tree and add them to a list. The requirements having the lowest priority then come first in the list. Finally, print the list [24].

Since the average path length from the root to a leaf in a binary search tree is $O(\log(N))$, inserting a requirement into a binary search tree takes on the average $O(\log(N))$ time. Consequently, inserting all $N$ requirements into a binary search tree takes on the average $O(N \times \log(N))$ time. The requirements are ranked on an ordinal scale [24].

Simple Ranking (Section 2.2.2), Bubble Sort (Section 2.1.4) and Binary Search Tree methods are all used for ranking requirements. Simple Ranking (Section 2.2.2) method is quite intuitive for people, bubble sort and binary search tree methods seem harder for people to use to rank requirements. One question which may arise is that if people can do simple ranking easily, why are bubble sort and binary search tree methods needed? The answer is that when there are a fairly small number of requirements needed to be prioritized, simple ranking seems easy for people to perform. But as the number of requirements increases, people may have difficulty remembering all the requirements. Psychology research (reviewed by Miller (1956) [33]) shows that people have difficulty remembering more than seven (plus or minus two) elements. Hatton (2007) [38] said that it would be difficult and probably incorrect for people to use the Simple Ranking method to rank 15 or more elements. If a large number of requirements need to be ranked, in order to get a high degree of accuracy, Bubble Sort and Binary Search Tree seem more suitable than Simple Ranking.

### 2.2.3 Hundred Dollar Method

Hundred Dollar Method (also known as *Cumulative Voting*) which is mentioned by Berander and Andrews (2005) [8] and Hatton (2008) [39] is an alternative technique for prioritizing requirements. The idea of the hundred dollar method is that each stakeholder is asked to
assume having $100 to distribute over the requirements. The result is presented on a ratio scale. The ratio scale result can provide the information on how much one requirement is more or less important than another one.

Also, sometimes during the prioritization process, for large number of requirements, the person doing the prioritization can miscalculate, and the sum may become greater or less than one hundred (Berander and Wohlin 2004) [10]. However use of automated tools, which can keep the count, can avoid this situation.

This method takes longer to perform and contains less user confidence than MoSCoW and Simple Ranking described in Section 2.2.1 and 2.2.2 respectively, but it is still relatively easy to use. When dealing with relatively large numbers of requirements, Berander and Svahnberg (2008) [9] argued that the stakeholders may lose the overview as the number of requirements increases when using Hundred Dollar Method. This prioritization technique is complex in terms of sophistication and fine in terms of granularity.

One shortcoming of this technique is that stakeholders might put all their units on their one or more favorite requirements which other stakeholders do not prioritize as highly, thus biasing the prioritization process. This can be avoided by limiting the number of units to be put on a single requirement. This results in forcing the stakeholders not to prioritize according to their own priorities (Leffingwell and Widrig (2000) [29]).

2.3 Combined Techniques

Combining Technique approaches combine two or more methods to have a better prioritization performance.

2.3.1 Planning Game

Beck (1999) [7] introduced a prioritization technique, called Planning Game, which is basically based on a combination of two prioritization techniques i.e. Numerical Assignment (Section 2.2.1) and Simple Ranking (Section 2.2.2). Planning Game is mostly used in agile projects. Requirements are first prioritized into three groups (a) those without which the system will not function, (b) those that are less essential but provide significant business value and (c) those that would be nice to have. After assigning the requirements into three groups, requirements are simply ranked in each group.

The requirements prioritization in an XP (Extreme Programming) organization is carried out in the Planning game, where the scope of the next release is determined by combining business priorities and technical estimates [7]. In practice, marketing people or customers sort the requirements by value as: (a) necessary; (b) significant business value providers; and (c)
nice-to-have.

Then developers sort the requirements by risk into another three: (a) estimate precisely; (b) estimate reasonably well; and (c) not estimate at all.

Marketing people choose a set of cards, either by setting a release date and choosing cards based on their estimates, or by choosing the cards and calculating the release date. An addition to the Planning game is to sort the cards within the considered criterion. The method takes advantage of card-based prioritization, using a reasonable amount of time and producing a list of sorted requirements [31].

2.4 Domain Knowledge based approaches

While prioritizing requirements, sometimes domain knowledge plays an important role to choose from two alternatives to find out the better order. Several classes of domain knowledge can be there to be considered as a part of decision making process i.e. the inter-dependency of the requirements while implementing, their priorities (e.g. importance or urgency level) and cost etc. From these domain knowledge, additional knowledge can be inferred to make a choice if appropriate knowledge is absent at any certain decision points. There are several methods for prioritizing requirement that take advantage of domain knowledge: Priority Groups (Section 2.4.1) and Genetic Algorithm (GA) (Section 2.4.2).

2.4.1 Priority Groups

In some software development projects, one set of requirements can be of more importance than another. One way to reduce the required effort is therefore not to compare the requirements in these distinct sets. The method initiates the prioritizing process by dividing the requirements into separate groups based on a rough prioritization. Subsequently, the groups are ranked internally (optional) by using a suitable approach for ordering the requirement e.g. using AHP [24].

The primary gain with this method is that engineers do not have to compare high priority requirements with low priority requirements, since they are placed in different groups. The actual choice of the number of groups depends on the situation and the knowledge of the people performing the prioritization. A simple strategy would be to use three distinct groups: low, medium and high-priority. It may even be the case that the high-priority requirements must be implemented, and hence there is no need to prioritize them. In the same way the low-priority requirements may perhaps be postponed to a later release.

According to Joachim Karlsson [24], the following three steps are involved while using the Priority Groups approach:
1. In initialization step: outline the candidate requirements

2. In execution step: put each of the requirements into one of the three groups. In groups with more than one requirement, create three new subgroups and put the requirements into these groups. Continue to apply this process recursively to all groups.

3. In presentation step: just read the requirements from left to right.

To ensure that the ordering of the requirements is correct, the lowest ranked requirement in one group can be compared with the highest ranked requirement in the next group. This could be done to ensure that the tail of one group has higher priority than the head of the following group. This comparison between tail and head in the groups must continue until the requirements are in the correct order. This is one way of minimizing the risk of ending up with the requirements in the wrong order. Thus, Priority Groups approach can be divided into two possible approaches: grouping without tail-head comparison and grouping with tail-head comparison [24].

2.4.2 Genetic Algorithm (GA)

Genetic Algorithms (GAs) were developed by Prof. John Holland and his students at the University of Michigan during 1960-70s [21]. Genetic algorithms (GAs) are computer programs that mimic the processes of biological evolution in order to solve problems and to model evolutionary systems [34]. Essentially, a method of breeding computer programs and solutions to optimization or search problems by means of simulated evolution. Processes based on natural selection, crossover, and mutation are repeatedly applied to a population of binary strings which represent potential solutions. Over time, the number of above-average individuals increases, and highly-fit building blocks are combined from several fit individuals to find good solutions to the problem at hand.

Several significant discussions on GA and related environmental approaches & parameters has been presented in [18, 17, 22]. Genetic algorithms have been used in science and engineering as adaptive algorithms for solving practical problems and as computational models of natural evolutionary systems. The descriptions of applications and modeling projects stretch beyond the strict boundaries of computer science to include dynamical systems theory, game theory, molecular biology, ecology, evolutionary biology, and population genetics [34].

Some applications of genetic algorithms include optimization, automatic programming, machine learning, economic model and ecological modeling etc. Genetic Algorithm has also been applied to the problem of Requirements Prioritization, yielding reasonable results with less effort and acceptable scaled solution w.r.t. the other state-of-the-art approaches. To achieve our research objective we performed a parallel experimental comparison of GA with our proposed new methodology, from which we had a substantially positive result. Genetic Algorithms are
stochastic methods that can be used to solve a broad class of optimization problems including allocation and prioritization problems [32].

**Properties of GAs (in requirements prioritization problems):**

(a) Sensitive to the mutation and crossover rates, which is unsurprising when we think about an extreme case: a 99% chance of a mutation of each bit on each generation means too little stability.

(b) Sensitive to the gradient of the search space curve leading towards solutions, as are almost all search algorithms (except exhaustive search and pure random search); if there’s no way to tell when we are getting close to a solution, then there’s no way to optimize the search. Thus GAs will not help a naive search for a huge number of factors: we don’t know when we are close to the right factors, we only know when we hit them exactly.

(c) Critically dependent on the fitness function. If fitness function is undefined then GA is inapplicable too.

**Problems:** The primary problem with GAs is that they require that the entire search space be a valid input to the evaluation function. In the worst case, Genetic Algorithms can severely distort the search space.
2.5 Summary: State of the Art

A high-level overview of all the state-of-the-art approaches discussed in the previous sections is presented in Table 2.2, 2.3 and 2.3.

Table 2.2 represents a comparison of the prioritization techniques that require pairwise comparison.

<table>
<thead>
<tr>
<th>Approach</th>
<th>AHP</th>
<th>Bubble Sort</th>
<th>CBRank</th>
<th>Cost-Value Appr.</th>
<th>IAHP</th>
<th>MST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Req, Criterion</td>
<td>Req</td>
<td>Req, Criterion, Knowledge</td>
<td>Req</td>
<td>Req, Criterion</td>
<td>Req</td>
</tr>
<tr>
<td>User/Domain Knowledge</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Scale</td>
<td>Ratio</td>
<td>-</td>
<td>Relative</td>
<td>Ratio</td>
<td>Ratio</td>
<td>-</td>
</tr>
<tr>
<td>Comparison</td>
<td>(n*(n-1))/2</td>
<td>(n*(n-1))/2</td>
<td>-</td>
<td>2 × AHP</td>
<td>n-1</td>
<td>n-1</td>
</tr>
<tr>
<td>Pros</td>
<td>Complete, Consistent</td>
<td>easy to use</td>
<td>considers knowledge</td>
<td>more practical</td>
<td>stopping rule; efforts saving</td>
<td>not redundant</td>
</tr>
<tr>
<td>Cons</td>
<td>Scalability</td>
<td>Scalability, time</td>
<td>can’t resolve contradictory</td>
<td>Time consuming</td>
<td>Inconsistent for large set of Reqs.</td>
<td>Inability to identify inconsistency</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of methods that require pairwise comparison

Table 2.3 and 2.4 represent summarized framework for the prioritization techniques that do not require pairwise comparison.
<table>
<thead>
<tr>
<th>Approach</th>
<th>BST</th>
<th>GA</th>
<th>$100 Method</th>
<th>MoScoW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Reqs</td>
<td>Reqs, Knowledge</td>
<td>Reqs, $100</td>
<td>Reqs</td>
</tr>
<tr>
<td>User/Domaian Knowledge</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Scale</td>
<td>-</td>
<td>-</td>
<td>Ratio</td>
<td>Nominal</td>
</tr>
<tr>
<td>Pros</td>
<td>Handle scalability problem better</td>
<td>Consider knowledge, fast convergence</td>
<td>easy to use, higher granularity</td>
<td>easy to use</td>
</tr>
<tr>
<td>Cons</td>
<td>Difficult to use</td>
<td>Can’t resolve contradictory; require valid search space; not general</td>
<td>Longer; less confidence, biased</td>
<td>Ambiguous final ordering</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison of methods that do not require pairwise comparison: part-I

<table>
<thead>
<tr>
<th>Approach</th>
<th>Numerical Assignment</th>
<th>Planning Game</th>
<th>Priority Groups</th>
<th>Simple Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Reqs</td>
<td>Reqs</td>
<td>Reqs, Knowledge</td>
<td>Reqs</td>
</tr>
<tr>
<td>User/Domaian Knowledge</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Scale</td>
<td>Nominal</td>
<td>-</td>
<td>-</td>
<td>Ordinal</td>
</tr>
<tr>
<td>Pros</td>
<td>easy to use</td>
<td>reasonable amount of time</td>
<td>handle scalability problem better</td>
<td>best for small number of reqs</td>
</tr>
<tr>
<td>Cons</td>
<td>Ambiguous final ordering</td>
<td>difficult to use</td>
<td>difficult to use</td>
<td>unable to handle complex scenarios</td>
</tr>
</tbody>
</table>

Table 2.4: Comparison of methods that do not require pairwise comparison: part-II
Chapter 3

Genetic Algorithm

A brief discussion on GA have already been presented in Chapter 2 under Section 2.4.2. In this chapter, we present some of the genetic operators and parameters that are closely related to GA.

3.1 Genetic Algorithm Operators

Here, an idea of few operators that are usually used in the Genetic Algorithm are presented briefly. As the GA operators Selection, Crossover & Mutation operators has been introduced.

3.1.1 Crossover

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome(s) (in our case a candidate order) from one generation to the next [19]. Crossover aims to interchange the information and genes between candidate orders. Therefore, the crossover operator combines two or more parents to reproduce new children. One of these children may hopefully collect all good features that exist in its parents. The crossover operator is applied to mating pairs based on probability. With given probability, crossover is used, otherwise the resultant offspring are simply copies of the parents. A commonly used crossover rate is between 0.75 and 0.90.

A classification of crossover operators is shown in Table 3.1:
### Binary Coded Crossover Operators

Some major crossover techniques are described below:

Among the three classes, *Binary Coded Crossover Operators* is one of the most commonly used approach. It has other four subdivisions. All of them are shortly introduced here.

**One-Point Crossover (1-PX):** A single crossover point on both parents’ organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms.

![Figure 3.1: Example: An One-Point Crossover Operator (1-PX)](image)

**k-Point Crossover (k-PX):** In this approach, k points are chosen randomly from two parents and
then new chromosome is produced alternatively choosing gene sequences from those parents one after another. Functionalities are almost the same as 1-point, except for the multiple cut points. A simple 3-point crossover is presented in Figure 3.2.

![Figure 3.2: Example: A 3-Point Crossover Operator (3-PX)](image)

**Uniform Crossover:** In the uniform crossover, using a template chromosome and two parents a new offspring is generated by applying the rule that if the bit in template is 0 & both the parent bit are not same then put the compliment bit in the corresponding offspring bit, else keep the same. A simple example of a uniform crossover operator can be viewed in Figure 3.3.

![Figure 3.3: Example: A Uniform Crossover Operator](image)

**Diagonal Crossover:** Diagonal Crossover creates N offspring from N parents by dividing each parent into N sections with (N-1) crossover points. Thereafter, the crossover composes the offspring by taking the resulting N sections from the parents along the diagonals. An example of diagonal crossover operator is presented in Figure 3.4.

![Figure 3.4: Example: A Diagonal Crossover Operator](image)
Real Coded Crossover Operators

**Arithmetical Crossover:** If the chromosomes are made up of floating point numbers instead of binary bits. Arithmetical crossover is defined as a linear combination of two vectors (i) if \( v_t \) and \( w_t \) are to be crossed, the resulting offspring are:  
\[
\text{Offspring}_t = a w_t + (1-a) v_t
\]
(ii) if the parameter \( a \) is a constant then this is called Uniform Arithmetical Crossover. An example of arithmetical crossover operator using logical-AND is presented in Figure 3.5.

