Published by the IEEE Computer Society
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314

IEEE Computer Society Order Number P4195
BMS Part Number: CFP1099G-PRT
Library of Congress Number 2010933544
ISBN 978-0-7695-4195-2

2nd International Symposium on Search Based Software Engineering

IEEE computer society

2nd International Symposium on

Search Based Software Engineering

Benevento, Italy
7-9 September 2010

sbse

[simula . research laboratory]

THE UNIVERSITY of York

Università degli Studi del Sannio

Proceedings

Second International Symposium on Search Based Software Engineering

SSBSE 2010

Proceedings

Second International Symposium on Search Based Software Engineering

7-9 September 2010 / Benevento, Italy

Editors

Lionel C. Briand and John A. Clark

CPS Conference Publishing Services

Los Alamitos, California

Washington • Tokyo

IEEE computer society

IEEE Computer Society Order Number P4195
BMS Part #: CFP1099G-PRT
ISBN 978-0-7695-4195-2
Library of Congress Number 2010933544

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Bob Werner
Cover art production by Joe Daigle/Studio Productions
Printed in the United States of America by Applied Digital Imaging

IEEE Computer Society
***Conference Publishing Services* (CPS)**
http://www.computer.org/cps

2nd International Symposium on Search Based Software Engineering

SSBSE 2010

Table of Contents

Message from the General Chairs vii
Message from the Program Chairs ix
Organizing Committee x
Program Committee xi
SSBSE 2010 Program Schedule xii
Keynote Speakers xv

Session 1: Landscapes and Representation

Applying Elementary Landscape Analysis to Search-Based Software Engineering 3
Guanzhou Lu, Rami Bahsoon, and Xin Yao
How Does Program Structure Impact the Effectiveness of the Crossover Operator in Evolutionary Testing? 9
Phil McMinn
A Novel Mask-Coding Representation for Set Cover Problems with Applications in Test Suite Minimisation 19
Shin Yoo

Session 2: PhD Papers

Test Case Selection Method for Emergency Changes 31
Fábio de A. Farzat
Sophisticated Testing of Concurrent Programs 36
Zdeněk Letko
An Optimization-based Approach to Software Development Process Tailoring 40
Andréa Magalhães Magdaleno

Session 3: Requirements and Prioritization

Search Based Optimization of Requirements Interaction Management 47
Yuanyuan Zhang and Mark Harman
Using Interactive GA for Requirements Prioritization 57
Paolo Tonella, Angelo Susi, and Francis Palma
Ant Colony Optimization for the Next Release Problem: A Comparative Study 67
José del Sagrado, Isabel María del Águila, and Francisco Javier Orellana

Session 4: Process Management

Search-based Prediction of Fault-slip-through in Large Software Projects 79
Wasif Afzal, Richard Torkar, Robert Feldt, and Greger Wikstrand
Genetic Programming for Effort Estimation: An Analysis of the Impact of Different Fitness Functions 89
Filomena Ferrucci, Carmine Gravino, Rocco Oliveto, and Federica Sarro

Session 5: Testing and Diagnostics

AUSTIN: A Tool for Search Based Software Testing for the C Language and Its Evaluation on Deployed Automotive Systems 101
Kiran Lakhotia, Mark Harman, and Hamilton Gross
A Search-Based Approach to Functional Hardware-in-the-Loop Testing 111
Felix Lindlar and Andreas Windisch
Using Search Methods for Selecting and Combining Software Sensors to Improve Fault Detection in Autonomic Systems 120
Maxim Shevertalov, Kevin Lynch, Edward Stehle, Chris Rorres, and Spiros Mancoridis

Session 6: General and Enablers

Search-based Resource Scheduling for Bug Fixing Tasks 133
Junchao Xiao and Wasif Afzal
The Human Competitiveness of Search Based Software Engineering 143
Jerffeson Teixeira de Souza, Camila Loiola Maia, Fabrício Gomes de Freitas, and Daniel Pinto Coutinho
Concept Location with Genetic Algorithms: A Comparison of Four Distributed Architectures 153
Fatemeh Asadi, Giuliano Antoniol, and Yann-Gaël Guéhéneuc

Author Index 163

Message from the General Chairs

It is our great pleasure to welcome you to the 2nd International Symposium on Search Based Software Engineering, SSBSE 2010.

Search Based Software Engineering (SBSE) reformulates software engineering tasks as optimization problems that can be solved using efficient, modern search algorithms. This is an increasingly attractive approach to software engineering. The use of automated search in today's context of affordable high-performance computing power enables SBSE techniques to solve, in a cost-effective manner, software engineering problems that are often intractable by other means. Moreover, these solutions often prove to be innovative and provide useful insight to the engineer.

With such inherent benefits, it is no surprise that the interest in SBSE techniques continues to flourish and mature, both in academia and in industry. This interest is reflected in the ever-growing number of papers on SBSE published not only in specialized venues, but also increasingly in mainstream, highly-rated conferences and journals. For example, 2010 will see the publication of special journal issues dedicated to SBSE in both *IEEE Transactions on Software Engineering* and *Software: Practice and Experience*. The latter special issue has a particular emphasis on the practical aspects of SBSE, and is an indication that SBSE techniques are now moving out of research environments and into industrial application. The International Workshop on Search-Based Software Testing (SBST) was this year co-located with a highly regarded software engineering conference, the IEEE International Conference on Testing, Verification and Validation (ICST), and the largest conference on evolutionary search, the ACM SIGEVO Genetic and Evolutionary Computation Conference (GECCO), incorporates an SBSE track this year, as it has done since 2002.

The purpose of this symposium, SSBSE, is to support the expanding SBSE community by providing a unique forum where novel – and indeed, revolutionary - ideas in SBSE can be proposed, by facilitating discussions on the directions we are taking as a community, and by welcoming the PhD students who are the newest members of our community.

The inaugural symposium, held in May 2009 at the Cumberland Lodge, Windsor, UK, was a great success that demonstrated not only the wide variety of SBSE topics that are the subject of active investigation, but also the genuinely international nature of the community. The objective of SSBSE 2010 is to build on the foundations laid in 2009 to create an exciting, thought-provoking and enjoyable event this year, and to ensure the continued success of the symposium in future years. Given the support and interest from the SBSE community in this year's symposium, and the quality and number of the papers submitted, we are confident that we will succeed in this objective.

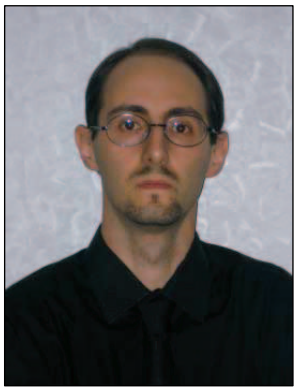
Our venue this year is Benevento, a small city of about 62,000 inhabitants located in southern Italy, between Naples and Rome. Benevento and its province have a very rich history and the city itself has many ancient monuments worthy of a visit. The most famous are the Roman Theater, and the triumphal arch created in honor of the emperor Trajan. We also recommend the historic pedestrian district with many restaurants where you will be able to sample the local

cuisine and wines from the area. We trust that as search-based researchers, you will be able to determine a near-optimal path among these monuments and pursue a multi-objective maximization of both the symposium program and the local attractions!

Organizing SSBSE 2010 has been a collaborative effort involving many people who we would like to acknowledge and thank. The authors: for submitting high-quality papers with many novel and exciting ideas. The program committee members and the additional reviewers: for providing timely, detailed, and constructive feedback, and for actively participating in the online committee discussions. Our renowned keynote speakers, Paolo Tonella and Riccardo Poli, and expert panel members: for agreeing to share their expertise at the symposium. The program chairs, Lionel C. Briand and John A. Clark, the PhD forum chair, Phil McMinn, and the fast abstract chair, Gerardo Canfora: for efficiently managing the extensive review process and for creating an innovative program. The submission and publication chair, Andrea Arcuri: for handling the complexities of the submission process. The publicity chair, Jan Staunton: for keeping the website up-to-date and for disseminating the call for papers, both of which have promoted the event so effectively. Andrea L. Thibault and Bob Werner of the Conference Publishing Services: for their support in publishing the symposium proceedings. Last, but not least, we would like to thank our sponsors, the Province of Benevento (Italy) and the University of Sannio, as well as the other cooperating organizations, the Simula Research Laboratory and the University of York. Without their support, this event would not have been possible.

We look forward to meeting you during the conference, and hope you enjoy the technical and social programs. We certainly will.

Massimiliano Di Penta, University of Sannio
Simon Poulding, University of York
SSBSE 2010 General Chairs
June 2010



Message from the Program Chairs

On behalf of the SSBSE 2009 program committee, we welcome you to the 2nd International Symposium on Search Based Software Engineering.

We are particularly grateful for the widespread participation and support from the SBSE community. 36 papers were submitted to the research and PhD tracks, with a truly international authorship. All submitted papers were reviewed by at least three experts in the field. After extensive discussion, 14 manuscripts were accepted as full papers and 3 were accepted to the PhD track.

A notable feature of this year's symposium is the breadth of problems addressed. Although the initial SBSE community had its origins in software testing, it is now clear that the community has grown to address a significant range of problems in software engineering. The accepted papers address topics as diverse as SBSE landscape analysis, instrumentation optimisation, and prediction tasks in software management. The mix of fundamentals and applied is most welcome, signalling, we believe, the emergence of SBSE as a true discipline.

Maintaining our outward looking mindset, we are delighted to have two outstanding keynote speakers: Riccardo Poli (from the search community) and Paolo Tonella (from the software engineering community).

Above all, we trust that you enjoy SSBSE 2010 and that we will see you again in 2011.

Lionel Briand, Simula Research Laboratories & University of Oslo
John A. Clark, University of York
SSBSE 2010 Program Chairs
June 2010

Organizing Committee

General Chairs
Massimiliano Di Penta, University of Sannio, Italy
Simon Poulding, University of York, UK

Program Chairs
Lionel Briand, Simula Research Laboratory & University of Oslo, Norway
John Clark, University of York, UK

Fast Abstracts Chair
Gerardo Canfora, University of Sannio, Italy

PhD Forum Chair
Phil McMinn, University of Sheffield, UK

Submission and Publication Chair
Andrea Arcuri, Simula Research Laboratory, Norway

Publicity Chair and Website
Jan Staunton, University of York, UK

Program Committee

Giuliano Antoniol, Ecole Polytechnique de Montréal, Canada
Andrea Arcuri, Simula Research Laboratory, Norway
Iain Bate, University of York, UK
Lionel Briand (co-chair), Simula Research Lab. & Univ. of Oslo, Norway
Gerardo Canfora, University of Sannio, Italy
Francisco Chicano, University of Málaga, Spain
John Clark (co-chair), University of York, UK
Myra Cohen, University of Nebraska Lincoln, USA
Vittorio Cortellessa, University of L'Aquila, Italy
Vahid Garousi, University of Calgary, Canada
Walter Gutjahr, University of Wien, Austria
Mark Harman, King's College London, UK
Youssef Hassoun, King's College London, UK
Rob Hierons, Brunel University, UK
Gregory Kapfhammer, Allegheny College, USA
Yvan Labiche, Carleton University, Canada
Spiros Mancoridis, Drexel University, USA
Mel Ó Cinnéide, University College Dublin, Ireland
Martin Shepperd, Brunel University, UK
Jan Staunton, University of York, UK
Paolo Tonella, Fondazione Bruno Kessler - IRST, Italy
Laurence Tratt, Bournemouth University, UK
Tanja Vos, Instituto Tecnológico de Informática, Spain
Joachim Wegener, Berner & Mattner, Germany
Westley Weimer, University of Virginia, USA

Additional Reviewers

Arthur Baars, Universidad Politécnica de Valencia, Spain
AbdulSalam Kalaji, Brunel University, UK

2nd International Symposium on Search Based Software Engineering

Program Schedule – Tuesday, September 7

09:00 – 10:30	**Welcome** **Keynote – Better Together: Hybridized Search Based Techniques** *Paolo Tonella, Fondazione Bruno Kessler, Trento, Italy*
10:30	Refreshments
11:00 – 12:30	**Session 1 – Landscapes and Representation** Applying Elementary Landscape Analysis to Search-Based Software Engineering *Guanzhou Lu, Rami Bahsoon and Xin Yao* How Does Program Structure Impact the Effectiveness of the Crossover Operator in Evolutionary Testing? *Phil McMinn* A Novel Mask-Coding Representation for Set Cover Problems with Applications in Test Suite Minimisation *Shin Yoo*
12:30	Lunch
14:00 – 15:30	**Session 2 – PhD Papers** Test Case Selection Method for Emergency Changes *Fabio Farzat* Sophisticated Testing of Concurrent Programs *Zdeněk Letko* An optimization-based approach to software development process tailoring *Andréa Magdaleno*
15:30	Refreshments
16:00 – 17:30	**Session 3 – Requirements and Prioritization** Search Based Optimization of Requirements Interaction Management *Yuanyuan Zhang and Mark Harman* Using Interactive GA for Requirements Prioritization *Paolo Tonella, Angelo Susi and Francis Palma* Ant Colony Optimization for the Next Release Problem. A comparative study. *José del Sagrado, Isabel María del Águila and Francisco Javier Orellana*

2nd International Symposium on Search Based Software Engineering

Program Schedule – Wednesday, September 8

09:00 – 10:30	**Keynote – Is There a Free Lunch for Human and Search-Based Software Engineering?** *Riccardo Poli, University of Essex, UK*
10:30	Refreshments
11:00 – 12:30	**SSBSE 2011 Announcement** **Session 4 – Process Management** Search-based prediction of fault-slip-through in large software projects *Wasif Afzal, Richard Torkar, Robert Feldt and Greger Wikstrand* Genetic Programming for Effort Estimation: an Analysis of the Impact of Different Fitness Functions *Filomena Ferrucci, Carmine Gravino, Rocco Oliveto and Federica Sarro*
12:30	Lunch
14:00 – 15:30	**Panel Discussion** Can search-based software engineering play an instrumental role in making software engineering technologies scalable and practical?
15:30	Refreshments
16:00 – 17:30	**Fast Abstracts Session**

2nd International Symposium on Search Based Software Engineering

Program Schedule – Thursday, September 9

09:00 – 10:30	**Session 5 – Testing and Diagnostics** AUSTIN: A tool for Search Based Software Testing for the C Language and its Evaluation on Deployed Automotive Systems *Kiran Lakhotia, Mark Harman and Hamilton Gross* A Search-Based Approach to Functional Hardware-in-the-Loop Testing *Felix Lindlar and Andreas Windisch* Using Search Methods for Selecting and Combining Software Sensors to Improve Fault Detection in Autonomic Systems *Maxim Shevertalov, Kevin Lynch, Edward Stehle, Chris Rorres and Spiros Mancoridis*
10:30	Refreshments
11:00 – 12:30	**Session 6 – General and Enablers** Search-based resource scheduling for bug fixing tasks *Junchao Xiao and Wasif Afzal* The Human Competitiveness of Search Based Software Engineering *Jerffeson Souza, Camila Maia, Fabricio Freitas and Daniel Coutinho* Concept Locations with Genetic Algorithms: A Comparison of Four Distributed Architectures *Fatemeh Asadi, Giuliano Antoniol and Yann-Gaël Guéhéneuc*
12:30 – 13:00	**Closing Remarks**
13:00	Lunch

Better Together: Hybridized Search Based Techniques

Paolo Tonella
Fondazione Bruno Kessler, Trento, Italy

Search based software engineering is not necessarily going to replace existing heuristics and solutions to software engineering problems. In fact, the search based framework is flexible enough to allow for smooth integration with several of the existing techniques. Hence, hybridization of search based algorithms with available methods has the potential of a win-win alliance, where the strengths of the various approaches get amplified by the union. Future research in search based software engineering should investigate in depth the challenging opportunities offered by such hybridization.

Is There a Free Lunch for Human and Search-Based Software Engineering?

Riccardo Poli
University of Essex, UK

Informally speaking, the no-free-lunch theory (NFL) originally proposed by Wolpert and Macready states that, when evaluated over all possible problems, all search algorithms are equally good or bad irrespective of our evaluation criteria. In the last 15 years there has been a lot of debate about the consequences of NFL.

In this talk I will first review previous key results on NFL. Then I will present some recent results about the inapplicability of NFL to program and function induction. Finally, I will consider to what extent search-based software engineering (SBSE), computer scientists working on SBSE and human programmers are constrained by NFL.