![Figure 3.4: Example: A Diagonal Crossover Operator](image1)

![Figure 3.5: Example: An Arithmetical Crossover Operator](image2)

**Mid-Point Crossover:** Given two parents where \( X \) and \( Y \) represent a floating point number for a gene value: the mid-point crossover operator can be found in this way: if Parent 1: \( X \) & Parent 2 = \( Y \) then Offspring := \((X+Y)/2\). A simple example of Mid-Point Crossover is presented in Figure 3.6.

![Figure 3.6: Example: A Mid-Point Crossover Operator](image3)
Simplex Crossover: The simplex crossover uses two better parents and one poor parent and makes one offspring. When both better parents have the same ‘0’ or ‘1’ at a certain bit position, the offspring copies the bit into the same bit position from the first/second better parent. When better parents have different bit at a certain bit position, then a complement bit of the poor parent is copied to the offspring. A simple example of simplex crossover is shown in Figure 3.7.

\[
\begin{array}{cccccccc}
\text{Better-Parent1} & 1 & 0 & 0 & 1 & 0 & 1 \\
\text{Better-Parent2} & 0 & 1 & 0 & 0 & 1 & 1 \\
\text{Poor-Parent} & 0 & 1 & 1 & 0 & 0 & 1 \\
\text{Offspring} & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

Figure 3.7: Example: A Simplex Crossover Operator

3.1.2 Mutation

Mutation is a genetic operator that alters one or more gene-values (i.e. requirements) in a chromosome (e.g. candidate order) from its initial state [19]. With the new orderings, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as it helps to prevent the population from stopping its progress or to reach a higher fitness from any local optima. Mutation occurs during evolution according to a user-definable mutation probability. This probability should usually be set fairly low (0.01 is a good first choice). If it is set too high, the search will turn into a primitive random search, which is not the GA goal.
Flip Bit: Simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes.

Boundary: Replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly). Can only be used for integer and float genes.

Uniform: Replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. Can only be used for integer and float genes.

Gaussian: Adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. Can only be used for integer and float genes.

Briefly, mutation operator can be summarized for those four mutation as below:

- With some low probability, a portion of the new individuals will have some of their bits flipped (for flip bit)
- Its purpose is to maintain diversity within the population and inhibit premature convergence
- Mutation alone induces a random walk through the search space
- Mutation and selection (without crossover) create a parallel, noise-tolerant, hill-climbing algorithms

3.1.3 Selection

We have used tournament selection as our selection operator, a brief introduction to the tournament selection has also taken place here.
Tournament Selection

A given number of individual is randomly chosen from the population. The best individual from this group is selected. Tournaments are often held between pairs of individuals (tournament size $s = 2$). This is known as *Binary Tournaments*, with $s = 2$. In this variant, two individuals are chosen at random and the better of the two individuals is selected with fixed probability $p$, $0.5 < p \leq 1$. Each competition in the tournament requires the random selection of a constant number of individuals from the population. The comparison among those individuals can be performed in constant time, and $n$ such competitions are required to fill a generation. Thus, the time complexity of Tournament Selection is $O(n)$.

Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel where all chromosomes are placed each with its place size proportional to its fitness function.

![Figure 3.9: An Example: Roulette Wheel Selection](image)

Then a marble is thrown to select the chromosome. Chromosome with bigger fitness will be selected more times. This can be simulated by the following pseudo code.

1. \[\text{Sum}\] Calculate sum of all chromosome fitnesses in population, that is $S$ (where $s$ is the fitness value of individuals)

2. \[\text{Select}\] Generate a random number from interval $[0, S]$, that is $r$

3. \[\text{Loop}\] Go through the population and sum fitnesses from 0, that is $\text{Sum}(s)$. When the $\text{Sum}(s)$ is greater than $r$, stop and return the current chromosome

   Of course, step 1 is performed only once for each population.

Rank Selection

The roulette wheel selection faces problems when the fitnesses differs very much. For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the population and then every
chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population). The roulette wheel when changing fitness to order number can be easily figured out from Figures 3.10 and 3.11.

![Figure 3.10: An Example: Situation before ranking (graph of fitnesses)](image1)

![Figure 3.11: An Example: Situation after ranking (graph of order numbers)](image2)

After this, all the chromosomes have a reasonable chance to be selected. However, this method can lead to slower convergence, because the best chromosomes do not differ so much from the other ones.

**Steady-State Selection**

The main idea of this selection is that big part of chromosomes should survive to next generation. In every generation a few (good - with high fitness) chromosomes are selected for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives and appears unchanged in the new generation.

**Elitism**

When creating a new population by crossover and mutation there is some chance that we will loose the best chromosome. Elitism is the method which first copies the best chromosome (or a few best chromosomes) to the new population. The rest is done in the classical way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.
In general the effect of the Genetic Operators are:

- Using selection alone will tend to fill the population with copies of the best individual from the population
- Using selection and crossover operators will tend to cause the algorithms to converge to a good but sub-optimal solution
- Using mutation alone induces a random walk through the search space
- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm

3.2 Genetic Parameters

There are also some genetic parameters that is introduced briefly below.

3.2.1 Population Size

Population size affects the efficiency of the algorithm. If we have smaller population, it would only cover a small search space and may results in poor performance. A larger population would cover more space and prevent premature convergence to local solutions. At the same time, a large population needs more evaluation per generations and may slow down the convergence rate.

3.2.2 Probability of Crossover

Probability of crossover or crossover rate is the parameter that affects the rate at which the crossover operator is applied. A higher crossover rate introduces new strings more quickly into the population. If the crossover rate is too high, high performance strings are eliminated faster than necessary selection in order to produce improvements.

3.2.3 Probability of Mutation

Probability of mutation or mutation rate is the probability with which each bit position of each string in the new population undergoes a random change after a selection process. A low mutation rate helps to prevent any bit positions from getting stuck to a single value, where as a high mutation rate results in essentially random search.
3.3 Genetic Algorithm pseudo-code

GAs is a robust general-purpose search algorithm based on the mechanism of natural selection and natural genetics [21]. Genes and chromosomes are the fundamental elements in GAs. A chromosome is a string of genes. In a real problem, genes are the variables that are considered influential in controlling the process being optimized, and a chromosome is a solution to the problem. Genetic Algorithms search for the optimal solution from populations of chromosomes. The representation chosen for the genome is pivotal to the performance of GA [12].

The Canonical GA (a simple pseudo code is presented here):

1. choose initial population
2. evaluate each individual’s fitness
3. determine population’s average fitness
repeat:

4. select best-ranking individuals to reproduce
5. mate pairs at random
6. apply crossover operator
7. apply mutation operator
8. evaluate each individual’s fitness
9. determine population’s average fitness
until terminating condition (e.g. until at least one individual has the desired fitness or enough generations have passed)

Termination Conditions for the GA can be one of the following or a combination of them:

(i) Certain number of generations has been reached;
(ii) Budgeting: allocated computation time/money used up;
(iii) An individual is found that satisfies minimum criteria;
(iv) The highest ranking individual’s fitness is reaching or has reached a plateau such that successive iterations are not producing better results anymore;
Chapter 4

Algorithm Description

4.1 Overall Proposed Process

The approach that leads to the goal of prioritization is performed through minimizing the disagreements between a total order of requirements holding priority values and different constraints that are embedded with them or that can be also generated or regenerated during the iterations. For the proposed approach, an Interactive Genetic Algorithm has been used to reach a targeted minimization by taking advantage of interactive feedback from the requirement engineers or stakeholders. This interaction feedback helps to define part of the fitness function that cannot be inferred from the existing information during the iterations.

Among the evolved populations, each individual represents an alternate solution candidate with a priority ordering. When individuals with the highest fitness values cannot be classified to extract the top rank one, as their fitness functions produce multiple equal valued orderings, user feedback is requested iteratively, to make the fitness function move downward, for further minimization. Highest fitness refers to the lowest disagreements w.r.t all constraints. The prioritization process terminates when either:

- a threshold disagreement is reached or
- the preferred time out is reached or
- a predefined total number of elicited pairs (the idea of elicitation process will be described formally later in this section) has been reached (but not crossed)

Threshold disagreement, time out and the total elicited pair; in the algorithm, these three are the baseline for termination during the evolutionary process, checking for minimum disagreement, execution time and count of elicited pairs respectively.

4.1.1 High Level Overview

A high level overview of the proposed approach is presented here along with related modules and participants during the whole evolutionary process. From the requirement documents, engineer
will acquire requirements details and represent them in a directed acyclic graph. All the initial precedence graphs are constructed in the same way.

![Diagram](image-url)

**Figure 4.1: A high level overview of the proposed IGA approach**

IGA engine then finds out the best individuals according to the disagreement criteria, starting from initial population and precedence graphs. During this process IGA may ask user several times to discriminate between tied individuals with minimum disagreement. During the evolution period this plays the vital role in the proper selection of individuals with the same fitness. A new precedence graph is produced from the user’s preferences.

Finally, before checking the loop termination conditions a new calculation of disagreement against all three precedence graphs takes place. The individual with minimum disagreement is returned as output when the loop is no more iterated.

In Table 4.1, lower priority level refers to more important requirements which are given preference. Requirements with higher priority level will be preceded by the requirement with the lower one in the precedence graph *Prio*. Also, dependency of a group of requirements on a particular requirement introduces edges from the latter one to each of the member requirement of that group in the precedence graph *Dep*. We will have an initial priority level assigned to each requirements and also the list of requirements on which a requirement is dependent during the development. From those priority level and dependency information, requirement engineer will produce DAGs *Prio* and *Dep*.  

43
<table>
<thead>
<tr>
<th>Req.</th>
<th>Prio. Level</th>
<th>Deps</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>10</td>
<td>R2, R3, R7</td>
</tr>
<tr>
<td>R2</td>
<td>20</td>
<td>R3</td>
</tr>
<tr>
<td>R3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>30</td>
<td>R4, R8</td>
</tr>
<tr>
<td>R6</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>30</td>
<td>R3</td>
</tr>
<tr>
<td>R8</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>50</td>
<td>R8, R5</td>
</tr>
</tbody>
</table>

Table 4.1: Nine requirements with priorities and dependencies

Figure 4.2: Prio: The priority graph for the requirements
For the detailed clarification of the approach, let’s consider the 9 requirements in Table 4.1. For each requirement, a priority value is presented, expressed by the requirement engineer, together with its corresponding dependencies with the other requirements in the project. Constraints and properties of the requirements can be presented by means of precedence graphs which also have been presented in Figure 4.2 and 4.3 for priority and dependency respectively.

**Some properties of the precedence graphs:**

- For these graphs, we always assume that the graphs are directed and acyclic (i.e. DAG). The precedence graphs are directed, as for the *Prio* two requirements can have same the priority level but two requirements with different priority levels must be at distinct level. For the *Dep*, 2 (or more) requirements cannot be mutually dependent in any case. One must be dependent on another or vice-versa.

- In the precedence graph an edge between two requirements indicate that according to the constraints the source node requirement should be implemented before the target node requirement.

- The edges may be weighted, actually to quantify the strength or importance of such a precedence; also infinite weight can be used for precedence relations that must in the final ordering of the requirements. For simplicity, each edge has been assumed the weight of 1 in the example.
During the prioritization process, the elicited precedence graph (Eli) will have more edges and after certain number of steps, will have its final shape with newly introduced constraints between requirement pairs.

There may be other types of constraints that can also be considered in the precedence graphs. For example, the priority level can be assigned to the requirements according to the current total approved budget. There is an estimated cost for each requirements, thus more required and low cost requirements can be given higher priority level or there can be a trade-off between cost & urgency for setting requirement’s priority level. So, this can define new ranking of the requirements and thus another precedence graph.

**An example with nine requirements:**

Figure 4.2 shows the precedence graph induced by the property priority (Prio) of the given requirements. Figure 4.3 represents the precedence graph induced by the dependencies (Deps) among priorities. Requirements with lower priority level should precede requirements with higher priority level. Hence, edges (R1, R4), (R8, R5) or (R7, R6) etc are introduced in the Prio graph. For Deps graph, R5 is dependent on R4 & R8, so R4 & R8 should be preceded by R5. Thus edges (R4, R5) and (R8, R5) are added in Deps graph.

<table>
<thead>
<tr>
<th>Pop. ID</th>
<th>Candidates</th>
<th>Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr1</td>
<td>&lt; R1, R3, R2, R4, R5, R6, R7, R8, R9 &gt;</td>
<td>8</td>
</tr>
<tr>
<td>Pr2</td>
<td>&lt; R2, R3, R4, R1, R5, R8, R6, R7, R9 &gt;</td>
<td>8</td>
</tr>
<tr>
<td>Pr3</td>
<td>&lt; R5, R2, R1, R3, R7, R8, R6, R9, R4 &gt;</td>
<td>16</td>
</tr>
<tr>
<td>Pr4</td>
<td>&lt; R4, R5, R6, R3, R2, R1, R8, R9, R7 &gt;</td>
<td>15</td>
</tr>
<tr>
<td>Pr5</td>
<td>&lt; R7, R8, R6, R5, R2, R3, R4, R9, R1 &gt;</td>
<td>23</td>
</tr>
<tr>
<td>Pr6</td>
<td>&lt; R5, R6, R7, R8, R9, R1, R2, R3, R4 &gt;</td>
<td>29</td>
</tr>
<tr>
<td>Pr7</td>
<td>&lt; R9, R8, R7, R6, R5, R4, R3, R2, R1 &gt;</td>
<td>30</td>
</tr>
<tr>
<td>Pr8</td>
<td>&lt; R8, R9, R6, R7, R4, R5, R2, R3, R1 &gt;</td>
<td>29</td>
</tr>
<tr>
<td>Pr9</td>
<td>&lt; R1, R3, R5, R7, R9, R2, R4, R6, R8 &gt;</td>
<td>17</td>
</tr>
<tr>
<td>Pr10</td>
<td>&lt; R1, R4, R2, R3, R5, R6, R9, R8, R7 &gt;</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.2: Prioritized candidate requirements and their related disagreements

The interactive genetic algorithm that has been used for requirements prioritization evolves a population of individuals, each representing a candidate for the final ordering. Table 4.2 shows 10 individuals (aka, prioritizations or candidates). A single candidate is the permutation of the sequence of all requirements to be implemented in different phases. In order to evolve this population, their fitnesses need to be evaluated first. To measure the disagreement, the inconsistency between the total order encoded in the precedence graphs constructed form the requirement documents is measured. Further constraints will be gathered from the user
feedback during the prioritizing process through evolution. Such relations or constraints are also considered during the disagreement computation in the steps afterwards. This is the *Elicited Precedence Graph* and it is initially empty. New edges are introduced during iterations. In the running example, it will be the third precedence graph that will be added beside Prio and Deps.