Applying Elementary Landscape Analysis to Search-Based Software Engineering

Guanzhou Lu, Rami Bahsoon, Xin Yao
School of Computer Science, University of Birmingham
Birmingham B15 2TT, UK
{*G.Lu, R.Bahsoon, X.Yao*}*@cs.bham.ac.uk*

***Abstract*—Recent research in search-based software engineering (SBSE) has demonstrated that a number of software engineering problems can be reformulated as a search problem, hence search algorithms can be applied to tackle it. However, most of the existing work has been of empirical nature and the techniques are predominately experimental. Therefore in-depth studies into characteritics of SE problems and appropriate algorithms to solve them are necessary. In this paper, we propose a novel method to gain insight knowledge on a variant of the next release problem (NRP) using elementary landscape analysis, which could be used to guide the design of more efficient algorithms. Preliminary experimental results are obtained to indicate the effectiveness of the proposed method.**

I. INTRODUCTION

In the current state of Search-based software engineering [1], a wide range of search-based optimisation techniques have been successfully developed and applied to a number of software engineering activities, right across the life-cycle from requirements engineering to software testing [2], including local search, simulated annealing, genetic algorithms, etc. As a new flourishing research area it is being natural that the existing methodology in SBSE has been predominately experimental and lacks in-depth theoretical work except for a few cases [3]. However, to allow the future development of the field requires a deeper understanding of problem and algorithm characteristics.

In order to acheive this goal, fitness landscape characterisation shows a lot of promise. Since no matter what algorithm is employed and what problem is investigated, it is the fitness landscape that captures the nature of the relationship between these two. So far little attention in the literature has concerned the analysis of the fitness landscapes arising from software engineering problems.

There is a special class of fitness landscapes termed "elementary landscapes" [4]. Elementary landscapes possess a unique characteristic where the objective function is an eigenfunction of the graph Laplacian induced by the search operator. This has resulted in several properties, which could be applied implictly or explicitly to design a novel more successful algorithm for a particular problem.

In particular, Whitley et al. [5] observed an interesting consequence that in most practical elementary landscapes studied, the objective function for a particular candidate solution can be written as a linear combination of a subset of a collection of components. A good example is TSP, in which a fitness function is a linear combination of edge weights. As a landscape is not necessarily an elementary landscape, this property enables the construction of elementary landscapes, where both the objective function and the search operator should be appropriately designed as well.

There are a number of software engineering problems sharing the property that the fitness function is linearly decomposable, e.g. a variant of the Next Release Problem (NRP) [6]. In this paper, we propose a methodology to apply the elementary landscape analysis to gain insight knowledge on a variant of NRP. The insight knowledge gained could be used to construct more suited algorithms for this class of problems. We choose to analyze the Sampling Hill Climbing (SHC) algorithm [7], since only a single operator is involved in this algorithm that makes it simple and clear for analysis.

The main contributions of this paper include:

- We develop a method to construct elementary landscapes and carry out the elementary landscape analysis to gain insight knowledge.
- We show how the insight knowledge gained could be applied to design a novel better algorithm.

The remainder of the paper is organised as follows. Section 2 is a brief introduction to elementary landscapes. The problem formulation that we choose to analyze follows in Section 3. In Section 4 we propose the method in detail. Section 5 presents the results from the experiments. Finally, we conclude this paper in Section 6.

II. ELEMENTARY LANDSCAPES

Elementary landscapes [4] are a special class of fitness landscapes, which possess certain properties that could be applied to design novel more successful algorithms. We could characterise software engineering problems that obey certain constraints using elementary landscape analysis, which would give insight knowledge on both selection and design of the algorithms.

A. Fitness Landscapes

The notion of fitness landscapes [8] has been studied extensively in both evolutionary biology and evolutionary optimisation. It has proved to be very powerful in evolutionary optimisation theory, particularly in understanding the

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.10

behaviour of search algorithms for combinatiorial optimisation problems and in predicting their performance.

Formally, the fitness landscape of a problem instance for a combinatorial problem is defined by a triple (X, N, f), where X is a set of candidate solutions, the objective function $f : X \longmapsto \Re$ asigns a real-valued fitness to each point in X and the neighbourhood operator $N : x \longmapsto N(x)$ imposes a neighbourhood structure among X. Given a candidate solution $x \in X, N(x)$ is the neighbourhood set that can be reached by one application of the operator.

B. Elementary Landscapes and Properties

Grover [9] first observed that the landscapes of certain combinatorial optimisation problems such as Travelling Salesman Problem (TSP), could be characterised by a wave equation. Stadler [4] gave a definition to this kind of landscape and named it "Elementary Landscape". A landscape is elementary when the objective function is an eigenfunction of the laplacian of the graph induced by the neighbourhood operator [4].

From a simpler perspective, Whitley et al. [5] provided a wave equation in terms of the expected value of the neighbours, which more concretely expressed the properties of elementary landscapes. Suppose x is some fixed but arbitrary element of X, y is an element drawn uniformly at random from the neighbourhood set $N(x)$ of x and $\overline{f}$ is the mean value over all solutions in X. On an elementary landscape, the following wave equation holds.

$$E[f(y)] = f(x) + \frac{k}{d}(\overline{f} - f(x))$$

for some k which is fixed for the entire landscape. Since y is drawn uniformly at random, the expected value of the fitness value of a neighbour y is always equal to the average fitness value over all solutions in the neighbourhood [5].

Barnes et al. [10] classifed the elementary landscapes in smooth and rugged. The wave equation holds for smooth elementary landscapes has resulted in several properties, which include relative smoothness, constraints on certain plateaus structures and local optima, as well as allowing for predictions about the fitness values of partial or full neighbourhoods during search, etc. In addition, arbitrary fitness landscapes can be decomposed into a superposition of elementary landscapes.

Additionally, landscapes with this property tend to be relatively smooth when contrasted to other combinatorial optimisation problems with well-studied local move operators, which could be considered to be an advantage for local search algorithms [5].

The wave equation also imposes constraints on the structure of local optima and precludes the existence of certain plateaus structure. One of the following observations by Codenotti and Margara [11] is true.

- if $f(x) = \overline{f} \quad f(x) = E[f(y)] = \overline{f}$
- if $f(x) < \overline{f} \quad f(x) < E[f(y)] < \overline{f}$
- if $f(x) > \overline{f} \quad f(x) > E[f(y)] > \overline{f}$

Grover [9] observed similar consequences. Let Z_{min} and Z_{max} be a local minimum and a local maximum, respectively. Then

$$Z_{min} < \overline{f} < Z_{max}$$

In other words, all local minima lie below the average function value of the search space.

Whitley et al. [5] also proved that for a plateau P on a (non-flat) elementary landscape, if $x \in P$ has only equal and disimproving neighbours, then there cannot exist a solution $z \in P$ with only equal and improving neighbors. A plateau is a set P of candidate solutions in X such that for all $a, b \in P$, $f(a) = f(b)$ and there is a path $(a = x1, x2, ..., xk = b)$ such that $x_{i+1} \in N(x_i)$. Plateaus (also known as neutral networks) are structural features that arise in many combinatorial problems [12]. Plateau structure is a challenge for local search that can cause the algorithm to cease progress.

We have seen that the expected fitness value of the full neighbourhood can be predicted by the wave equation. Moreover, we could expand a partial neighbourhood during search, and make predictions for the remaining neighbourhood. This property gives significant insight knowledge to the search algorithm that could be explicitly applied in designing algorithms.

In addition, arbitrary fitness landscapes can be decomposed into a superposition of "elementary landscapes" via a Fourier series expansion. A series expansion $f(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x)$, where φ_i forms a complete and orthonormal system of eigenfunctions of the graph laplacian, is termed a Fourier series expansion of the objective function. This decomposition is helpful in a sense that some statistical properties of the landscape could be computed and the decomposed elementary landscapes can be studied individually. The information about the effective hardness of an elementary landscape is contained in the relative ordering of the associated eigenvalues [13].

C. Component-based Model

Whitley et al. [5] observed several interesting consequences arise from the expected value equation. In most practical elementary landscapes studied, the objective function for a particular candidate solution can be written as a linear combination of a subset of a collection of components. A good example is TSP, in which a fitness function is a linear combination of edge weights. Let C be a set of real valued components and there exists $C_x \subset C$ such that $f(x) = \sum_{c \in C_x} c$. The set C_x is referred to as the intracomponents of a solution x and the set $C - C_x$ as the intercomponents of x. When a local search move has

been made from an incumbent solution x to a neighbouring solution, an exchange of components is made. In particular, a subset of the intracomponents is removed and a subset of the intercomponents is added.