Initially no elicited precedence graph is available. Hence, disagreement is computed only compared to the initial graphs obtained directly from the requirement specification guidelines. 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. Considering Pr$_1$ in Table 4.2, it can be noticed that R5, R6 and R7 come before R8, R7 & R8 and R8 respectively while these occur in the opposite order according to the precedence graph Prio. This accounts for 4 pairs in the disagreement namely (R5, R8), (R6, R7), (R6, R8) and (R7, R8). With respect to the partial order Deps, Pr$_1$ also has another 4 disagreements namely, R1 and R5 come before R3, R7 & R2 and R8 respectively. Thus another four pairs i.e. (R1, R3), (R1, R7), (R1, R2) and (R5, R8) have been found. In total this accounts for disagreement 8 for Pr$_1$.

Considering Pr$_2$ in Table 4.2, it can easily be seen that R2, R4, R5 and R6 come before R3 & R1, R1, R8 and R7 respectively while these are opposite to the precedence graph Prio. This accounts for 5 pairs in the disagreement namely (R2, R3), (R2, R1), (R4, R1), (R5, R8) and (R6, R7). W.r.t. the partial order Deps, Pr$_2$ also has another 3 disagreements namely, R2, R1 and R5 come before R3, R7 and R8 respectively. Thus another 3 pairs i.e. (R2, R3), (R1, R7) and (R5, R8) have been found. In total this accounts disagreement 8 for Pr$_2$ too. In a similar way, the other 9 candidates can be assumed w.r.t. the precedence graphs presented in Figure 4.2 and Figure 4.3.

In order to select the best individuals, the disagreement measure is considered as an indicator of fitness can be considered. The best individuals are then evolved into the new population by applying some mutation and crossover operators to them, the operators will be described in detail in the next sections. However, it may happen that such an indicator does not allow a precise discrimination of some high ranked individuals holding the same fitness value. In such case we take feedback from the users, that will be the primary source of additional information to be used as new constraints. In Table 4.2 this happens for individuals Pr$_1$ and Pr$_2$ (having equal disagreements of 8). The available fitness function cannot guide the search for an optimal prioritization, since Pr$_1$ & Pr$_2$ cannot be ranked relatively to each other; this means that currently available precedence graphs (or information) do not allow finding out the best ordering.

At this stage, the requirement engineer is asked for information that allows users to rank the prioritizations in a tie situation. More specifically, the disagreement between each couple of prioritizations in a tie need to be considered. The pairs in the disagreement are those on which the equally scored candidates differ. Hence, they can be easily discriminated if it’s decidable on the precedence holding for such pairs. As an outcome of the user feedback,
a pairwise comparison between requirements that are ordered differently in equally scored prioritizations will be listed. Considering the disagreement between Pr\textsubscript{1} and Pr\textsubscript{2}, consisting of (R1, R3), (R2, R3), (R6, R8) and (R7, R8), all these pairs can be responsible for proper discrimination. In Table 4.3 we show the pairs in the disagreement between Pr\textsubscript{1} and Pr\textsubscript{2}.

<table>
<thead>
<tr>
<th>Tie</th>
<th>Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr\textsubscript{1}, Pr\textsubscript{2}</td>
<td>(R1, R3), (R2, R3), (R6, R8), (R7, R8)</td>
</tr>
</tbody>
</table>

Table 4.3: Pairwise comparison to resolve ties between individuals

The user is requested to express a precedence relationship between each pair in the disagreement computed for candidates in a tie. Given a pair of requirements e.g. (R1, R3) the user can respond in the way:

- R1 precedes R3: R1 $\rightarrow$ R3
- R3 precedes R1: R3 $\rightarrow$ R1 and
- don’t know: no edge

In the first two cases, a new edge will be introduced in the precedence graph Eli whereas no edges will be added for the last response. In this example, the user will be asked pairs (R1, R3), (R2, R3), (R6, R8) and (R7, R8). After collecting user input in terms of pairwise comparisons, the existing elicited graph will be updated with the new precedence edges. The appearance of a cycle in the elicited graph indicates the existence of contradictory information. Hence, if there is any cycle, there will be no disagreement with any orderings that involves the requirements in that cycle. The cycle can be interpreted as don’t care ordering of the requirements involved or can be prevented explicitly.

Eli: The elicited graph for our example after having user feedback considering the order of preference between the elicited requirement pairs is presented in Figure 4.4.

Figure 4.4: Eli: The elicited pairs, representing edges in the third precedence graph

When the new elicited precedence graph is available (partially or fully depending on the
variables used), the fitness function is computed again for all the candidates. Such a fitness evaluation is expected to be much more discriminative than the previous one. Thus high ranked individuals can easily be distinguished. The best individuals in the population are selected, applied mutation or crossover operators to constitute the next population. After a number of generations, the algorithm is able to discriminate the best individuals in the population, thus leading to a final selection of the prioritization with lowest disagreement (highest fitness) with respect to all precedence graphs including the elicited one.

**What is Elicitation Process?**

The process of finding and formulating requirements is called elicitation [28]. According to Lauesen, there are several techniques that may used for elicitation. Likewise,

- Stakeholder analysis,
- Observation,
- Document studies,
- Questionnaires,
- Brainstorm,
- Risk analysis,
- Cost/benefit analysis,
- Goal-domain analysis,
- Domain

Requirement pair elicitation is similar kind of task for finding solutions in the requirements prioritization process when there is information deadlock within the *Domain Knowledge* (the sum or range of what has been perceived, discovered, or learned from the requirement documents). User is asked with a pair to decide which one precedes the other. Judging on the basis any of above platforms, user gives his preference.

### 4.2 Operators: IGA

Several GA operators are also the part of the proposed algorithm as it is based on GA functionalities. Three GA operators and their particular type that we included in our algorithm are presented below.
4.2.1 Crossover Operator

In the proposed algorithm, we used a sophisticated crossover operator namely \textit{cut-head/fill-in-tail} and/or \textit{cut-tail/fill-in-head}.

Two parents creating the offspring consist of the genes of the both parents. While generating offspring, there can be three options regarding the fitness of the offspring, it can be weaker, the same or fitter than its parents. The operator we have used will definitely produce a offspring with better fitness as in a sense we are combining both the parents maintaining a existing partial-ordering within either of parents. Of course, the variation caused by this process allows the offspring to search out different available niches, find better fitness values and subsequently better solutions. As our approach is non-deterministic, choosing \textit{cut-head/fill-in-tail} and/or \textit{cut-tail/fill-in-head} will also reflect that properties more suitably. On the other hand, using \textit{cut-head/fill-in-tail} will never produce chromosomes containing duplicate genes. It also greatly accelerates search early in evolution of a population. Below Figure 4.5 is a graphical representation of how \textit{cut-head/fill-in-tail} operator works.

![Figure 4.5: Example: Cut-head/fill-in-tail operator of the proposed IGA](image)

4.2.2 Mutation Operator

We used \textit{requirement-pair-swap} as our mutation operator in the proposed algorithm.

Mutation causes movement in the search space both locally or globally and also restores lost information to the population. Occasionally may produce a stronger chromosome. It can be assumed that mutation is doing something new by changing some part of the chromosome and with no reference or affect from other members of the population. As, our proposed algorithm is non-deterministic, we selected requirements to be swapped randomly. If more sophisticated heuristic can be employed, of course better chromosome can be generated with higher fitness. But by using random selection, the main purpose of mutation in GAs i.e. preserving and introducing diversity, can be achieved easily. Random selection also allow
the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other. Below Figure 4.6 is a graphical representation of how requirement-pair-swap operator works.

![Diagram of requirement-pair-swap operator](image)

Figure 4.6: Example: Requirement-pair-swap operator of the proposed IGA

### 4.2.3 Selection Operator

As the selection operator, we used *Tournament Selection* for selecting the high fitness chromosomes from the population.

The most important benefit for *Tournament Selection* is that it allows the selection pressure to be easily adjusted. As our algorithm is non-deterministic, we also apply random selection within the population for selecting chromosomes with the fixed tournament size (k=2). Also, tournament selection is one of the most effective method for selecting best individuals. A study by Simon Mardle and Sean Pascoe in [41] showed that convergence is generally slower using the Roulette Wheel Selection (see Section 3.1.3) than tournament selection. It can be mentioned that implementing tournament selection is more feasible than other selection mechanisms and can give better convergence too. Below Figure 4.7 is a graphical representation of how we implemented tournament selection in our proposed algorithm.
4.3 The Algorithm

In this section, an Interactive Genetic Algorithm will be described that implements the approach presented in short in the previous section. Before introducing the algorithm, a formal definition of the intuitive notion of disagreement [43] is presented. It plays a fundamental role when evaluating the fitness of an individual and when deciding which pairwise comparisons to elicit form the user. The definition in the general case where two partial orders are compared is given. Of course, it holds also when one or both orders are total ones.

\[ dis(\text{ord}_1, \text{ord}_2) = \{(p, q) \in  \text{ord}_1^* \mid (q, p) \in  \text{ord}_2^*\} \] (4.1)

The disagreement between two (partial or total) orders \(\text{ord}_1\) and \(\text{ord}_2\), defined upon the same set of elements \(R\), is the set of pairs in the transitive closure of the first order, \(\text{ord}_1^*\), that appear reversed in the second order closure \(\text{ord}_2^*\). A measure of disagreement is given by the size of the set \(\text{dis}(\text{ord}_1, \text{ord}_2)\) [43].

The other input of the algorithm is a set of one or more partial orders \(\text{ord}_1, \ldots, \text{ord}_k\),
derived from the requirement documents (i.e. Prio & Deps 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 initial (input) partial orders, so that we can start with an
already good shaped population. Greedy heuristics may be used in this step to produce better
initializations.

**Algorithm 1** Compute Prioritized Requirements

**Input** $R$: A set of requirements

**Input** $ord_1, \ldots, ord_k$: partial orders defining priorities and constraints upon $R$ ($ord_i \subseteq R \times R$
defines a DAG)

**Output** $<R_1, \ldots, R_n>$: An ordered list of requirements

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

Input of the proposed algorithm is primarily a list of requirements and some constraints defining priorities and dependencies between them. Output will be the ordered list of requirements. An in-depth description of the algorithmic steps follows.

**Step 1-6:** Set some important parameters of the algorithm. Step 1 initializes the population. Step 2 to 5 initialize few variables, like counter of the total elicited pairs (`elicitedPairs`), maximum number of elicited pairs (`maxElicitedPairs`), a minimum disagreement value upon which evolution process can be stopped (`thresholdDisagreement`). Finally the graph of elicited orders (`eliOrd`) by which the final elicited precedence graph will be constructed. Another relevant parameter of the algorithm is the maximum execution time (`timeOut`) which bounds the total optimization time. Also, the typical parameters of any GA i.e. population size, proportion of mutation w.r.t crossover etc are initialized here.

**Step 7-9:** The fitness of the individuals are measured by their disagreement with the initial available precedence graphs. Then the main loop for optimization is entered. New generations of individuals are produced as long as the disagreement is above the threshold value. After the maximum execution time, the algorithm stops anyway and reports the individual with disagreement w.r.t the initial & newly elicited partial order from the user.

**Step 10-16:** Step 10 indicates the stopping conditions of the evolutionary process. And among 11-16, the first operation to be performed is selection in step 11. Though any selection mechanism can be used, best performance have been experimented when using tournament selection. When evolutionary optimization process 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 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`). At this stage, it also needed to update the counter for the number of total elicited pair (`elicitedPairs`).

Resorting to the user to resolve ties is the most innovative aspect for our approach and represents the major difference with the non-interactive state-of-the-art approaches. This aspect made our prioritization approach better in terms of performance and superior in terms of acceptance level. Till now, all the state-of-the-art approaches do not include any **User Intervention** in the middle of the process. We took advantage of it to remove all the ambiguities, to properly discriminate among equally ranked individuals during the optimization process.

**Step 17-18:** After the selection and the optional interactive step, the population is evolved through mutation and crossover. For mutation, **requirement-pair-swap operator** has
been used, which consists of selecting two requirements and swapping their position in the muted individual. Selection of the two requirements to swap can be done randomly and may either involve neighboring or non-neighboring requirements. For example, if mutate individual Pr\(_1\) in Table 4.2 and select R2, R7 for swap, we obtain the new individual Pr\(_1^\prime\) = < R1, R3, R7, R4, R5, R6, R2, R8, R9 >. More sophisticated heuristics for the selection of the two individuals to swap may be employed as well (i.e. based on the disagreement of the individual with the available precedence graphs). In the reported experiments, only random selection has been considered.

For the crossover, cut-head/fill-in-tail and cut-tail/fill-in-head operators have been used, 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, while doing crossover Pr\(_1\) and Pr\(_2\) using the cut-head/fill-in-tail and selecting the cut point at between 4 & 5, will have, Pr\(_1^\prime\) = < R1, R3, R2, R4, R5, R8, R7, R6, R9 > and Pr\(_2^\prime\) = < R2, R3, R4, R1, R5, R6, R7, R8, R9 > i.e. keeping both the head < R1, R3, R2, R4 > and < R2, R3, R4, R1 > then doing fill-in the tail with the missing requirements in the order in which they appear in respectively Pr\(_2\) & Pr\(_1\). Selection of the cut point can be done randomly as well as following a heuristics approach too i.e. cut point can associate a requirement pair which has high disagreement w.r.t precedence graphs.

A Short Note: The mutation & crossover operations described above may generate chromosomes (individuals) which are already part of the new population being formed. In general this is not a problem as the best individuals can be present multiple times. It may become a problem in degenerate cases where most of the population has of a single or 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 already present chromosomes in the population. Mutation and crossover are applied repeatedly, until the population diversity exceeds a predefined threshold.

Step 19-22: The last few steps of the algorithm from 19-22 determine the fitness measure to use in the next selection process of the best individuals. This computation of the disagreements takes into account all the three precedence graphs including the third one, Eli graph.

As a final note, it can be mentioned that 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.
Chapter 5

The Case Study

For our work, we have used ACube [4], a project funded by the local government of the Autonomous Province of Trento, in Italy. In the following sections a detailed description of the technical and functional requirements are presented;

5.1 ACube Project:

What is ACube?
ACube, the acronym Ambient Aware Assistance is a project started on October 7, 2008 with a duration of 36 months, coordinated by Fondazione Bruno Kessler (FBK) and funded by Autonomous Province of Trento under Bando Grandi Progetti, 2006. The Autonomous Province of Trento has identified Social Welfare as an important strategic objective. To meet this goal, it supports the creation of an Innovative Information System offering integrated services and increased service efficiency, effectiveness and quality.

The three screen shots presented in the Appendix in Figure A.1, A.2 and A.3 represent a part of the ACube Project, showing three possible macro scenarios for MON A.1, ESC A.2 and FAL A.3.