On this basis, Whitley et al. [5] constructed a component-based model that can be used to characterise a neighbourhood structure. In this model, the neighbourhood size is regular and denoted by d. The model consists of the following equations.

$$\overline{f} = p3 \sum_{c \in C} c$$

$$E\{f(y)\} = f(x) - p1 f(x) + p2(\sum_{c \in C} c - f(x))$$

$$= f(x) - p1 f(x) + p2(\frac{1}{p3}\overline{f} - f(x))$$

where $0 < p1 < 1$ is the proportion of the intracomponents that that are removed from the solution in one move, $0 < p2 < 1$ is the proportion of the intercomponents that are added to the solution in a move. Finally, $0 < p3 < 1$ is the proportion of the total components in C that contribute to the cost function for any randomly chosen solution, which is independent of the neighborhood size. Whitley et al. [14] proposed a component theorem:

Theorem 1: If $p1, p2$ and $p3$ (must be constants) can be defined for any regular landscape such that the evaluation function can be decomposed into components where p1 = α/d and p2 = β/d and

$$\overline{f} = p3 \sum_{c \in C} c = \frac{\beta}{\alpha + \beta} \sum_{c \in C} c$$

then the landscape is elementary.

III. THE NEXT RELEASE PROBLEM (NRP)

The Next Release Problem (NRP) was originally formulated by Bagnall et al. [6]. The variant of the NRP studied in this paper is a representative of a class of software engineering problems where the fitness functions could be linearly decomposed. The problem is formulated as follows.

Given a software product, let R denote a set of candidate requirements to be considered to implement for the next release of the software, each $r \in R$ has an associated cost(r) which is a measure of the resouce consumption to implement it. and a weight w_i which reflects the requirement's importance. Also there is a budget for the total cost of the implemented requirements.

Associated with R, there is a directed acyclic graph G = (R,E) where $(r_i, r_j) \in E$ iff r_i is a prerequisite of r_j, G is also transitive since $(r_i, r_j) \in E \wedge (r_j, r_k) \in E \Rightarrow (r_i, r_k) \in E$. If the company decides to satisy requirement r_i, it must satisfy the prerequisites of r_i. In a special case where no requirement has any prerequisite $E = \emptyset$, we say the problem is basic.

Assuming there are n requirements, the problem faced is to find a subset S of R, the cardinality of S is fixed and is k, such that

$$\sum_{r_i \in S} w_i \text{ is maximised,}$$

$$\sum_{r_i \in S} cost(r_i) \text{ is minimised.}$$

Different search algorithms have been applied to NRP [6], but they were all experimental work. There is no analysis of whether the obtained results are good and whether they could be improved. There is no analysis either what characteristics the NRP has and whether the search algorithms used are appropriate.

IV. PROPOSED METHOD FOR ANALYSING SE LANDSCAPES

A. Overview of the Proposed Method

The elementary properties possessed by certain fitness landscapes are promising and could be applied to improve the performance of certain algorithms on particular problems. Initially, a fitness landscape is not necessarily elementary, and thus we will need to modify either the fitness function or the neighbourhood operator to construct an elementary landscape. In addition, to the best of our knowledge, the fitness function should be linearly decomposable in order to enable the construction of an elementary landscape.

In this section, we give a detailed description of our proposed method, with a case study on how the elementary landscape analysis could be applied to the Sampling Hill Climbing (SHC) algorithm on a variant of the Next Release Problem (NRP).

We categorize the elementary properties into two classes. One is implicit, which are inherent given the landscape is elementary and do not affect the design of the algorithm, e.g. relative smoothness. The other one is explicit, which can be explicitly applied while designing algorithms e.g. allow prediction for partial neighbourhoods. Here is an overview of the proposed method.

a. For a given software engineering problem where its fitness function is linearly decomposable, pick up a search algorithm and develop a fitness function and a neighbourhood operator for the selected algorithm.
b. Carry out the elementary landscape analysis.
c. Apply the insight knowledge to the initial algorithm and develop the improved algorithm.
d. Evaluate the performance of the improved algorithm.

B. Initialisation

As a simple but effective local search algorithm, and involving only a single move operator, Sampling Hill Climb-

ing (SHC) is selected as the initial search algorithm. This algorithm works by moving from an inital solution to a local optima providing the move is improving. In each iteration, it simply samples, randomly, a number of solutions from the neighbourhood and take the best of them. The algorithm might get stuck when each of the samples is worse than the current solution, we chose to continue the search by accepting the best move irrespective of whether or not it is improving. The algorithm will terminate after a fixed number of iterations.

In each local search algorithm, there is an objective function to guide the search. According to the problem formulation of a variant of the Next release problem (NRP), the objective function can be defined as

$$f(S) = \sum_{R_i \in S} w_i + Budget - \sum_{R_i \in S} cost(R_i).$$

With respect to the local search operator, the initial choice is 2-exchange, which will randomly exchange two requirements in S and $R - S$.

C. Elementary Landscape Analysis

First of all, we apply the component-based model to determine whether the fitness landscape generated by the initial algorithm is elementary or not.

The objective function defined above is a combination of weights and costs, which is similar to Whitley's observation of components. Let the set $w_i - cost(R_i)$ make up the set of components C, where $|C| = |R|$, we could apply the component-based model and component theorem to prove whether the induced landscape is elementary or not.

We first compute $p3$ and $\overline{f}$, since k components have been picked up to contribute to the objective function.

$$p3 = \frac{k}{|R|}, \quad \overline{f} = p3 \sum_{c \in C} c = \frac{k}{|R|}(\sum_{R_i \in S} w_i - \sum_{R_i \in S} cost(R_i) + |R| * Budget).$$

To compute $p1$ note there are k components in any solution, and two-exchange changes exactly 2 components. Therefore $p1 = 2/k$.

To compute $p2$ note there are $|R| - k$ components with the components in $f(x)$ removed and 2 new components are picked from these.Therefore $p2 = \frac{2}{|R| - k}$.

Adding the terms to the component-based model yields:

$$Avg\{f(y)\} = f(x) - p1 f(x) + p2(\frac{1}{p3}\overline{f} - f(x))$$
$$= f(x) - \frac{2}{k} f(x) + \frac{2}{|R| - k}(\frac{|R|}{k}\overline{f} - f(x))$$
$$= f(x) + \frac{2|R|}{(|R| - k)k}(\overline{f} - f(x))$$

Hence the fitness landscape induced is elementary.

D. Apply the Insight Knowledge

Given the fact that the landscape is elementary, certain explicit elementary properties could be applied in a form of heuristics to replace certain components of the initial algorithm. In the initial Sampling Hill Climbing (SHC) algorithm, it randomly samples N solutions from the neighbourhood. The size of samples has a large impact on the search behaviour - expanding the size is more likely to find an improving move. Since one of the following observations is true for elementary landscapes

- if $f(x) = \overline{f} \quad f(x) = E[f(y)] = \overline{f}$
- if $f(x) < \overline{f} \quad f(x) < E[f(y)] < \overline{f}$
- if $f(x) > \overline{f} \quad f(x) > E[f(y)] > \overline{f}$

When $f(x) < \overline{f} \quad f(x) < E[f(y)]$, one can be sure that a neighbourhood includes an improving move (Assume maximisation). A significantly smaller sample size could have identified an improving move under this circumstance.

When $f(x) > \overline{f} \quad f(x) > E[f(y)]$, one cannot be sure that a neighbourhood includes an improving move, however, elementary properties allow for expanding a partial neighbourhood and predict for the remaining neighbourhood. This prediction is expected to guide the search to a more promising direction and reduce the time wasted in less promising expansions and evaluations. Here is a sketch of the algorithm.

Algorithm 1 Elementary Sampling Hill Climbing.

```
Randomly generate an initial solution x.
for K iterations do
    if f(x) <= f̄ then
        Sample A solutions in the neighbourhood and take the best move among them, A << N;
    else
        Expand a partial neighbourhood of size B, compute the expected value of the remaining neighbourhood.
        if Exp[Remaining neighbourhood] > f(x) then
            Sample C solutions in the remaining neighbourhood and take the best move among them, (B + C) << N;
        else
            Take the best move in the partial neighbourhood;
        end if
    end if
end for
return local optima found;
```

V. COMPUTATIONAL STUDIES

To evaluate the performance of the algorithm incorporated with the insight knowledge obtained by the elementary

Table I
ALGORITHM PARAMETERS

PARAMETERS	VALUES
Number of iterations K	100
Initial neighbourhood size N	10^{-3} * full neighbourhood size
Neighbourhood size A	0.1 * N
Partial neighbourhood size B	0.6 * A
Neighbourhood size C	0.4 * A

landscape analysis, both Elementary Sampling Hill Climbing (ESHC) and Sampling Hill Climbing (SHC) have been implemented in MATLAB. We have carried out an empirical study to pick up the value of different neighbourhood sizes that can find the solution of best quality. Table I presents the algorithm parameters obtained.