5.1.1 Innovative Aspects
The major technical outcome of the project is the development of a monitoring system that requires the integration of a variety of heterogeneous technologies i.e. video, audio, rfid, wsn and biomedical etc. The development of advanced algorithms to recognize events, situations, activities, behaviors in complex multi-person scenarios will enable the smart environment to understand who is doing what, where, when and how. This knowledge allows the system intelligence to take decisions (i.e. rising alarms). Processing includes adaptation capabilities, to fit different environments and users.
5.1.2 Technology

The project is highly technical, both in terms of the sensors (i.e. microphones, video camera, accelerometers, environmental sensors) and the algorithmic aspects for reasoning and planning. The communication infrastructure has been conceived to offer a high degree of configurability with a distributed environment in which the sensors and algorithms work together. To support the project, an event-driven middleware has been realized to mediate among the various technologies and to adapt to the natural events typically found in the domains and applications addressed by ACube. The possibility to use wireless technologies with low energy consumption identifies ACube as a sustainable system in terms of its impact on the structure where it will be employed.

5.1.3 Technical Objectives

Create an advanced, generic monitoring infrastructure for Assisted Living, able to monitor in a uniform, adaptive, and high quality manner the patients, the environment and its operators, and the ongoing activities [35].

- A monitoring system: Connected to the environment through a distributed sensor networks and actuators.
- Capability to operate in complex scenarios: To be able to recognize, analyze and support complex physical and social processes.
- Highly technological: Autonomous and self-configurable.
- Low intrusiveness: Adaptation capabilities to fit different environments and users.


Context: In a typical scenario, ACube acts as a supervisor of a physical environment where people with cognitive problems (some are more severe than others) are living. ACube serves as an aide to the various caregivers who provide both monitoring and care for the guests.

5.1.4 System requirement details

ACube aims 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 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) and three macro-scenarios have been identified. Specifically, the three macro scenarios are:

(i) *Localization and tracking to detect falls of patients* (*FAL*)
(ii) *Localization and tracking to detect patients escaping from the nursing home* (*ESC*)
(iii) *Identification of dangerous behaviors of patients* (*MON*) and
(iv) *A comprehensive scenario that involves the simultaneous presence of the previous three scenarios* (*ALL*)

Table 5.1 summarizes the number of technical requirements for each macro-scenario. Together with the set of technical requirements, we considered two sets of technical constraints: 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 1–1000), 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.

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Macro Scenario</th>
<th>Total no of reqs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON</td>
<td>Monitoring dangerous behavior</td>
<td>21 reqs.</td>
</tr>
<tr>
<td>ESC</td>
<td>Monitoring escapes</td>
<td>23 reqs.</td>
</tr>
<tr>
<td>FALL</td>
<td>Monitoring falls</td>
<td>26 reqs.</td>
</tr>
<tr>
<td>ALL</td>
<td>All three scenarios together</td>
<td>49 reqs.</td>
</tr>
</tbody>
</table>

Table 5.1: Four macro-scenarios and the number of technical associated requirements

One of the top priority requirements categorized by the stakeholders with the priority label ‘10’ for the ALL scenario is:

*Rt001: The system monitors the private rooms of the center.*
Another top priority requirement categorized by the stakeholders with the same priority label for the FALL scenario is:

RT007: The system identifies the presence of a person in a given area.

A top priority requirement categorized by the stakeholders with the priority label 10 for the ESC scenario is:

RT003: The system monitors, via sensors, the external area of the centers (gardens, gate).

And one top priority requirement categorized by the stakeholders with the same priority label for the MON scenario is:

RT012: The system identifies the exact position of a person with respect to some objects (static or dynamic).

Technical Requirements: A detailed list of all technical requirements for the ACube project is presented in Appendix A followed by Sections A.1, A.2, A.3 and A.4 for MON, ESC, FAL and ALL macro scenarios respectively.

Gold Standard:
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. Of course, in reality the GS is not available explicitly. In our case, we used it to show the comparative results with other methods.

Not all methods are fully automatic i.e. for every requirements prioritization methods we need some predefined input domain knowledge and user knowledge. This input domain knowledge can be the requirement list with their descriptions or explicitly mentioned urgency or other related information by which one set of requirements can be distinguished from the other. It depends on the used methodology. In our case, all needed information is available for ACube. When we applied our method, the ACube project was at the initial development state. We had a long list of requirements which we received from the engineer of the project (software architect) and then we applied our proposed technique for prioritizing the requirements, having more elaborative input domain knowledge which includes the individual priorities according to the customers and the inter-dependencies analyzed and defined by the requirement engineer.

We assessed our method by comparing method’s output (prioritized list of requirements) with the Gold Standard.
Chapter 6

Experiments & Results

6.1 Experiments

The term experiment is defined as the systematic procedure carried out under certain conditions in order to discover an unknown effect or to test/establish a hypothesis or to illustrate a known effect [3]. To evaluate the performance of our approach in a practical way, a fully structured experiment has been conducted. The goal of this experiment was to evaluate the effectiveness of the new interactive approach for requirements prioritization w.r.t. the non-interactive state-of-the-art approaches. Here is a full illustration of the formal experiment that we have conducted.

6.1.1 Design

The three main components of an experimental design [3] are (i) factors, (ii) levels and (iii) response.

Factors refer to the inputs to the process and can be classified as either controllable or uncontrollable variables. In our case main factors are (i) Applied algorithm, (ii) User model and (iii) Degree of available knowledge. There are also other minor factors i.e. number of elicited pair, total execution time, minimum threshold disagreement etc.

Levels or settings of each factor i.e. the value for those factors that are defined within the environment and going to be used during the execution.

Response or output of the experiment. In our case, mainly the disagreement w.r.t. the Gold Standard (formally defined in Section 5.1.4, also see Sections A.1, A.2, A.3 & A.4).

Experimental Design: The factors to be tested

(i) Applied Algorithm, for our experimental design we used several prioritization algorithms including ours. We want to compare the resulting disagreement w.r.t. GS of the final ordering using both interactive & non-interactive approaches.
(ii) **User Model**, in our experiments we have shown both the cases where there is an ideal user (i.e. user makes no error) and the user that frequently gives wrong answers. So, having different settings for the user model will help us to evaluate our algorithm under different user error conditions.

(iii) **Degree of Available Knowledge**, availability of different degrees of knowledge also affects the performance of IGA. We experimented with both or any of the precedence graphs (i.e. Prio & Dep) available.

The other minor factors that are merely considered as an important part of the optimization process are:

(a) **Maximum Elicited Pairs**, for each target objective we will have a maximum number of elicited pair, the algorithm terminates when it reaches this value.
(b) **Threshold Minimum Disagreement**, a minimum disagreement value will be used as threshold value.
(c) **Time Out**, the total timeOut set for running the evolutionary part of the algorithm. After the specified timeOut is reached the evolution process will terminate.
(d) **Top Population Percentage**, if we have N individuals in the population and topPopulation-Perc is set to k% then N×k% will be evaluated for further discrimination using interaction with the user when there are ties.

**Experimental Preparation: The levels of the factors**

(i) **Applied Algorithm**: The algorithm that we apply is the main factor for us i.e. use of different algorithms produce different level of disagreements.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Levels</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>IGA</td>
<td>Interactive, Domain Knowledge &amp; Pairwise comparison</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>Non-Interactive, Domain Knowledge</td>
</tr>
<tr>
<td></td>
<td>RAND</td>
<td>Non-Interactive, Domain Knowledge</td>
</tr>
</tbody>
</table>

Table 6.1: Levels for the factors used in IGA

In Table 6.1 above, GA and RAND approaches are considered as non-pairwise comparison.

(ii) **User Model**: We have conducted experiments with all four macro scenarios with a different number of pairs elicited from the user and with variations of user error rates. More specifically, for all macro scenarios we experimented with 25, 50 & 100 elicited pairs and error rates between 5% and 20%. Table 6.2 reflects the user and error model we had in our case study (numbers remain same for all macro scenarios i.e. ALL, FAL, ESC & MON):
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tot. Eli Pair</td>
<td>Err Rate</td>
<td>Wrong Eli Pair</td>
<td>Elicited% on Total Pairs</td>
</tr>
<tr>
<td>25</td>
<td>5%</td>
<td>2</td>
<td>3, 2, 2, 1</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>3</td>
<td>3, 2, 2, 1</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>5</td>
<td>3, 2, 2, 1</td>
</tr>
<tr>
<td>50</td>
<td>5%</td>
<td>3</td>
<td>6, 5, 4, 1</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>5</td>
<td>6, 5, 4, 1</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>11</td>
<td>6, 5, 4, 1</td>
</tr>
<tr>
<td>100</td>
<td>5%</td>
<td>5</td>
<td>12, 10, 8, 2</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>10</td>
<td>12, 10, 8, 2</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>20</td>
<td>12, 10, 8, 2</td>
</tr>
</tbody>
</table>

Table 6.2: Numerical representation of user error model used for simulation in IGA

In Table 6.2, column A represents the total number of elicited pairs we experimented with. Column B represents error the rates we used. Column C represents the calculated number of elicited pairs that were wrongly answered by user. Column D represents the actual percentage of the elicited pairs w.r.t. total possible pairs in the order MON-ESC-FAL-ALL.

(iii) Degree of Available Knowledge, settings in Table 6.3 presents the availability of domain knowledge in different experiments.

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Available Domain Knowledge</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Both Prio &amp; Dep</td>
<td>IGA, GA, RAND</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Only Prio</td>
<td>IGA, GA</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Only Dep</td>
<td>IGA, GA</td>
</tr>
</tbody>
</table>

Table 6.3: Different degrees of domain knowledge i.e. precedence graphs in three experiments

Using the degree of availability of domain knowledge in Table 6.3 we run our experiments and then boxplot them to get a comparable view to show how it affects the performance of IGA.

The other levels of some minor factors are below:

(a) Maximum Elicited Pairs, the default value is 100; we have experimented all the macro scenarios with 25, 50 and 100 elicited pairs. We then terminate the evolution process and measure the disagreement with the Gold Standard. The primary objective of the diversity in setting the number of maximum elicited pair is to see the fluctuations on the disagreement w.r.t. Gold Standard i.e. how close we can reach in terms of orderings.

(b) Threshold Minimum Disagreement, default value is 0; For our experiments, the threshold
value was set to zero to reach the closest ordering to GS.

(c) Time Out, the total time out set for our experiments was between 480 and 1080 seconds depending on the number of requirements and number of elicited pairs.

(d) Top Population Percentage, default value is 1% for the topPopulationPerc.

**The structure and layout of experimental runs, or conditions:**
For each macro scenarios presented in Appendix A, we run experiments multiple times depending on the total execution time it takes for each run (we reduced the number of runs if it exceptionally consumes more time i.e. if it takes 1800 seconds or more per run).

**Execution**
Since IGA is non deterministic, we replicated each experiments a minimum number of 10 to maximum of 20 times with a timeOut between 600 & 1080 seconds. We varied this execution time due to variability in elicited pairs, the more pairs we elicit, in practice the more operational time it consumes. So, for the proper judgment among the execution settings we assumed a range of 600, 840 and 1080 execution time (in seconds) while eliciting 25, 50 & 100 pairs respectively from the user. We then compute disagreements and average distances after the optimization process terminates. We simulate the artificial user who automatically responds according to the GS in case of error-free settings, and just in the reverse order in case of erroneous settings i.e. when setting the $p_c$ value greater than zero, the user will make wrong elicitation with the probability of $p_c$. Finally, we do the box-plots for all the results we had, to have a clear comparable view with other traditional non-interactive approaches.

**6.2 Research Questions**
In this section, we present the research questions by which we investigate the effectiveness of our method in requirements prioritization. For all the experiments below, we have some common settings which are excluded from the settings tables to avoid duplication of the data presentation, those are: (i) thresholdDisagreement value as 0, (ii) topPopulationPerc as 1%, (iii) mutationProbability value as 10% and (iv) crossoverProbability value as 90%.

The most important research questions that we explored and experimented are below.

**6.2.1 Research Question 1: [Convergence]**

*Can convergence be observed with respect to the finally elicited fitness function? (i.e. all the best individuals across iterations have decreasing disagreement w.r.t. final elicited graph)*
The fitness function in our case is not constant and is formulated gradually during the interactive elicitation process. So, it may not be the case that the algorithm converges. Initially, IGA optimizes the ordering, so as to minimize the disagreement with the available precedence graphs (in our experiment Pri & Dep). Then constraints are added as edges into the Elicited Graph, Eli by the requirement engineer. After having a non-empty Eli as set, encoded as directed acyclic graph, the objective and fitness calculation will slightly change. Thus, the question can be raised whether the overall optimization process converges, once all precedence graphs are considered (a-posteriori), including the elicited one in its final shape. To answer this research question, we measured the final fitness function (i.e. disagreement with all precedence graphs) over generations after the evolutionary process is terminated (hence it is assumed that elicited graph is in its complete shape). It is to mention that such fitness function is absent during the optimization process with initial precedence graphs and with the partially constructed elicited graph Eli.

**Experimental Settings**

Though we show results only for ALL macro scenario (49 req), we also performed experiments for the other macro scenarios i.e. MON, ESC & FALL. To perform the experiment for RQ1 for ALL macro scenario (49 req) we used the settings shown in Table 6.4:

<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Convergence</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Convergence</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
</tbody>
</table>

Table 6.4: Experimental settings for the RQ1

**Experiment Execution**

For running experiments regarding RQ1, we made a total of 20 runs with different number of iterations (i.e. 1200 to 1800 depending on scenarios as the algorithm is non deterministic) during the evolutionary process. We captured all the populations, their disagreements & elicited pairs into external data files. Also, we calculated Disagreement and Average Distance against final elicited graph, (Eli).
Discussion on First Experiment of RQ1

Figure 6.1 shows the convergence for the best individuals through different iterations w.r.t. the final elicited graph. We can observe that the convergence w.r.t to the final fitness function is clearly visible. Hence, the answer to RQ1, which is the primary step for an acceptable approach, is positive. We have considered here the ALL macro scenario (49 req) with 25 elicited pairs and plotted the final disagreement along the y-axis, assuming no user error i.e. 0%. From the initial iteration, the convergence was not so fast as the elicited graph was just growing and hence the populations were not so good w.r.t. to the final elicited graph. After few iterations (from 100-400) a completely declining slope (steep decrease) is observed due to a radical decrease in disagreement, which indicates the algorithm is indeed optimizing (minimizing) the final fitness function. And then from iteration number 600, without few changes in the behavior, the convergence line is almost horizontal. At this stage the elicited graph, Eli is almost in its final shape. We generated similar convergence plots for the other three macro scenario MON, FAL & ESC. For this particular experiment, 600 seconds was used as timeOut.
Discussion on Second Experiment of RQ1

In Figure 6.2, we show the convergence through different iterations with respect to the final elicited graph for ALL macro scenario (49 req) with 100 elicited pairs. We measure the disagreement along the y-axis, assuming there is no error at this time too. From the initial iteration, convergence was very fast (steeper than the case with 25 elicited pairs in Figure 6.1). After few iterations (from 100-500) a completely declining slope (steep decrease) is observed due to a radical decrease in disagreement, which indicates the algorithm is optimizing (minimizing) the final fitness function. Then from iteration number 500, with only slight changes in behavior, the convergence line is almost horizontal. At this stage the elicited graph is almost in its final shape. We also, generated similar convergence plots for other three macro scenario MON, FAL & ESC. For this particular experiment, 1080 seconds was used as timeOut.

It can also be noticed that when we have more elicited pairs (in the previous Figure 6.1 it was 25), we have a steeper decrease and hence better convergence, because the Eli graph grows faster.

For the RQ1, we conclude that the convergence is occurring w.r.t. the final elicited graph (Eli). The results for RQ1 are clearly visible in Figure 6.1 & 6.2.
6.2.2 Research Question 2: [Role of Interaction]

Does Interactive Genetic Algorithm generate improved prioritizations of requirements compared to non-interactive ordering (using state-of-the-art approaches)?

The key point in our research hypothesis is that User Knowledge plays an important role in requirements prioritization process. RQ2 test this hypothesis with respect to the non-interactive approaches. To assess the importance of the elicited information from user, we of course compare the results of IGA with the approaches that do not consider external knowledge feedback. As a sanity check we consider RAN, the random requirement orderings in addition to GA. In GA, a minimization is conducted w.r.t. the initial precedence graphs without eliciting any pairs from the user. This can easily be done by setting \( \text{maxElicitedPair} \) to zero in IGA.

To compare the performance of the proposed approach in a meaningful way, we used two main metrics: Disagreement w.r.t. the Gold Standard and Average Distance from the reference position of each requirement in the GS. Disagreement was already presented in Section 4.3 formally. About the Average Distance, as shown in Equation 6.1, we take the distance count for the position of each requirement against the GS, sum it up and then make the average by dividing by the number of requirements. This is a way of describing the quality of final ordering that is produced by IGA, which is more interpretable than the disagreement.

In fact, disagreement measurement is based on a quadratic number of comparisons, thus its absolute value is not easily understandable. On the contrary, it is easy to understand how far a requirement is from the correct position in GS.

\[
\text{AverageDistance} = \frac{\sum d_i}{N} \tag{6.1}
\]

where, \( d_i \) := distance of \( R_i \) in \( Pr_i \) w.r.t \( R_i \) in GS & \( N := \text{total requirements} \)

Experimental Settings

To perform the RQ2 experiments, for ALL (49 req) and MON (21 req) macro scenarios we used the following experimental settings, shown in Table 6.5 and 6.6 separately.
<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>25, 50 &amp; 100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
</tbody>
</table>

| ALL (49 req)   | targetAlgorithm     | IGA, GA             |
|                | measurement         | Average Distance    |
|                | maxElicitedPairs    | 25, 50 & 100        |
|                | execTime            | 1080 seconds        |
|                | errorPerc           | 0% error rate       |

Table 6.5: Experimental settings for the RQ2 (part-I)

<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON (21 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA &amp; RAND</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>25, 50 &amp; 100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
</tbody>
</table>

| MON (21 req)   | targetAlgorithm     | IGA, GA & RAND      |
|                | measurement         | Average Distance    |
|                | maxElicitedPairs    | 25, 50 & 100        |
|                | execTime            | 600 seconds         |
|                | errorPerc           | 0% error rate       |

Table 6.6: Experimental settings for the RQ2 (part-II)

**Experiment Execution**

To run experiments for RQ2, we made a total of 20 runs with different number of iterations during the evolutionary process. We captured all the populations, their disagreements, elicited pairs into external data files. We also calculated disagreement and average distance against *Gold Standard*. 
Figure 6.3: Box-Plot of Disagreement with respect to GS for 25, 50 & 100 Elicited pairs for ALL (49 req)

Discussion on First Experiment of RQ2 | Metric: Disagreement

Figure 6.3 shows the experimental results when assuming user error rate as 0% and the number of elicited pairs are gradually increased from 25 to 100. We plot the boxplots of disagreement for ALL macro scenario (49 req) with 25/50/100 elicited pairs. The key factor of this result is that even if we have the minimum number of elicited pair (i.e. 25 pairs which is only 2% of all possible pairs) we have better behavior than non-interactive approach (i.e. GA & RAND). Also, while we are increasing the number of elicited pairs from 25 to 50 and from 50 to 100 and so on, performance of IGA itself is increasing i.e. decreasing the disagreement (plotted along the y-axis). When we elicit 25 pairs, the median (Q2) of final disagreement is around 125, with 50 elicited pairs, Q2 improves and becomes 120 while with 100 elicited pairs the median Q2 is around 115. On the other hand, median (Q2) of final disagreement using GA is around 140 and using RAND it is around 590, which are much higher than even after eliciting 25 pairs using IGA. So, User Interaction plays an important role in requirements prioritization.
Figure 6.4: Box-Plot of Average Distance with respect to GS for 25, 50 & 100 Elicited pairs for ALL (49 req)

Discussion on First Experiment of RQ2 | Metric: Average Distance

In Figure 6.4, we show an experimental result measuring the average distance assuming user error rate as 0% and the number of elicited pairs are gradually increased from 25 to 100. Unlike the previous Figure 6.3 we put box-plot of Average Distance here for ALL macro scenario (49 req) with 25/50/100 elicited pairs. The key factor of this result is that even if we have the minimum number of elicited pairs (i.e. 25 pairs which is only 2% of all pairs) we have better behavior than other non-interactive approaches (i.e. GA). Also, while we are increasing the number of elicited pairs from 25 to 50 and from 50 to 100 and so on, performance of IGA itself is increasing i.e. decreasing the average distance (plotted along the y-axis). When we elicit 25 pairs, the median (Q2) of the average distance is around 4.1, with 50 elicited pair, Q2 improves and reaches 3.8 finally with 100 elicited pairs the median Q2 is around 3.6. On the other hand, median (Q2) of average distance using GA is around 4.5 and using RAND it is around 16, which are much higher even than when eliciting only 25 pairs. So, again we have shown that user interaction plays an important role in improving prioritization.
ANOVA: F-measure & p-test

Statistical significance of the observed differences were tested using ANOVA. Table 6.7 and 6.8 represent a comparative overview among IGA, GA and RAND for disagreement and average distance.

<table>
<thead>
<tr>
<th>Disagreement</th>
<th>F value</th>
<th>Pr(&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGA_25, GA, RAND</td>
<td>450.57</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>IGA_50, GA, RAND</td>
<td>453.9</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>IGA_100, GA, RAND</td>
<td>453.22</td>
<td>&lt; 2.2e-16</td>
</tr>
</tbody>
</table>

Table 6.7: Analysis of variance (ANOVA) comparing Disagreement of IGA, GA and RAND

<table>
<thead>
<tr>
<th>Average Distance</th>
<th>F value</th>
<th>Pr(&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGA_25, GA, RAND</td>
<td>605.26</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>IGA_50, GA, RAND</td>
<td>618.92</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>IGA_100, GA, RAND</td>
<td>601.89</td>
<td>&lt; 2.2e-16</td>
</tr>
</tbody>
</table>

Table 6.8: Analysis of variance (ANOVA) comparing Average Distance of IGA, GA and RAND

ANOVA allows and is useful for two or more groups to be compared in a statistical way. The significance of differences between the groups depends on (i) the difference in the means (ii) the standard deviations of each group and (iii) the sample sizes. Basically, ANOVA measures (a) variation BETWEEN groups i.e. for each data value difference between its group mean and the overall mean (b) variation WITHIN groups i.e. for each data value difference between that value and the mean of its group. The ANOVA F-measure, is a ratio of the Between Group Variation divided by the Within Group Variation.

F-value: Usually, higher F-value denotes significant amount of difference among the groups and lower value denotes there is more similar values in the collections.

Pr(>F) value: In general, the smaller the Pr(>F) is, the more significant the result is. This is our main observable measure to differentiate the performance of IGA w.r.t. the other non-interactive approaches. Pr(>F) value refers to the probability of the result, assuming the null hypothesis (i.e. same mean). If one-way ANOVA reports a Pr(>F) value of <0.05, we can reject the null hypothesis that all the data are sampled from populations with the same mean. Roughly, the higher the F-value, the smaller the Pr(>F) value.
Discussion on Second Experiment of RQ2 | Metric: Disagreement

Figure 6.5 shows the experimental results with different settings while assuming user error rate as 0% and the number of elicited pairs are gradually increased from 25 to 100. Here, we show box-plot of disagreement for MON macro scenario (21 req) with 25/50/100 elicited pairs. The key factor of this result is that even if we have minimum number of elicited pair (i.e. 25 pairs which is only 12% of total possible pairs) we have better behavior than non-interactive approaches (i.e. GA and RAND). Also, while we are increasing the number of elicited pairs from 25 to 50 and from 50 to 100 and so on, performance of IGA itself is increasing, which means decreasing disagreement (plotted along the y-axis). When we elicit 25 pairs, the median (Q2) of final disagreement is around 25, after making it 50, the Q2 improves and become 20 and with 100 elicited pairs the median is around 18. On the other hand, median (Q2) of final disagreement using GA is around 30 and the performance of RAND is much higher even than GA (i.e. around 100), both of which are much higher than with 25 pairs elicited using IGA. So, again with another experiment we show that the user interaction plays an important role.
Figure 6.6: Box-Plot of Average Distance with respect to GS for 25, 50 & 100 Elicited pairs for MON (21 req)

Discussion on Second Experiment of RQ2 | Metric: Average Distance

Figure 6.6 shows the experimental results obtained, assuming user error rate assumed 0% and number of elicited pairs are gradually increased from 25 to 100. Unlike Figure 6.5, we put here the box-plot of Average Distance for MON macro scenario (21 req) with 25/50/100 elicited pairs. The key factor of this result is that even if we have the minimum number of elicited pair (i.e. 25 pairs which is only 12% of all pairs) we have indeed a better behavior than the non-interactive approaches (i.e. GA & RAND). Also, while we are increasing the number of elicited pairs from 25 to 50 and from 50 to 100 and so on, performance of IGA itself is increasing i.e. decreasing the average distance (plotted along the y-axis) w.r.t. GS. When we elicit 25 pairs, the median (Q2) of the average distance is around 2.1, with 50, Q2 improves a lot and becomes around 1.8 finally with 100 elicited pairs, the median, Q2 is around 1.5. On the other hand, median (Q2) of average distance using GA is around 2.6, which is much higher than even with 25 pairs elicited. For RAND, Q2 is around 5.8 and certainly worse than GA, even IGA. So, again user interaction plays an important role in requirements prioritization for better performance.
Discussion on Third Experiment of RQ2

From Figure 6.7, an interesting behavioral outcome can be observed. We performed this experiment to show the minimum disagreement for each run w.r.t. the *Gold Standard* using IGA and other non-interactive approaches. We performed this experiment for ALL scenario (49 req) and with 100 elicited pairs while assuming 0% of user error rate. Though most of our experiments consists 20 runs, in this case we limited it to 10 runs for a faster execution. Among the three approaches, clearly RAND (random and non-heuristic ordering) has the worst performance in compared to IGA and non-interactive approach (e.g. GA). RAND has disagreement with the minimum of 500 to maximum of 610 which is very unlikely to be acceptable by software engineers. A non-interactive approach (e.g. GA) has a minimum disagreement of 180 and a maximum of 190. On the other hand, our proposed approach (e.g. IGA) has minimum disagreement of 150 and maximum of around 160 w.r.t GS. Another important characteristics that can be noticed from Figure 6.7 is that the minimum disagreement values of non-interactive approaches are even higher than the maximum disagreement value of IGA and in particular there is a big difference, of about 20-30; this difference occurs because of the advantage of user knowledge during the optimization process. Hence, again it can be concluded that the role of interaction has a very positive impact on improving performance in requirements prioritization.

Also, we can observe from Figure 6.7 that RAND has a bigger fluctuations in its behav-
ior than GA and IGA. This is due to complete random orderings of requirements. On the other hand, GA has some behavioral fluctuations due to the absence of user interactions; because without exploiting user knowledge further improvement is not possible, hence variability in performance exists. But using IGA, there is merely any fluctuations in the performance i.e. the performance line is more or less steady. So, IGA is more acceptable w.r.t. other state-of-the-art approaches. For RQ2, we can conclude that User Interaction has a significant role in building a better requirement orderings.

6.2.3 Research Question 3: [Role of Initial Precedence Constraints]

How does the availability of the initial precedence graphs (constraints) affect the performance of IGA i.e. presence of both or any of the precedence graphs between Prio and Dep at the beginning & during the optimization process.

Our hypothesis is that IGA shows a good behavior even when little information is available in terms of precedence constraints produced during the requirement analysis phase from the requirement specification bundles (e.g. priorities, dependencies, costs and other properties related to requirements). This would indicate that IGA is an effectively recommended approach in the circumstances when requirement documents provide insufficient information and in some cases contradictory information, which are not explicitly explained in the requirement documents. In order to test this research question we run augmented experiments in which only a subset of the initial precedence information is considered as constraints (i.e. not the full set of constraints, only partial). More specifically, we prioritized requirements only with Prio or only with Dep as the initial constraints and we compare the results in the three scenarios, where in the first scenario, we have both Prio and Dep; in the second, we only have Prio (not Dep); and in the third scenario only Dep is available (not Prio).

Experimental Settings

To perform the experiment for RQ3 for ALL (49 req), MON (21 req) & FAL (26 req) macro scenarios we used the following settings, shown in Table 6.9 and 6.10.
<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
<tr>
<td>MON (21 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>840 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
<tr>
<td>FAL (26 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>840 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
</tbody>
</table>

Table 6.9: Experimental settings for the RQ3: (part-I)

<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAL (26 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0% error rate</td>
</tr>
</tbody>
</table>

Table 6.10: Experimental settings for the RQ3: (part-II)

**Experiment Execution**

To conduct the experiments for RQ3, we made a total of 20 runs with different number of iterations during the evolutionary process. We captured all the populations, their disagreements, elicited pairs into external data files as well. We calculated disagreement and average distance against final elicited graph and Gold Standard.
Discussion on First Experiment of RQ3: ALL (100 Eli pairs)

In Figure 6.8, we present the experimental results how the initial constrains affect IGA in its performance when they are partially available i.e. in our case either Prio or Dep or both. We plot here the disagreement for ALL macro scenario (49 req) w.r.t. the GS assuming absence of user errors. For IGA we plot three different plots i.e. with both Prio & Dep, only with Prio and only with Dep. We did the same for GA. While considering IGA alone, IGA with Prio & Dep and IGA with only Prio are much better than IGA with Dep. With Dep only, we have a very limited amount of domain knowledge, and through the optimization process it is not that easy to reach a solution close to the GS. It is harder even for the user on whom we are depending to make the real difference i.e. to break the ties. This also indicates the importance of exploiting domain knowledge. For IGA_Prio_Dep median (Q2) is about 115, whereas for IGA_Prio it’s about 120 but for IGA_Dep it is 260. For the GA, we have a similar picture and orderings with Prio-Dep and with only Prio are much better than the ordering with knowledge about Dep only. For GA_Prio_Dep median (Q2) is about 140, whereas for GA_Prio about 130 but for GA_Dep it is 550. Now, if we compare between paired result of IGA & GA, it can be easily found that IGA_Prio_Dep, IGA_Prio & IGA_Dep have higher performance than GA_Prio_Dep, GA_Prio & GA_Dep respectively. More specifically, median (Q2) for these
Discussion on First Experiment of RQ3: MON (50 Eli pairs)

Figure 6.9 shows another experiment on how the initial constraints affect IGA in its performance when they are partially available i.e. in our case either Prio or Dep or both. We have plotted here the disagreement for MON macro scenario (21 req) also w.r.t. the GS assuming the absence of user errors. For IGA, we place three different plots i.e. both with Prio & Dep, only with Prio and only with Dep. We did the same for GA (i.e. a non-interactive approach). When considering IGA alone, IGA with Prio-Dep and IGA with Prio, both are much better than IGA with Dep only. With Dep only, we have a very limited amount of domain knowledge and through the optimization process it’s not that easy to reach to a solution close to the GS. It also indicates the importance of exploiting domain knowledge. For IGA_Prio_Dep median (Q2) is about 20, whereas for IGA_Prio about 24 but for IGA_Dep it is 30. For GA, we have a similar picture, and orderings with Prio-Dep and with Prio only, are much better than the ordering with knowledge on Dep only. For GA_Prio_Dep median (Q2) is about 29, whereas for GA_Prio it’s around 35 but for GA_Dep it is 85. Now, if we compare paired result of IGA & GA, it can be easily observed that IGA_Prio_Dep, IGA_Prio & IGA_Dep have higher performance than GA_Prio_Dep, GA_Prio & GA_Dep respectively. More specifically, median (Q2) for these three cases using IGA are 20, 24 & 30, whereas using GA they are 29, 35 & 85. The availability of different levels of knowledge produces different levels of performance,
hence the amount of knowledge affects the performance of IGA.