To rate the effectiveness of ESHC requires comparison with SHC on different problem instances, in terms of the quality of the best solution found and the time consumed. By varying the size of candidate requirements set $|R|$ and the selected requirements set $|S|$, we have studied 16 problem instances. These synthetic problem instances are randomly generated according to the data sets generator described in [15]. For each problem instance (PI), 50 algorithm runs are performed. The experimental results are listed in Table II, in the first column, the value after slash is the size of candidate requirements set R and the ratio before slash specifies the proportion of R to be selected.

In order to measure the performance of ESHC, we have performed a statistical analysis on the experimental data above using T-test with confidence level at 95%. From which we could show that the algorithm running time of ESHC is significantly less than that of SHC, while there is no significant difference between the quality of the best solution found by both algorithms, on each problem instance studied.

The experimental results show that the Sampling Hill Climbing algorithm incorporated with elementary properties outperforms the initial SHC with far less running time while being able to find roughly the same optimal solution. As described in Section IV, the insight knowledge gained from the elementary landscape analysis can lead the search to focus on promising moves that prevent wasting time in non-promising exploitations. Hence we suppose this is where the performance improvement in time comes from. The experimental results presented in this work are still preliminary, since only one elementary property has been used, which is the predictions for the fitness values of the partial or full neighbourhood. However, if more elementary properties could be appropriately applied to the algorithms, it shows some promise that the elementary landscape analysis can be useful to construct a more suited algorithm for a particular problem, or a class problems sharing certain similarities.

Table II
PERFORMANCE AVERAGES AND STANDARD DEVIATIONS OF 50 RUNS OF ESHC AND SHC ON 16 PROBLEM INSTANCES

	BEST SOLUTION		TIME	
PROBLEM INSTANCE	ESHC	SHC	ESHC	SHC
PI-1 (10%/50)	31.2±0.88	25.8±2.17	0.03±0.0007	0.08±0.003
PI-2 (20%/50)	59.2±2.4	61.9±1.9	0.15±0.0036	0.26±0.0027
PI-3 (50%/50)	117±1.5	118.6±1.56	0.36±0.0077	0.66±0.0015
PI-4 (80%/50)	146.2±2	146.1±2.74	0.15±0.0035	0.26±0.0009
PI-5 (10%/100)	74.76±0.43	72±1.02	0.6±0.0026	1.42±0.0028
PI-6 (20%/100)	137.1±1.1	137.9±0.81	2.43±0.05	4.72±0.025
PI-7 (50%/100)	269.4±0.85	270.3±0.79	6.06±0.15	11.9±0.12
PI-8 (80%/100)	326.1±1.3	327.1±1.1	2.5±0.07	4.78±0.04
PI-9 (10%/200)	149.9±0.76	150.5±0.71	14.1±0.33	28.4±0.12
PI-10 (20%/200)	241.1±0.57	241.7±0.5	43.8±1.03	92.9±0.37
PI-11 (50%/200)	536.6±0.6	536.8±0.5	108.9±2.4	231.3±0.4
PI-12 (80%/200)	370.4±0.55	371±0.71	64.3±0.1	91.7±0.16
PI-13 (10%/500)	357.7±0.48	358.5±0.71	81.65±3.11	178.22±3.70
PI-14 (20%/500)	675.2±0.84	675.4±5.37	241.4±5.37	551.57±1.66
PI-15 (50%/500)	1301.2±1.30	1302±0.45	593.93±12.4	1376.75±17
PI-16 (80%/500)	1601.8±1.30	1604±0.00	249.17±0.23	557.54±1.1

VI. CONCLUSION

In this work we have developed a fitness landscape analysis method to gain insight knowledge on certain software engineering problems. The main goal is to characterise a class of fitness landscapes sharing certain similarities, which would give more insights that could be applied to construct more suited algorithms for particular problems. So far the proposed method is applicable for problems where the objective functions could be linearly decomposed into components.

We carried out a case study to analyze the effectiveness of the proposed method, which is Sampling Hill Climbing (SHC) on a variant of the Next Release Problem (NRP). We found out that if the objective function of a software engineering problem is linearly decomposable, it is possible to construct an elementary landscape and apply elementary properties to design a better algorithm for this problem. The experimental results show that SHC incorporated with elementary properties outperforms the initial algorithm. Therefore we could assume that the performance of algorithms for

a particular problem could be improved by the applicaiton of the elementary landscape analysis.

The future work will include exploiting other elementary properties that could be applied to the algorithms, and extending this method to more software engineering problems.

ACKNOWLEDGEMENT

This work was partially supported by an EPSRC grant (No. EP/D052785/1) on "SEBASE: Software Engineering By Automated SEarch."

REFERENCES

[1] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43:833–839, 2001.

[2] M Harman. The current state and future of search based software engineering. In *FOSE '07: 2007 Future of Software Engineering*, pages 342–357, Washington, DC, USA, 2007. IEEE Computer Society.

[3] P. K. Lehre and X. Yao. Runtime analysis of the (1+1) ea on computing unique input output sequences. *Information Sciences*, 2010.

[4] P. F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, volume 461, pages 77–163, Berlin, New York, 1995. Springer Verlag.

[5] D. Whitley, A. M. Sutton, and A. E. Howe. Understanding elementary landscapes. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 585–592, New York, NY, USA, 2008. ACM.

[6] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883 – 890, 2001.

[7] L.D. Davis. Bit-climbing, representational bias, and test suite design. In *The Fourth International Conference on Genetic Algorithms*, pages 18–23, San Mateo, CA, USA, 1991.

[8] S.Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proc. 6th Congr. Genetics*, volume 1, page 365, 1932.

[9] Lk. Grover. Local search and the local structure of NP-complete problems. *Operations Research Letters*, 12(4):235–243, OCT 1992.

[10] J. W. Barnes, B. Dimova, S. P. Dokov, and A. Solomon. The theory of elementary landscapes. *Applied Mathematical Letters*, 16:337–343, 2002.

[11] B. Codenotti and L. Margara. Local properties of some np-complete problems. Technical report, ICST, 1992.

[12] J Frank, P Cheeseman, and J Stutz. When gravity fails: local search topology. *J. Artif. Int. Res.*, 7(1):249–281, 1997.

[13] P. F. Stadler and G. P. Wagner. Algebraic theory of recombination spaces. *Evol. Comput.*, 5(3):241–275, 1997.

[14] D. Whitley and A. M. Sutton. Partial neighborhoods of elementary landscapes. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 381–388, New York, NY, USA, 2009. ACM.

[15] A Finkelstein, M Harman, S. A Mansouri, J Ren, and Y Zhang. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requir. Eng.*, 14(4):231–245, 2009.

How Does Program Structure Impact the Effectiveness of the Crossover Operator in Evolutionary Testing?

Phil McMinn
University of Sheffield, Department of Computer Science
Regent Court, 211 Portobello, Sheffield, S1 4DP, UK

***Abstract*—Recent results in Search-Based Testing show that the relatively simple Alternating Variable hill climbing method outperforms Evolutionary Testing (ET) for many programs. For ET to perform well in covering an individual branch, a program must have a certain structure that gives rise to a fitness landscape that the crossover operator can exploit. This paper presents theoretical and empirical investigations into the types of program structure that result in such landscapes. The studies show that crossover lends itself to programs that process large data structures or have an internal state that is reached over a series of repeated function or method calls. The empirical study also investigates the type of crossover which works most efficiently for different program structures. It further compares the results obtained by ET with those obtained for different variants of hill climbing algorithm; which are found to be effective for many structures considered favourable to crossover, with the exception of structures with landscapes containing entrapping local optima.**

***Keywords*-Evolutionary Testing, Crossover, Search-Based Test Data Generation**

I. Introduction

Researchers in the field of Evolutionary Computation have devoted much attention to characterising the class of fitness functions for which different search operators perform well. The Royal Road fitness functions, proposed by Mitchell *et al.* [1], [2], were an early attempt to identify the types of fitness landscapes in which crossover worked well. Watson *et al.* [3] later proposed the H-IFF ('Hierarchical If-and-only-if') fitness functions, which are mutation-deceptive and result in Genetic Algorithms with crossover outperforming hill climbing. Later work by Jansen and Ingo Wegener [4] gave rise to the 'Real' Royal Road fitness functions, for which crossover was shown to be provably essential. Conversely, recent work by Richter *et al.* [5] produced the 'Ignoble Trail' fitness functions, which are characterisations of crossover-deceptive landscapes for which crossover is shown to be provably harmful.

The Search-Based Software Engineering community has only begun to investigate theoretical issues relating search-based optimisation to software engineering problems [6], [7], [8], [9], with the aim of developing a clearer understanding of why different techniques may be more effective for different types of problem, and how they might be improved.