Discussion on First Experiment of RQ3: FAL (50 Eli pairs)

In Figure 6.10, we again show the effect of partial availability of initial constrains on IGA performance (i.e. in our case either Prior or Dep or both). We plot here the disagreement for FAL macro scenario (26 req) w.r.t the GS assuming the absence of user errors, when 50 pairs are elicited from the user. For IGA again we plot three different plots i.e. both with Prior and Dep, only with Prior and only with Dep. We did the same for a non-interactive approach, i.e. GA. IGA with Prior-Dep and IGA with Prior both are much better than IGA with Dep only. With Dep only, we have a very limited amount of domain knowledge, and through optimization process it’s hard to reach a good optimization, with such minimal amount of domain knowledge. For IGA_Prio_Dep median (Q2) is about 12, whereas for IGA_Prio it’s around 18 but for IGA_Dep it is 60. For the GA, orderings with Prior-Dep and with Prior only are much better than the ordering with the knowledge about Dep only. For GA_Prio_Dep median (Q2) is about 17, whereas for GA_Prio it’s about 25 but for GA_Dep it is 125. Now, if we compare paired results of IGA & GA, it can be noticed that IGA_Prio_Dep, IGA_Prio & IGA_Dep have higher performance than GA_Prio_Dep, GA_Prio & GA_Dep respectively. More specifically, median (Q2) for these three cases using IGA are 12, 18 & 60, whereas using GA they are 17, 25 & 125. If we compare Figure 6.10 with Figure 6.9 we may wonder why the performance (disagreement) in Figure 6.10 is better than Figure 6.9, even though we have
a higher number of requirements. This depends on the specific parameters & constraints that hold for the two sets of requirements.

Figure 6.11: Box-Plot of Disagreement w.r.t. GS for 100 Elicited Pairs & 26 Reqs. for 0% Error with different constraints

Discussion on First Experiment of RQ3: FAL (100 Eli pairs)

Figure 6.11 shows experimental results on how the initial constraints affect IGA in its performance when they are partially available. Here, we plot the disagreement for FAL macro scenario (26 req) w.r.t. the GS assuming the absence of user errors when 100 pairs are elicited from user. Performance is better than the previous figures as expected from the outcome of RQ2, since more interaction from users implies better performance. For IGA, three different plots i.e. both with Prio and Dep, only with Prio and only with Dep have been plotted. We did the same for a non-interactive approach i.e. GA. IGA with Prio-Dep and IGA with Prio both are much better than IGA with Dep only. Because with Dep only, very little domain knowledge is available, and this is an obstacle for any optimization process. For, IGA_Prio_Dep median (Q2) is about 10, whereas, for IGA_Prio it’s about 13 but for IGA_Dep it is 25. For GA_Prio_Dep median (Q2) is around 16, whereas for GA_Prio about 24 but for GA_Dep it is 120. If we compare paired results of IGA & GA, it can be noticed that IGA_Prio_Dep, IGA_Prio & IGA_Dep have better performance than GA_Prio_Dep, GA_Prio & GA_Dep respectively. More specifically median (Q2) for these three cases using IGA are 10, 13 & 25, whereas, using GA they are 16, 24 & 120. The performance gain in the overall results is due to more elicited pairs, w.r.t. the previous Figure 6.10.
For RQ3, we conclude that the initial precedence graphs contribute to IGA’s performance. Without some initial domain knowledge (precedence constraints) IGA outperforms other non-interactive approaches. It can be mentioned that not all constraints are equally important. Absence of Prior has a higher impact, resulting in worse performance than absence of Dep, which can be compensated by the elicitation process.

6.2.4 Research Question 4: [Robustness]

*Is Interactive Genetic Algorithm robust with respect to errors that are committed by users (i.e. not an ideal user) during the elicitation of pairwise comparisons?*

In order to perform the experiment on the robustness of the proposed algorithm at increasing user error rates, we simulate such errors by means of a simple stochastic model. We first fix a probability of committing an elicitation error, let’s say $p_c$. Then, during the execution of the algorithm, whenever a pairwise comparison is elicited from the user, we generate a response in agreement with the GS with probability $1 - p_c$ and we generate an error (i.e. a user response in disagreement with the GS, that is, in the opposite direction in terms of position w.r.t GS with probability $p_c$. We have varied the probability rate $p_c$ between 5% and 20%. More specifically, if we have a set of elicited pairs of length $n_e$ and we have error rate of $p_c$, in total we will have $tot_{err} = n_e \times p_c$ number of error pairs. So, we randomly select $tot_{err}$ indexes from the $eliOrd$ and during the optimization process whenever we encounter such indexes in $eliOrd$, we elicit the wrong pair.

**Experimental Settings**

To perform the experiments for RQ4 for ALL (49 req) & FAL (26 req) macro scenarios we used the following settings, shown in Table 6.11:
### Table 6.11: Experimental settings for the RQ4

<table>
<thead>
<tr>
<th>Macro scenario</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>840 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0%, 5%, 10% &amp; 20% (error rate)</td>
</tr>
<tr>
<td>ALL (49 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0%, 5%, 10% &amp; 20% (error rate)</td>
</tr>
<tr>
<td>FAL (26 req)</td>
<td>targetAlgorithm</td>
<td>IGA, GA</td>
</tr>
<tr>
<td></td>
<td>measurement</td>
<td>Disagreement</td>
</tr>
<tr>
<td></td>
<td>maxElicitedPairs</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>execTime</td>
<td>1080 seconds</td>
</tr>
<tr>
<td></td>
<td>errorPerc</td>
<td>0%, 5%, 10% &amp; 20% (error rate)</td>
</tr>
</tbody>
</table>

**Experiment Execution**

For running experiments regarding RQ4, we made a total of 20 runs with different number of iterations during the evolutionary process. We captured all the populations, their disagreements, elicited pairs into external data files. We calculated disagreement and average distance against the final elicited graph and Gold Standard.
Results

Figure 6.12: Box-Plot of Disagreement w.r.t. GS for 50 Elicited Pairs & 49 Reqs. with error rates 5% to 20%

Discussion on First Experiment of RQ4: ALL (50 Eli pairs)

Figure 6.12 shows a comparative view using box-plots both for the interactive and non-interactive approach. We plot the ALL macro scenario (49 req), when 50 pairs are elicited from the user, assuming that user is making erroneous decisions with a probability between 0% and 20%. This can be easily achieved by using a parameter of the algorithm, errorPerc. We vary the value of errorPerc from 0% to 20% and show that still IGA is better than other state-of-the-art approaches, which indicates good robustness of IGA. We plot disagreement along the y-axis. While with the error rate 0% IGA has the median (Q2) of 120, with the error rate 5% it is a bit lower i.e. 121 and with 10% and with 20% error rate the median (Q2) is 124 and 125 respectively. Whereas using GA the median (Q2) of the disagreement is still much higher than IGA i.e. 140. If we closely observe Figure 6.12, we can see the range of Lower Quarantile (Q1) and Upper Quarantile (Q3) tends to increase, as well as the range between sample minimum and sample maximum. This reflects the non deterministic behavior of IGA. More user errors are associated with more variability in the outcome of the IGA algorithm. Even when the user makes erroneous decisions up to 20% IGA is still an outperformer.
Discussion on First Experiment of RQ4: ALL (100 Eli pairs)

In Figure 6.13, we show a comparative view using box-plots both for the interactive and non-interactive approach. We plot the ALL macro scenario (49 req), when 100 pairs are elicited from the user, assuming that user is making erroneous decisions with a probability between 0% and 20%. This can be easily achieved in the same way as before by using a parameter of the algorithm, errorPerc. We vary the value of errorPerc from 0% to 20% and show that still IGA is better than other state-of-the-art approaches, which indicates good robustness of IGA. We plot disagreement along the y-axis. We have better performance than in Figure 6.12 (i.e. lower disagreement), as we have a higher number of elicited pairs. While with the error rate 0% IGA has the median (Q2) of 70, with the error rate 5% it is a bit lower i.e. 100 and with 10% and 20% error rate the median (Q2) are 120 and 130 respectively. Whereas using GA the median (Q2) of the disagreement is still much higher than IGA i.e. 140. If we closely observe Figure 6.13, we can see the range of Lower Quarantile (Q1) and Upper Quarantile (Q3) tends to increase as well as the range between sample minimum and sample maximum (some exceptions may happen due to the non-deterministic behavior of IGA). Even when the user makes erroneous decisions up to 20% IGA is still an outperformer, compare to the other approaches.
Discussion on First Experiment of RQ4: FAL (100 Eli pairs)

Figure 6.14 also shows a comparative view using box-plots both for the interactive and non-interactive approach. We plot the FAL macro scenario (26 req), when 100 pairs are elicited from the user, assuming that user is making erroneous decisions with a probability between 0% and 20%. This can be easily achieved in the same way described in the previous section. We vary the values of errorPerc from 0% to 20% and show that still IGA is better than other state-of-the-art approaches, which indicates good robustness of IGA. With the error rate 0% IGA has the median (Q2) of 11, with the error rate 5% it is a bit lower i.e. 14 and with 10% and 20% error rate the median (Q2) are 22 and 21 respectively. Whereas, using GA the median (Q2) of the disagreement is higher than IGA i.e. 19 only when the error rate is 5%. In this case, when we reach an error rate of 10%, GA is outperforming IGA. In this case, IGA cannot afford many wrong decisions from the user (who is not exhibitly the ideal behavior). So, up to the level of 5% error rate, robustness can still be observed. If we closely observe Figure 6.14, we can see the range of Lower Quarantile (Q1) and Upper Quarantile (Q3) is always increasing as well as the range between sample minimum and the sample maximum. Even though user makes erroneous decisions up to 5% (in this case) IGA is still an outperformer of GA.

For RQ4, we tried to establish is that even though users make error during the elicitation period, IGA is still outperforming the other non-interactive approaches. Robustness holds
up to 20% error rate in all cases but the last one, where a smaller error rate (5%) can be tolerated.

6.3 General Discussion on Results

We conducted experiments with all four macro scenarios including various ranges of elicited pairs (i.e. 25, 50 & 100) and error rates (shown in Table 6.2), with a broad range of criteria and constraints. Due to lack of space, we have shown here only a number of results that are significant to us and to the research questions to be answered. IGA converges even though the fitness function is known in its complete form only at the end of the elicitation process (see RQ1). When we are comparing the prioritization produced by the algorithm with GS, IGA outperforms to a large extent GA and RAND, specially when a higher number of pairwise comparisons is carried out (see RQ2), both in terms of final disagreement and average distance. The improvement of IGA is even higher when only a subset of the initial precedence graphs is available (see RQ3). Moreover, the algorithm has a very good performance characteristics in terms of robustness to user errors. Even though some of the elicited pairs are provided in the wrong direction (see RQ4), IGA can tolerate them well (up to 20% error rate)

6.3.1 Comparison with AHP

Analytical Hierarchy Process (AHP) is the most commonly used state-of-the-art approach for requirements prioritization. It requires an exhaustive pairwise comparisons by the expert decision maker. This is often impractical and in our case study it is not quite possible to elicit 1,176 (N*(N-1)/2) pairs for ALL scenario, which is 210, 253 & 325 for MON, ESC & FAL respectively. In case of ALL scenario, with an artificial user (expert decision maker) who makes no error, using AHP would give zero disagreement w.r.t. the GS. By eliciting only 10% of the pairs needed by AHP (in case of ALL scenario) we cannot expect as good results as AHP, but we save a tremendous amount of time and effort. In fact, the final disagreement value returned by IGA is not zero. However, if we closely observe average distance of each requirement from the corresponding position in the GS, the result is still acceptable, with a much smaller number of elicited pairs (3.5 to 4 maximum, whereas this value could be maximum N-1, in this example 48, N := number of requirements). Hence, the cost-benefit value offered by IGA, compared to AHP is extremely appealing.

6.4 Threats to Validity

The main threat to the validity of our case study and all our experiments resides in the External Validity [43]. External validity refers to the possibility to generalize our findings to
requirements sets collected for other systems with different features. To mitigate this threat, we considered four macro scenarios in addition to the complete one and we assumed the same set of requirement but in different groups and with different initial constraints (priorities & dependencies). This diversity of the information and constraints improves the generalization of the shown results.

There is another threat to the validity for our proposed approach that can be stated as Construct Validity [43]. Construct validity is related to the observations we made to test our research hypotheses. To state more specifically, we have used Disagreement and Average Distance (requirement position distance) as the metrics that determine the algorithm’s performance with respect to Gold Standard. Other metrics could have been used to describe what we actually want to answer against the research questions. However, Disagreement is widely used in the prioritization schemes and Average Distance is quite meaningful and explainable for the orderings that are produced as the outcome of any prioritization approach. Another construct validity threat is related to the error model we used to simulate a user who is not ideal. The exploration of a more sophisticated error model deserves future work.
Chapter 7

Conclusion & Future Works

7.1 Conclusion

Requirements prioritization plays an important role in software development. Most state-of-the-art approaches deal only with expert decision making and are non-interactive (e.g. AHP). They can handle requirements prioritization problems for cases where number of requirements is small. Scalability is a major challenge, which most non-interactive approaches fail to meet.

We summarized the non-interactive approaches that have been proposed in the literature in a unified framework in Section 2.5. As our own method, we propose a framework for requirements prioritization, which adopts a unique Elicitation Process based on the acquisition of pairwise preferences. It also exploits domain knowledge that is properly encoded and supplied with the requirement documents. Using that encoded domain knowledge, our method then performs the evolutionary process using the operators of GA (see Section 3.1) and do the optimization. If convergence remains unchanged for few iterations, user involvement is required, as GA fails to improve further. User helps for further optimization by adding more constraints (also properly encoded) in addition to the initial constraints. This is how IGA works.

For the user interaction, we introduced a user model in Section 6.1.1, Table 6.2 that also defines an error model. We did this to simulate a real context, where the user can certainly make mistakes while comparing two requirements. In the following, for each issue we propose a short discussion of the results.

**Does IGA converge with respect to the complete (final) domain knowledge?**

With RQ1 in Section 6.2.1, we have showed that IGA converges w.r.t. the completed (final) elicited graphs as well as w.r.t. the evolving new & initial constraints.

**Does IGA deal with the scalability problem?**

Unlike AHP, where scalability is a big issue, our approach enables a prioritization pro-
cess even over a large set of requirements, and it can incorporate user knowledge even for large sets of requirements.

**Does user interaction has an important effect on the effectiveness of the approach?**

User interaction is the key hypothesis of our proposed approach. We have tested it with RQ2 in Section 6.2.2, where the benefits of user interaction are clearly shown. None of the state-of-the-art approaches includes this advantage.

**Does the degree of availability of domain knowledge affect the prioritization process?**

We have showed that in our approach, the fitness of different requirements orderings depends on the availability of different kinds of domain knowledge. Our approach takes into account this knowledge when comparing a set of requirements, while other existing approaches for prioritizing requirements do not. We have discussed thoroughly the role of domain knowledge in the requirements prioritization process and how it can be exploited within the IGA framework. In particular, we noticed how the performance of the prioritization process behaves according to the availability of domain knowledge. We answered to this question with RQ3 in Section 6.2.3.

**Has the IGA framework been experimented on a real case-study?**

A real case-study, *ACube* was considered for experiments. It was in the early stage of development. This project includes a well organized software engineering process; and thus classification of available domain knowledge was much easier related to software development. We exploited this real case-study to test the effectiveness of IGA for requirements prioritization, using its requirement document set.

**Is our proposed approach robust in terms of User Errors?**

We also experimented our approach to test another important property, i.e. robustness. While answering this question, we compared several non-interactive approaches with our proposed method and had quite good results. In some cases, it outperforms non-interactive approaches even with an error rate between 5% and 20% (in some cases 10%). That means, if we are eliciting 100 pairs, and the user makes 10 or 15 wrong decisions, IGA still does better than the traditional non-interactive approaches.
7.2 Future Works

In our future works, we will conduct more experiments with more alternative settings on other real life case studies to generalize our findings. We also plan to conduct a thorough empirical study with real subjects (i.e. humans) in the role of requirements analysts. We want to refine our proposed algorithm, to seek better performance. We also want to try more elaborate and specific user (error) model. As part of more advanced research, we will find other SE problems where IGA can be applied, to explore new problem domains and to generalize the findings we obtained for IGA.
Appendix A

ACube: Detailed User Requirements

ACube has four different possible macro scenarios according to which the requirement sets has been classified into different categories. In the following sections all the 4 macro scenarios have been presented along with their priorities in the braces and dependencies in a separate table. Also, the Gold Standard is presented here for each categories that were ordered and proposed by the Requirement Engineer of the ACube Project.

A.1 Monitor Scenario

An event for MON macro scenario can be like Figure A.1 below.

Figure A.1: An event for MON macro scenario in ACube
Each requirement has been assigned a priority label by the engineers according to the users’ needs and considering other dependencies related to the development of the ACube project, following revises all the list of requirement of the MON scenario (priorities are within braces):

RT008(30): The system identifies the role of the persons in the scene (health-care operator, doctor, patient)
RT009(20): The system identifies the identity of the persons in the scene
RT010(10): The system identifies the exact coordinates of the person in the center
RT011(20): The system identifies the exact position of a person with respect to some given points
RT012(10): The system identifies the exact position of a person with respect to some objects (static or dynamic)
RT014(10): The system identifies the trajectory of a person in the center
RT015(20): The system identifies the area of the center a patient is used to stay
RT017(10): The system identifies the differences in the posture of the patient
RT018(10): The system identify if a patient remains immobile for a long time
RT019(35): The system learns the way of life of the patients
RT022(10): The system identifies sounds in the room
RT023(40): The system subdivides the center in areas under the control of an operator
RT025(40): The system minimizes the intersection between the areas monitored by the operators
RT031(20): The system infers the kind of event based on the available information
RT033(40): The system identifies the role that has the responsibility to manage the event
RT034(30): The system logs the info related to a particular event (videos, communications)
RT039(60): The operator can browse the working schedule for a given day
RT042(60): The operator can manually modify the fields for the report at the end of the working time
RT044(50): The authorized operator can access the infos of a patient directly browsing the reports from the PC
RT045(50): The authorized operator can access to the infos of a patient also from other devices (such as mobile p.c.)
RT061(10): The system can monitor the biological parameters of the patient

Dependency table:
Fulfillment of one requirement can be dependent on the accomplishment of another requirement, thus following table can be the base of idea how requirements are dependent in the MON scenario:
<table>
<thead>
<tr>
<th>Req. ID</th>
<th>Dependent requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT018</td>
<td>RT017</td>
</tr>
<tr>
<td>RT012</td>
<td>RT011</td>
</tr>
<tr>
<td>RT014</td>
<td>RT011</td>
</tr>
<tr>
<td>RT034</td>
<td>RT010</td>
</tr>
<tr>
<td>RT015</td>
<td>RT014</td>
</tr>
<tr>
<td>RT025</td>
<td>RT023</td>
</tr>
<tr>
<td>RT019</td>
<td>RT034, RT015</td>
</tr>
<tr>
<td>RT044</td>
<td>RT034</td>
</tr>
<tr>
<td>RT045</td>
<td>RT034</td>
</tr>
</tbody>
</table>

Table A.1: Dependency table for MON scenario

*Gold Standard (GS):*  
The order in which the requirements should be released and the release will maximize user needs & minimize costs and efforts in the project development is defined here.

*RT017 RT018 RT022 RT010 RT011 RT012 RT014 RT031 RT061 RT034 RT008 RT015 RT023 RT025 RT009 RT033 RT019 RT044 RT045 RT039 RT042*

### A.2 Escape Scenario

An event for ESC macro scenario can be like Figure A.2 below.
Each requirement has been assigned a priority label by the engineers according to the users’ needs and considering other dependencies related to the development of the ACube project, following revises all the list of requirement of the ESC scenario (priorities are within braces):

RT002(10): The system monitors the public areas of the center
RT003(10): The system monitors, via sensors, the external area of the centers (gardens, gate)
RT007(10): The system identifies the presence of a person in a given area
RT008(15): The system identifies the role of the persons in the scene (health-care operator, doctor, patient)
RT009(15): The system identifies the identity of the persons in the scene
RT010(10): The system identifies the exact coordinates of the person in the center
RT016(10): The system identifies the distance between the patient and the nearest healthcare operator
RT017(15): The system identifies the differences in the posture of the patient
RT020(15): The system identifies a patient exiting from the monitored area
RT023(30): The system subdivides the center in areas under the control of an operator
RT025(30): The system minimizes the intersection between the areas monitored by the operators
RT026(10): The system verifies that the patients are all in the observed area
RT027(15): The system notifies the operator in the competence area
RT028(40): An operator can accept or deny a call
RT030(30): The system is able to find the nearest operator to a given point in the center
RT031(20): The system infers the kind of event based on the available information
RT032(20): The system associates a gravity level to the event
RT033(30): The system identifies the role that has the responsibility to manage the event
RT034(30): The system logs the info related to a particular event (videos, communications)
RT037(40): The operator can ask for the position of another operator
RT038(50): The operator can ask for the intervention of the nearest operator with respect to an event
RT046(50): The operator can ask for an authorization to perform special interventions to the patients
RT047(50): A user can give an authorization to perform special interventions to other operators

Dependency table:
Fulfillment of one requirement can be dependent on the accomplishment of another requirement, thus following table can be the base of idea how requirements are dependent in the ESC scenario:

<table>
<thead>
<tr>
<th>Req. ID</th>
<th>Dependent requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>RT007, RT002, RT001</td>
</tr>
<tr>
<td>RT009</td>
<td>RT007</td>
</tr>
<tr>
<td>RT027</td>
<td>RT007</td>
</tr>
<tr>
<td>RT031</td>
<td>RT034</td>
</tr>
<tr>
<td>RT032</td>
<td>RT034</td>
</tr>
<tr>
<td>RT025</td>
<td>RT007, RT023</td>
</tr>
<tr>
<td>RT030</td>
<td>RT010</td>
</tr>
<tr>
<td>RT037</td>
<td>RT010</td>
</tr>
<tr>
<td>RT038</td>
<td>RT010</td>
</tr>
</tbody>
</table>

Table A.2: Dependency table for ESC scenario

Gold Standard (GS):
The order in which the requirements should be released and the release will maximize user needs & minimize costs and efforts in the project development is defined here.

RT002 RT007 RT026 RT003 RT010 RT016 RT034 RT020 RT008 RT009 RT017 RT027 RT031 RT032 RT023 RT025 RT030 RT033 RT028 RT037 RT046 RT047 RT038
A.3 Fall Scenario

An event for FAL macro scenario can be like Figure A.3 below.

![Figure A.3: An event for FAL macro scenario in ACube](image)

Each requirement has been assigned a priority label by the engineers according to the users’ needs and considering other dependencies related to the development of the ACube project, following revises all the list of requirement of the FALL scenario (priorities are within braces):

- **RT001(10):** The system monitors the private rooms of the center
- **RT002(10):** The system monitors the public areas of the center
- **RT003(20):** The system monitors, via sensors, the external area of the centers (gardens, gate)
- **RT007(10):** The system identifies the presence of a person in a given area
RT008(15): The system identifies the role of the persons in the scene (healthcare operator, doctor, patient)

RT009(15): The system identifies the identity of the persons in the scene

RT010(10): The system identifies the exact coordinates of the person in the center

RT016(10): The system identifies the distance between the patient and the nearest healthcare operator

RT017(15): The system identifies the differences in the posture of the patient

RT018(15): The system identify if a patient remains immobile for a long time

RT020(15): The system identifies a patient exiting from the monitored area

RT023(40): The system subdivides the center in areas under the control of an operator

RT025(40): The system minimizes the intersection between the areas monitored by the operators

RT026(10): The system verifies that the patients are all in the observed area

RT027(30): The system notifies the operator in the competence area

RT028(50): An operator can accept or deny a call

RT030(40): The system is able to find the nearest operator to a given point in the center

RT031(30): The system infers the kind of event based on the available information

RT032(30): The system associates a gravity level to the event

RT033(40): The system identifies the role that has the responsibility to manage the event

RT034(30): The system logs the info related to a particular event (videos, communications)

RT035(50): The operator can ask for the position of another operator

RT038(50): The operator can ask for the intervention of the nearest operator whit respect to an event

RT046(50): The operator can ask for an authorization to perform special interventions to the patients

RT047(50): A user can give an authorization to perform special interventions to other operators

RT061(15): The system can monitor the biological parameters of the patient

Dependency table:
Fulfillment of one requirement can be dependent on the accomplishment of another requirement, thus following table can be the base of idea how requirements are dependent in the FALL scenario:
<table>
<thead>
<tr>
<th>Req. ID</th>
<th>Dependent requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT026</td>
<td>RT003, RT002, RT001</td>
</tr>
<tr>
<td>RT016</td>
<td>RT011</td>
</tr>
<tr>
<td>RT010</td>
<td>RT007, RT002, RT001</td>
</tr>
<tr>
<td>RT009</td>
<td>RT007</td>
</tr>
<tr>
<td>RT018</td>
<td>RT017</td>
</tr>
<tr>
<td>RT027</td>
<td>RT007</td>
</tr>
<tr>
<td>RT034</td>
<td>RT010</td>
</tr>
<tr>
<td>RT031</td>
<td>RT034</td>
</tr>
<tr>
<td>RT032</td>
<td>RT034</td>
</tr>
<tr>
<td>RT025</td>
<td>RT007, RT023</td>
</tr>
<tr>
<td>RT030</td>
<td>RT010</td>
</tr>
<tr>
<td>RT037</td>
<td>RT010</td>
</tr>
<tr>
<td>RT038</td>
<td>RT010</td>
</tr>
</tbody>
</table>

Table A.3: Dependency table for FALL scenario

*Gold Standard (GS):*
The order in which the requirements should be released and the release will maximize user needs & minimize costs and efforts in the project development is defined here.

RT001 RT002 RT007 RT026 RT016 RT010 RT020 RT008 RT009 RT017 RT018 RT061 RT003 RT027 RT034 RT031 RT032 RT023 RT025 RT030 RT033 RT028 RT037 RT046 RT047 RT038

### A.4 All Scenario

Each requirement has been assigned a priority label by the engineers according to the users’ needs and considering other dependencies related to the development of the ACube project, following revises all the list of requirement of the ALL scenario (priorities are within braces):

*RT001(10): The system monitors the private rooms of the center*

*RT002(10): The system monitors the public areas of the center*

*RT004(10): The system monitors the external area of the center (gardens, gates)*

*RT004(30): The system uses the set of sensors as a single multi-functional sensor exploiting reasoning and abstraction*

*RT005(200): The system is able to adapt to the change of sensors in the sensor network*

*RT007(10): The system identifies the presence of a person in a given area*

*RT008(40): The system identifies the role of the persons in the scene (health-care operator, doctor, patient)*
The system identifies the identity of the persons in the scene.
The system identifies the exact coordinates of the person in the center.
The system identifies the exact position of a person with respect to some given points.
The system identifies the exact position of a person with respect to some objects (static or dynamic).
The system identifies the area of the center in an approximated manner.
The system identifies the trajectory of a person in the center.
The system identifies the area of the center a patient is used to stay.
The system identifies the distance between the patient and the nearest health-care operator.
The system identifies the differences in the posture of the patient.
The system identify if a patient remains immobile for a long time.
The system learns the way of life of the patients.
The system identifies a patient exiting from the monitored area.
The system identifies if a patient is exiting form a room.
The system identifies sounds in the room.
The system subdivides the center in areas under the control of an operator.
The system minimizes the intersection between the areas monitored by the operators.
The system verifies that the patients are all in the observed area.
The system notifies the operator in the competence area.
An operator can accept or deny a call.
The system is able to find the nearest operator to a given point in the center.
The system infers the kind of event based on the available information.
The system associates a gravity level to the event.
The system identifies the role that has the responsibility to manage the event.
The system logs the info related to a particular event (videos, communications).
The system logs missing infos for the given event.
The operator can ask for the position of a patient.
The operator can ask for the position of another operator.
The operator can ask for the intervention of the nearest operator with respect to an event.
The operator can browse the working schedule for a given day.
The operator can check a report produced by the system.
The system inserts the validated report into the report at the end of the working time.
The operator can manually modify the fields for the report at the end of the working time.
The operator can fill the report with missing infos.
The authorized operator can access the infos of a patient directly browsing the reports from the PC.
The authorized operator can access to the infos of a patient also from other devices (such as mobile p.c.)
RT046(180): The operator can ask for an authorization to perform special interventions to the patients
RT047(200): A user can give an authorization to perform special interventions to other operators
RT059(200): The system can block the automatic opening of doors on the bases of a particular critical context
RT060(200): The system can adapt itself to work to different environmental conditions
RT061(30): The system can monitor the biological parameters of the patient
RT062(170): The system can extract crucial events from the daily report
RT063(30): The operator can communicate the system the execution of an activity