This paper concerns the characterisation of programs for which structural test data search by Genetic Algorithms (GAs), referred to as Evolutionary Testing (ET), will perform well. The primary difference between GAs and simpler hill climbing approaches is the ability of a GA to exchange information between candidate solutions in a population, through the crossover operator. Recent studies by Harman and McMinn [8], [9] on a selection of open source and industrial programs revealed that simple hill climbing search in the form of Korel's Alternating Variable Method (AVM) was able to outperform ET in covering the vast majority of branches considered. For the small number of branches for which ET outperformed the AVM, a 'Royal Road'-type property was found to be present. The *constraint-schema* theory for ET was developed, based on the schema theory of binary GAs, and used to characterise Royal Road-type landscapes for ET. However, despite the characterisation of *search landscapes* for which ET is predicted to do well, no clear understanding presently exists regarding the types of *programs and program structures* that give rise to these landscapes.

This paper, therefore, is the first to ask the following research questions:

- What types of programs and program structure enable ET to perform well, through crossover, and how?
- Which form of crossover is best suited to such programs and structures, and why?

This paper addresses the above questions with theoretical and empirical studies. Its primary contributions include:

1. A theoretical analysis of the types of program structure that will favour crossover in ET. The analysis indicates that crossover is likely to perform well when a program has a large number of input variables which discretely impact conditions surrounding the target with a low probability. The types of program likely to benefit from the use of the crossover operator, and therefore search by GAs, are those that process large data structures or have an internal state that is reached by a series of repeated function or method calls.
2. An empirical study involving a number of artificial case studies designed to validate the theoretical study.
3. An empirical study to determine which type of crossover operator works best for different programs. The results show that the operator used by ET is often outperformed by standard uniform crossover.

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.11

4. A further empirical study comparing variants of hill climbing on the programs used to test the effectiveness of crossover; including Random Mutation Hill Climbing (RMHC). As with GA Royal Roads [2], RMHC outperforms ET on landscapes for which crossover performs well, but are free of deception in the form of entrapping, local optima. Program structures do exist with the latter property, for which ET will outperform RMHC.

The paper begins by reviewing important background.

II. Background

A. Search-Based Testing (SBT)

Search-Based Testing (SBT) generates test data through optimisation of a fitness function that underpins the test criterion of interest. This paper concentrates on branch coverage, where each uncovered branch is the focus of an individual test data search. The search space is the input domain of the program under test. The fitness function scores input vectors on the basis of how close they were to executing the branch of interest. The first obstacle for the generated input is to penetrate the layers of statements in which the target branch is nested. The *approach level* [10] measures the number of control graph nodes on which the branch is control dependent, but were not executed in the path taken by a particular input. An example approach level computation is shown in Figure 1. If the path diverges from the target at some control dependency, the *branch distance* calculation is performed using the values of variables appearing in the control dependency's condition. The calculation reflects how close the conditional was to being executed with the alternative, desired outcome. For example, if a condition '`a == b`' needs to be executed as true, the branch distance is found using the formula $|a - b|$ [11]. The overall fitness function to be minimised for a branch target t and for assessing an input vector v is $fit(t, v) = approach\_level(t, v) + normalise(branch\_distance(t, v))$. When the fitness value is zero, t is executed by v. Experiments in this paper use the normalisation function $1 - 1.001^{-d}$ for a distance d.

The **Alternating Variable Method (AVM)** [11] was one of the earliest algorithms used for SBT. An input vector is initially chosen at random. The algorithm then optimises the fitness function for each input variable of the vector in turn. First, an 'exploratory' move is performed by increasing and decreasing the value by a small amount k ($k = 1$ for experiments with integer types in this paper). If an exploratory move results in an improvement in fitness, further 'pattern' moves are made in the direction of improvement with increasing step sizes of 2^i for the i^{th} successive move. Pattern moves continue until fitness fails to improve, where upon exploratory moves are recommenced. If exploratory moves fail to yield improvement, the focus moves to the next input variable in the vector. Moves are made until a target-executing input is found or until exploratory moves have been attempted on each input variable without an improvement in fitness. At this point the search may be restarted with a new random input. The search continues until the target is covered or the pre-determined budget of fitness evaluations has been exhausted.

```
void example1(int a, int b, int c) {
    if (a == b) {
      if (b >= 100) {
        if (c <= 10) {
          // target branch
        }
//...
```

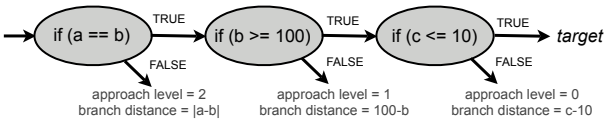


Figure 1. Approach level and branch distance calculation

Evolutionary Testing (ET) is the name given to SBT techniques where Evolutionary Algorithms are used as the optimisation technique. A widely-used GA in the literature is that of Joachim Wegener *et al.* [10], which formed part of DaimlerChrysler's ET System. Instead of using a binary encoding, input vectors are optimised directly with input values forming the 'genes' of each individual chromosome. The overall population of 300 individuals is initially divided equally over 6 competing subpopulations. Individuals are linearly ranked [12] for selection with a pressure Z=1.7, with stochastic universal sampling as the selection mechanism.

Of particular interest to this paper is the use of *discrete recombination* as the crossover operator. Discrete recombination is similar to uniform crossover, but differs in one important aspect. With uniform crossover, each gene is copied into exactly one of the offspring, decided with an even probability. With discrete recombination, however, a gene may be copied into both children, one child, or neither child; with an equal probability.

The Breeder GA [13] mutation operator is used and applied at an inverse of chromosome length. Genes are mutated by the addition or subtraction of values chosen from a range decided by the subpopulation in which the individual currently resides. The range for a subpopulation p, $1 \leq p \leq 6$, is 10^{-p}. Elitist reinsertion is applied, with the top 10% of the current generation retained, while the remaining individuals are discarded and replaced with the best offspring. A progress value, $p_g = 0.9 \cdot p_{g-1} + 0.1 \cdot r$, is computed for each subpopulation at the end of the g^{th} generation, where r is the subpopulation's average ranked fitness following linear ranking of its individuals using $Z = 1.7$. After every 4^{th} generation, subpopulations are ranked according to their progress value and a new share of the overall population computed for each, with weaker subpopulations transferring individuals to stronger ones. A

subpopulation cannot lose its last 5 individuals, preventing its extinction. Finally, individuals migrate every 20^{th} generation, with subpopulations exchanging 10% of their individuals at random.

B. Evolutionary Testing Constraint-Schemata

Harman and McMinn [8], [9] introduced the concept of *constraint-schemata* in order to develop a formal understanding of the crossover operator for ET. The notion of a constraint-schema is a generalisation of Holland's binary GA schema [14]. Whereas binary GA schemata are 'templates' denoting the possible chromosomes that the schema may instantiate, constraint-schemata represent explicit sets of chromosomes, defined in terms of constraints over the input variables of a program. Example constraint-schemata for the program of Figure 1, for example, include $P_1 = \{(a, b, c) \mid a = b\}$ and $P_2 = \{(a, b, c) \mid a = 0\}$. Constraint-schemata, as with their binary predecessors, allow for reasoning about the fitness of chromosomes that may be prevalent in the current population of the GA. For example, chromosomes (input vectors) belonging to P_1 will have a higher average fitness than those belonging to P_2, since P_1 defines the set of chromosomes traversing the initial approach level en route to the target. The schema theory [14] (and its ET generalisation [9]) predicts schemata of above average fitness will proliferate in successive generations of the search.

Harman and McMinn show that the crossover operator will work well for ET when the test data generation problem has a structure such that chromosomes of simpler constraint-schemata can be recombined to produce chromosomes of more specific schemata with higher average fitness. This is effectively a restating of the building block hypothesis for ET. Higher fitness, specific schemata are modelled through the conjunction of the constraints of more general schemata. For example, with the program of Figure 2, chromosomes belonging to $Q_1 = \{(a, b, c) \mid a = b\}$ may be recombined with those of $Q_2 = \{(a, b, c) \mid c \leq 10\}$ to produce chromosomes belonging to $Q = \{(a, b, c) \mid a = b \wedge c \leq 10\}$. Q has a higher average fitness than either Q_1 or Q_2, as the value of `count` will be at least 2, as opposed to 1; resulting in a lower and more favourable branch distance at the condition guarding the target. In general, if some schemata $S_1 = \{I \mid c_1\}$ and $S_2 = \{I \mid c_2\}$ are combined to produce a fitter schema $S = \{I \mid c\}$, $c = c_1 \wedge c_2$, S must respect the constraints of the more general schemata from which it was created. That is, $c \Rightarrow c_1$ and $c \Rightarrow c_2$. This is equivalent to stipulating that S is a subset (*subschema*) of S_1 and S_2; *i.e.* $S \subset S_1$ and $S \subset S_2$. Correspondingly, S_1 and S_2 are referred to as *superschemata* of S.

Surprisingly, in a study of programs comprising of 760 different branches in open source and production code, Harman and McMinn [9] found only 8 branches that allowed the crossover operator to work effectively. As such, although constraint-schemata give a clearer picture regarding the type of fitness landscape that will allow crossover to work effectively for ET, the issue of what types of programs result in such landscapes is less well understood. This is the subject under investigation for the remainder of this paper.

```
void example2(int a, int b, int c) {
  int count = 0;
  if (a == b) count ++;
  if (b >= 100) count ++;
  if (c <= 10) count ++;

  if (count == 3) {
    // target branch
    // ...
```

Figure 2. Target branch not nested but reached under the same conditions as the program of Figure 1

III. A Theoretical Analysis of Program Structures Favouring the Crossover Operator

This section investigates the factors underlying programs and target structures which will allow the crossover operator to work effectively in test data searches. The foundation for this is the concept of constraint-schemata introduced in the last section.

A. Number of conjuncts in the input condition

The *input condition* for covering a structural target in a program, such as a branch, is a constraint over a program's input variables that describes when the target will be executed. The input condition for the target of the program of Figure 2, for example, is $a = b \wedge b \geq 100 \wedge c \leq 10$. The input condition is equivalent to the constraint of the schema that describes the chromosomes (input vectors) that will cover a target structure:

Definition 1 (Covering Constraint-Schema)**.** *A constraint-schema S is said to be the* covering constraint-schema *for a target t if all chromosomes of S execute t (and there do not exist superschemata of S for which this is also true).*

Depending on the structure of the program concerned, the covering constraint-schema may be generalisable into a number of distinct superschemata; the constraints of which denote simpler sub-test data generation problems that must be solved individually in order to reach the final solution (a target-executing input vector). A schema that defines a set of chromosomes which make a step towards the test goal is said to be *fitness-affecting*:

Definition 2 (Fitness-affecting Constraint-Schema)**.** *Let w be the worst fitness value for a structure t in a program p obtained by an input vector drawn from p's input domain. A constraint-schema S is* fitness-affecting *if there does not exist some chromosome $l \in S$ where $fit(l, t) = w$.*

The simplest, most general fitness-affecting schemata encapsulate the building blocks of the test data generation problem.

Definition 3 (Building Block Constraint-Schema)**.** *Let vars(S) be the set of input variables involved in the constraint c of a constraint-schema $S = \{I \mid c\}$. Recall w from Definition 2. A constraint-schema $S_1 = \{V \mid c_1\}$ is said to be a* building block constraint-schema *if it is fitness-affecting (Definition 2) and for all superschemata S_2 of $S_1, S_2 = \{V \mid c_2\}, c_1 \Rightarrow c_2, vars(S_1) \supset vars(S_2)$, there exists some chromosome $l \in S_2, fit(l,t) = w$.*

For example, the covering schema $\{(a,b,c) \mid a = b \wedge b \geq 100 \wedge c \leq 10\}$ for the target of the program of Figure 2, may be generalised into three distinct building block schemata: $\{(a,b,c) \mid a = b\}$, $\{(a,b,c) \mid b \geq 100\}$ and $\{(a,b,c) \mid c \leq 10\}$.

The covering constraint-schema for a target must be generalisable into at least two distinct building blocks in order for crossover to have the opportunity to do any work and contribute to the progress of an ET search. In practice, this is still not enough to guarantee that crossover will have any discernible effect in progressing the search; chromosomes of the covering constraint-schema may be more easily discoverable through mutation. However, it follows that the larger the number of distinct building block constraint-schemata inherent in a test data generation problem the more *opportunity* crossover has to positively impact the progress of the search. This is because there is greater scope for crossover to arrive at the covering constraint-schema through the recombination of building block constraint-schemata; potentially via intermediate schemata that act as stepping stones between building blocks and final solutions.

The number of building block constraint-schemata inherent in the test data generation problem is closely linked to the target's input condition. It follows that the larger the number of conjuncts it has, the larger the number of building blocks may be involved in arriving at a solution.

B. Input condition conjuncts over disjoint sets of variables

If two chromosomes are recombined from two constraint-schemata that reference different input variables in their respective constraints, more specific constraint-schema may be reached by simply copying the genes of input variables referenced in the constraints of each of the original schemata. The constraint-schemata $\{(a,b,c) \mid a = b\}$ and $\{(a,b,c) \mid c \leq 10\}$ for the program of Figure 2, for example, and the input vectors $<a = 10, b = 10, c = 105>$ and $<a = 50, b = 20, c = 5>$ belonging to the former and latter schemata respectively, may be crossed over to produce the offspring $<a = 10, b = 10, c = 5>$, a member of the more specific schema $\{(a,b,c) \mid a = b \wedge c \leq 10\}$.

Recombination is more awkward for *contending* constraint-schemata, however, where one or more input variables are referenced in the respective constraints of each schema.

Definition 4 (Contending Constraint-Schemata)**.** *Recall the function vars(S) from Definition 3. Two constraint-schema S_1 and S_2 are said to be* contending *if their constraints reference one or more of the same input variables, i.e. $vars(S_1) \cap vars(S_2) \neq \emptyset$.*

The ability of the crossover operator may be impaired if contending superschemata are inherent in a test data generation problem. The target of the program of Figure 2, for example, involves the contending schemata $R_1 = \{(a,b,c) \mid a = b\}$ and $R_2 = \{(a,b,c) \mid b \leq 0\}$, which both contain references to the variable b in their respective constraints. Crossover of chromosomes of these schemata is not guaranteed to respect the conjunction of their constraints. For example, recombination of $<a = 10, b = 10, c = 5>$ of R_1 and $<a = 50, b = 20, c = 105>$ of R_2 cannot result in offspring that satisfy $a = b \wedge b \leq 0$.

As such, for a target structure to have a fitness landscape easily exploitable by crossover, not only should the number of input condition conjuncts be numerous, but the conjuncts should involve disjoint sets of variables. This further implies that the input vector to the program under test should also be numerous.

C. Difficulty of discovering input variable values

The harder the input values belonging to fitness-affecting constraint-schemata are to discover, the scarcer they will be in any given generation of a test data search. This situation favours the crossover operator, since it has the capability to build larger pieces of a solution from building blocks that are in existence across different chromosomes in the population. The chances of mutation generating solutions containing from all the required building blocks for an individual chromosome are much smaller. However, if the discovery of individual building blocks is too hard, the genetic material will not come into existence in the first place for crossover to then be able to make use of it.

The difficulty of discovering chromosomes belonging to a constraint-schema depends on two factors. The first factor is the probability of generating inputs at random that satisfy the schema's constraint. Some types of constraint are easy to satisfy at random, *e.g.* the constraint $a > b$ for two input variables a and b of type `int`, which has a probability of 0.5 of being satisfied randomly. A constraint such as $a = 0$, however, is harder to satisfy through random generation of values for a as its domain size increases.

The second factor is the shape of the fitness landscape. Where the landscape has a smooth gradient, providing the search with good guidance to inputs satisfying the constraint, the search will easily find the required input values (genes), regardless of the probability of finding inputs by pure chance. Figure 3 shows a program which takes an array as input. When an individual array value is in a certain range, the `count` variable is incremented. When every value of the array is in range, *i.e.* `count` equals the size of the array, the target branch is executed. The first variant of the program increments the `count` variable by a whole amount,

resulting in a fitness landscape with flat steps down to the global optimum, as depicted in part B of the figure. Array elements are found to be in range through the pure trial and error of mutation. As such the ratio of 'in-range' values to the size of the domain of each array element decides the rate of success the mutation operator will have in finding building block genes that contribute to the overall final solution. The alternative version of the program increments `count` by an amount proportional to the nearest in-range value. This feeds into the branch distance calculation of the target branch, and the corresponding fitness landscape (part C of the figure) is instead formed of downward gradients to the global optimum. The target of the program is executed under exactly the same conditions, yet this variant of the program will ultimately be 'easier' for the search than its counterpart.

D. Absence or Limited Use of Nesting and Short-Circuiting

Nested structures and conditions composed using short-circuiting operators are known to cause problems for SBT [15], [16]. This is because the input condition for executing the target is only revealed to the fitness function step-by-step, as each individual condition is satisfied in turn. For the program of Figure 1, for example, inputs cannot be optimised to fulfil the condition $b \geq 100$ until it is reached in the code, *i.e.* the predicate $a = b$ has been satisfied. This is reflected in the fitness function. It is not until $a = b$ is satisfied that later predicates become the subject of the branch distance calculation, and chromosomes close to evaluating them as true are rewarded.