Dependency table:
Fulfillment of one requirement can be dependent on the accomplishment of another requirement, thus following table can be the base of idea how requirements are dependent in the ALL scenario:
<table>
<thead>
<tr>
<th>Req. ID</th>
<th>Dependent requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT026</td>
<td>RT003, RT002, RT001</td>
</tr>
<tr>
<td>RT027</td>
<td>RT007</td>
</tr>
<tr>
<td>RT010</td>
<td>RT007, RT002, RT001</td>
</tr>
<tr>
<td>RT012</td>
<td>RT011</td>
</tr>
<tr>
<td>RT014</td>
<td>RT011</td>
</tr>
<tr>
<td>RT016</td>
<td>RT011</td>
</tr>
<tr>
<td>RT018</td>
<td>RT017</td>
</tr>
<tr>
<td>RT025</td>
<td>RT007, RT023</td>
</tr>
<tr>
<td>RT009</td>
<td>RT007</td>
</tr>
<tr>
<td>RT015</td>
<td>RT007, RT014</td>
</tr>
<tr>
<td>RT021</td>
<td>RT007, RT001, RT002</td>
</tr>
<tr>
<td>RT019</td>
<td>RT007, RT015, RT034</td>
</tr>
<tr>
<td>RT031</td>
<td>RT034</td>
</tr>
<tr>
<td>RT032</td>
<td>RT034</td>
</tr>
<tr>
<td>RT034</td>
<td>RT010</td>
</tr>
<tr>
<td>RT035</td>
<td>RT034</td>
</tr>
<tr>
<td>RT036</td>
<td>RT010</td>
</tr>
<tr>
<td>RT037</td>
<td>RT010</td>
</tr>
<tr>
<td>RT044</td>
<td>RT034</td>
</tr>
<tr>
<td>RT045</td>
<td>RT034</td>
</tr>
<tr>
<td>RT043</td>
<td>RT034</td>
</tr>
<tr>
<td>RT040</td>
<td>RT034</td>
</tr>
<tr>
<td>RT062</td>
<td>RT034</td>
</tr>
<tr>
<td>RT041</td>
<td>RT034</td>
</tr>
<tr>
<td>RT030</td>
<td>RT010</td>
</tr>
<tr>
<td>RT038</td>
<td>RT010</td>
</tr>
</tbody>
</table>

Table A.4: Dependency table for ALL scenario

**Gold Standard (GS):**
The order in which the requirements should be released and the release will maximize user needs & minimize costs and efforts in the project development is defined here.

`RT001 RT002 RT003 RT007 RT026 RT027 RT010 RT011 RT012 RT014 RT016 RT017 RT018 RT020 RT061 RT004 RT063 RT008 RT023 RT025 RT009 RT015 RT028 RT021 RT022 RT019 RT031 RT032 RT033 RT034 RT035 RT036 RT037 RT044 RT045 RT039 RT042 RT043 RT040 RT062 RT041 RT046 RT047 RT005 RT013 RT030 RT038 RT059 RT060`
Appendix B

IGA Implementation

In Section B.1, we present the complete class diagram for IGA implementation. In Section B.2, we briefly describe all the subroutines that we used in our implementation. And in Section B.3, we present a program model directing the control flow of the implementation between classes and subroutines.

B.1 Class Diagrams

Figure B.1 shows the class diagram for the implementation of the prioritization algorithm and Figure B.2 shows class diagram for the implementation that we did for further data analysis.

IGAMain
This is the principle class of our implementation and contains the main() class that also initiates the prioritization algorithm which contains the main evolutionary loop. The class also initiates the building process of the domain knowledge i.e. building graphs (in our case Prio & Dep). For the user simulation, we also setup the initial environment i.e. user model inside this class.

Algorithm
This class contains the main engine for IGA i.e. it has all the genetic functionalities either invoked or directly implemented inside. All the major global parameters and objects are declared here for feasibility of the implementation. Several major subtasks like calculating disagreement w.r.t. initial precedences i.e. disAgreementCalc, applying genetic operators i.e. crossOverCalc, mutationCalc & samplePopTournamentSelect, user interaction module and calculating disagreement w.r.t. the final precedence graphs i.e. disAgreementElicited are also initiated from this class. In other word, this is the heart class of the overall implementation.

getSize
To get the population size and the number of requirements for each macro scenario from the requirement files this class is used. It also preserves the value of the population size for the use by other classes.
**countTies**

This class is responsible for counting the ties within the population if there is any.

**buildPriorityGraph**

Building the graphs i.e. encoding the domain knowledge in a usable manner in the coding segment is required. This class creates the instances for building priority graph *Prio* from the requirement priority document and is initiated from the main() function as it needs to be performed in the early stage of the prioritization process.

**buildDependencyGraph**

Figure B.1: Class diagram for IGA Implementation of the algorithm part (in Java)
Like the previous class, this class creates the instance for building dependency graph $Dep$ from the priority document and is initiated from the main() function as it also needs to be performed before the prioritization process starts.

**getPriority**
This class deploys a subroutine to get the requirements with their priorities to put in a stack for further use i.e. building priority graph ($Prio$) etc.

**crossOverCalc**
Crossover operator is one of three important GA operators and is performed within this class. It does not have any public variables declared it is just invoked within the Algorithm by creating its instance.

**mutationCalc**
Likewise previous class, mutation operator is another important GA operator of the three and is performed within this class. It also does not have any public variables declared and is invoked by the Algorithm by creating its instance.

**disAgreementElicited**
This class is responsible for calculating disagreement with respect to all the three precedence graphs including the elicited one, $Eli$. It also uses the instance of another class e.g. $minDisAgree$ for finding the minimum disagreement value that we have just computed in the former class.

**disAgreementCalc**
Likewise the previous class, this class is responsible for calculating disagreement with respect to the initial precedence graphs i.e. priority and dependency graphs. Again, it uses the instance of the class $minDisAgree$ for finding the minimum disagreement value that we have just computed in the former class.

**samplePopTournamentSelect**
Among the three GA operators selection is the first operator to apply. And this class is responsible to apply the selection operator to the population. More specifically, we applied Tournament Selection as our selection operator and this class is duly responsible for that action.

**Random**
To generate one or more random number every time we need we simply used a Java built-in API i.e. Random().

**minDisAgree**
This class is responsible to find out the minimum disagreement in the list of disagreement of all individuals in a population. Some other classes also invoke the instance of this class with the same purpose.
**initPopulation**
This class principally responsible for initializing the population just in a randomized order. After reading the requirements through the class `getPriority` and `getSize` for population size, the requirements are just randomly chosen to build a randomized chromosome and in this way, generate number of individuals equals to the population size.

**disAgrElicitedOrder**
Likewise the class `disAgreementCalc`, this class is responsible for calculating disagreement with respect to the *Gold Standard* only.

**disAgrFinalElicitedGraph**
Like the previous class `disAgrElicitedOrder`, this class is responsible for calculating disagreement with respect to the *Final Elicited Graph* i.e. both the initial precedence graphs *Prio & Dep* and *only* the final shaped elicited graph *Eli*.

**checkTies**
This class simply checks for the existence of ties in the population and return *true or false* based on the checked result.
**Figure B.2: Class diagram for IGA Implementation of the analysis part (in Java)**

**analysisMain**
This is the main class for the implementation of the analysis part that creates instances of all the other classes and invoke their methods. It also contains the main() module with no arguments.

**findMindisAgrFinalEli**
This class is responsible for finding the minimum disagreement over the generations w.r.t. only the final shaped elicited graph and other initial constraints (against all the populations that we stored during the optimization iterations). Also, this class finds the local minimum and the global minimum across all populations and all runs.

**findMindisAgreement**
Likewise the previous one, this class is also responsible for finding the minimum disagreement over the generations but w.r.t. the evolving elicited graph and other initial constraints (against all the populations that we stored during the optimization iterations). Also, this class finds the local minimum and the global minimum across all populations and across all runs.
finddisAgrGoldSTD
This class do the same as previous class except disagreement for the Gold Standard.

disAgrGoldSTD
This class holds the average of disagreements against the Gold Standard in each populations and generates a summary.

disAgrFinalEli
Likewise the previous class, this class also holds the average of disagreements against the final shaped elicited graph in each populations and generates a summary.

disAgreement
Also, this class holds the average of disagreements but against the evolving elicited graph along with initial constraints in each populations and generates a summary.

B.2 Function Definitions
We are describing in this section only the user defined functions those are invoked in each class.

<table>
<thead>
<tr>
<th>Function</th>
<th>prioritizationRequirement()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Algorithm.java</td>
</tr>
<tr>
<td>Action</td>
<td>Initialize the population in a randomized order, calculate the disagreement with the initial precedence graphs, apply the selection operator, break the ties using interactive user responses, apply mutation/crossover operator, then finally measure the disagreement with all the precedence graphs including the elicited one. Continue this process until any of the loop termination condition becomes true.</td>
</tr>
<tr>
<td>Input</td>
<td>NUM_REQ, maxElicitedPairs, thresholdDisagreement, topPopulationPerc, eliOrderedPair, startTime, endTime, execTime, populationSIZE, disagreement, Prio as the collection of individuals, elicitedPairs, adjacencyMatrix, eliCitedMatrix.</td>
</tr>
<tr>
<td>Output</td>
<td>a prioritized order with minimum disagreement.</td>
</tr>
<tr>
<td>Function</td>
<td>getPopulationSize()</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Class</td>
<td>getSize.java</td>
</tr>
<tr>
<td>Action</td>
<td>Read the text file and count the total number of requirements then return the population size as N+1 if there are N requirements in the file.</td>
</tr>
<tr>
<td>Input</td>
<td>a text file containing requirements containing priorities</td>
</tr>
<tr>
<td>Output</td>
<td>populationSIZE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>countTotalTies()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>countTies.java</td>
</tr>
<tr>
<td>Action</td>
<td>Set matchFlag as true; compare each requirement pair in two consecutive individuals with the same index and check for the equality. If, they are not equal set matchFlag as false else continue.</td>
</tr>
<tr>
<td>Input</td>
<td>populationSIZE, Prio as the collection of all individuals</td>
</tr>
<tr>
<td>Output</td>
<td>matchFlag true or false (as 1 or 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>buildPrioGraph()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>buildPriorityGraph.java</td>
</tr>
<tr>
<td>Action</td>
<td>Reads the priority of the requirements from the nodesPriority along with the nodes. And then classifying them in the priority category to update the adjacencyMatrix i.e. similar to building a DAG. Of course the requirements are to be parsed to have few more information i.e., the index of the requirement in the matrix or the name of the corresponding nodes in the graph.</td>
</tr>
<tr>
<td>Input</td>
<td>NUM_REQ, nodesPriority, listNodes</td>
</tr>
<tr>
<td>Output</td>
<td>adjacencyMatrix, a matrix representing the priority graph, Prio.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>buildDepGraph()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>buildDependencyGraph.java</td>
</tr>
<tr>
<td>Action</td>
<td>Reads the dependency info of the requirements from the dependency text file along with the nodes. And then order them using pre-position or post-position manner to update the dependencyMatrix i.e. similar to building a DAG. Of course the requirements are to be parsed to have few more information i.e., the index of the requirement in the matrix or the name of the corresponding nodes in the graph.</td>
</tr>
<tr>
<td>Input</td>
<td>a text file containing dependency information of each requirement.</td>
</tr>
<tr>
<td>Output</td>
<td>dependencyMatrix, a matrix representing the dependency graph, Dep.</td>
</tr>
<tr>
<td>Function</td>
<td>Action</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>getPriorityArray()</td>
<td>Read the text file and parse the requirement details i.e. the requirement as the nodes, their priorities and the requirement description (which is occasionally used) this will help building the priority graph.</td>
</tr>
<tr>
<td>performCrossOver()</td>
<td>Using the algorithm of cut-head/fill-in-tail, choose two parents, pick a cut point from first parent and remove the either the former or the later part and then fill-in with the missing requirements from the second individual. Thus we have a new offspring chromosome from those parents.</td>
</tr>
<tr>
<td>performMutation()</td>
<td>Using the requirement-pair-swap algorithm just selects two random positions and swaps the requirements.</td>
</tr>
<tr>
<td>disAgreementFindElicited()</td>
<td>For each individual, calculate disagreement with respect to all precedence graphs including the elicited one, and increase the disagreement counter 1 each time a mismatch is found for either of the graphs.</td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>disAgreementFind()</td>
<td>disAgreementCalc.java</td>
</tr>
<tr>
<td>tournamentSelection()</td>
<td>samplePopTournamentSelect.java</td>
</tr>
<tr>
<td>minDisAgreement()</td>
<td>minDisAgree.java</td>
</tr>
<tr>
<td>initializePopulation()</td>
<td>initPopulation.java</td>
</tr>
<tr>
<td>disAgreementEliOrder()</td>
<td>disAgrElicitedOrder.java</td>
</tr>
<tr>
<td>Function</td>
<td><code>disAgreementElicited()</code></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Class</td>
<td><code>disAgrFinalElicitedGraph.java</code></td>
</tr>
<tr>
<td>Action</td>
<td>After each run, calculate disagreement with respect to all precedence graphs including the elicited one, in the same manner. And store them in the external file for further process.</td>
</tr>
<tr>
<td>Input</td>
<td><code>eliCitedMatrix</code>, <code>NUM_REQ</code>, <code>populationSIZE</code>, <code>disagreement</code>, <code>nodesPriority</code>, <code>pGraphFlag</code>, <code>dGraphFlag</code>, <code>listNodes</code></td>
</tr>
<tr>
<td>Output</td>
<td>a text file with the disagreements for the individuals in each run.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th><code>checkTiesTOPpopulationPerc()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td><code>checkTies.java</code></td>
</tr>
<tr>
<td>Action</td>
<td>Set flag as false; compare each disagreement pair for two consecutive individuals in the disagreement array and check for the equality. If, they are equal set flag as true else continue.</td>
</tr>
<tr>
<td>Input</td>
<td><code>disagreement</code>, <code>populationSIZE</code></td>
</tr>
<tr>
<td>Output</td>
<td>true or false (as 1 or 0)</td>
</tr>
</tbody>
</table>
B.3 Program Model

For the implementation of IGA, the basic program model can be shown as in Figure B.3.
Bibliography