This impacts the crossover operator. Chromosomes satisfying (or close to satisfying) constraint-schemata for 'later' conditionals will not be rewarded by the fitness function, and as such will not proliferate in the population for eventual use in recombination. This can be explained in terms of constraint-schemata and the schema theorem: consider a target $inner$ with input condition c nested inside a structure $outer$ with input condition a. Let b represent the part of the input condition particular to $inner$, *i.e.* $c = a \wedge b$. For crossover to produce inputs to cover $inner$, there must exist constraint-schemata present in the current population $S_a = \{V \mid a\}$ and $S_b = \{V \mid b\}$, which can be combined to produce chromosomes belonging to the required constraint-schemata $S_c = \{V \mid a \wedge b\}$. S_b must contain some chromosome of worst possible fitness, since $\{V \mid \neg a \wedge b\}$, whose chromosomes cannot traverse the initial approach level, is a subschema of S_b. Therefore S_b cannot be fitness-affecting (Definition 2) for $inner$. Consequently, chromosomes of S_b will not be of above average fitness and not prevalent in the population for recombination with chromosomes of S_a. This reasoning can easily be re-phrased for conditions co-joined using the short-circuiting 'and' operator ('`&&`' in C).

Nesting and short-circuiting limit the search to solving one condition at a time, rather than as parallel sub-problems whose solutions may be recombined.

E. Summary Variables and Internal States

Since nesting and short-circuiting can hinder a search, it follows that crossover will be at its most effective if input variables influence one atomic condition guarding a target, rather than being spread over several co-joined or nested conditions. If more than two input variables are to affect one condition, giving rise to multiple building block constraint schemata, those inputs must do so via an intermediate variable. The target branch of Figure 2, for example, is covered under exactly the same circumstances as Figure 1, except that all inputs directly impact one condition guarding the target through the `count` variable. This allows the search to find solutions to each individual sub-condition necessary to cover the target in parallel. This is because input vectors satisfying $b \geq 100$ and $c \leq 10$ are now rewarded by the fitness function regardless of whether other conditions were previously satisfied. The reward comes in the form of a lower branch distance at the branching condition concerned. Crossover may then recombine these input vectors to find an overall solution; *i.e.* a target-executing input vector.

The types of programs that involve large numbers of inputs and utilising intermediate variables, include programs that process large data structures, storing the results in 'summary' variables, or programs with an internal state that is only reached via long sequences of function or method calls. Furthermore, the conditions under which these intermediate values are modified may give rise to coarse landscapes (as with the program of Figure 3A and the landscape in Figure 3B) further favours crossover since the building blocks are more difficult to discover, as discussed in Section III-C.

F. Conclusions of the Theoretical Analysis

In conclusion, the theoretical analysis predicts that for the crossover operator to be most effective in test data searches:

- The overall input condition for a target must be decomposable into a number of sub-problems, whose solutions can be sought in parallel. These partial solutions form building blocks that can be used by crossover. Ideally then, the **target program structure needs to have an input condition composed of several different conjuncts**.
- To avoid clashes of input vector values during crossover, each conjunct should reference different input variables. Therefore the **program needs to have several input variables** to accompany an abundant number of conjuncts.
- Input condition conjuncts cannot be solved in parallel if they are nested, and conjuncts should be relatively hard to solve randomly, suggesting that **inputs affect a single condition guarding the target via internal variables modified under 'special' circumstances**.

Although the theoretical analysis can suggest conditions favourable for crossover, only an empirical study can establish the extent to which they are true. The results of such a study are presented in the next section.

IV. Empirical Study

A. Research Questions

An empirical study was designed to answer a number of research questions to validate the claims of the theoretical analysis presented in the previous section.

Q1: Number and difficulty of input condition conjuncts. The theory predicts that the more conjuncts in the input condition for covering a target, the more crossover will be able to exploit the resulting fitness landscape. Is this the case in practice, and how many conjuncts are required before crossover noticeably impacts the search? The theory also predicts that the greater difficulty of satisfying individual conjuncts, the more beneficial crossover will be to progressing the search. Is this the case in practice, and under what circumstances does this effect occur?

Q2: Input condition conjuncts with non-disjoint sets of variables. The theory predicts that crossover will be less effective for test data generation problems involving 'contending' constraint-schemata. Is this the case in practice?

Q3: Nested conditionals. The theory predicts that nesting prevents input condition conjuncts from being solved in parallel, thus reducing the number of building blocks available to crossover and rendering it less effective. Is this the case in practice?

In addition to those raised by the theoretical analysis, two further research questions were investigated:

Q4: Performance of ET compared to hill climbing. Over a range of different search problems designed to test crossover for ET, how does hill climbing perform? In what situations will ET outperform hill climbing and vice-versa?

Q5: Crossover type. Across a range of different search problems, which type of crossover performs best for ET?

B. Case studies

In order to answer the research questions above, four case studies were designed.

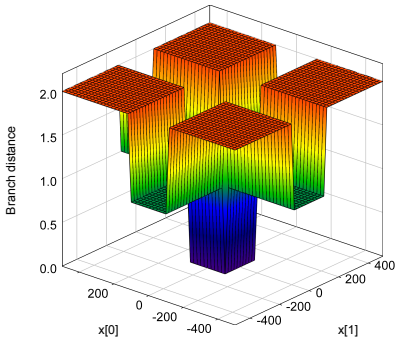
Q1 - Case Study 1 (Figure 3)**.** With this case study, the target branch is executed when every integer in an array is within a certain range. The input conditions conjuncts therefore correspond to individual array elements that are 'in-range', *i.e.* $\{(x[i])|x[i] \geq MIN\_RANGE \wedge x[i] \leq MAX\_RANGE\}$. The satisfaction of each individual conjunct forms an individual building block for the search.

The domain size of each array element is artificially fixed to 1,000 elements (set from -500 to 499). The variables `MIN_RANGE` and `MAX_RANGE` are adjusted to generate the desired level of probability for generating an input satisfying an input condition conjunct; *i.e.* an 'in-range' array element, a building block for the test data search.

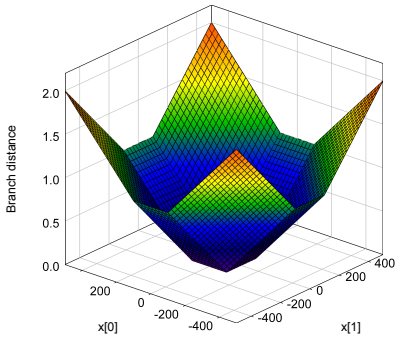
The case study has two variants. When the `count` variable is incremented by whole amounts, a flat fitness landscape results (Figure 3B). In order to investigate the effect of 'easier' landscapes on crossover, a smooth gradient landscape (Figure 3C) can be generated through the inclusion of lines 6-11, where `count` is instead increased by an amount proportional to the distance to the nearest in-range value for a particular array element.

```
(1)   void case_study_1(int x[SIZE]) {
(2)     double count = 0; int i;
(3)     for (i=0; i < SIZE; i++) {
(4)       if (x[i] >= MIN_RANGE && x[i] <= MAX_RANGE)
(5)         count ++;
(6)       else if (x[i] < MIN_RANGE)
(7)         count += 1 - ((MIN_RANGE - x[i]) /
(8)                       (MIN_RANGE - MIN_VAL));
(9)       else
(10)        count += 1 - ((x[i] - MAX_RANGE) /
(11)                      (MAX_VAL - MAX_RANGE));
(12)    }
(13)
(14)    if (count == SIZE) {
(15)        // target branch
(16)    }
(17)  }
```

A. Code



B. Fitness Landscape (lines 6-11 omitted) that is Flat



C. Fitness Landscape (lines 6-11 included) with Downward Gradient

Figure 3. Case study 1: code to assess the impact of different forms of crossover with different numbers of input condition conjunct (through varying array size) and conjunct satisfaction probability (through varying `MIN_RANGE` and `MAX_RANGE`). Lines 6-11 appearing in grey are omitted from the flat landscape version of the experiment, which is pictured for two array elements in part B. Part C depicts the gradient landscape where lines 6-11 are included in the second version of the experiments

Q2 - Case Study 2 (Figure 4)**.** With this case study, the integer elements of the array must be in descending order for the target branch to be executed. The variable `count` keeps track of how many pairs of array elements are in descending order before the branch can be executed. In this way, input condition conjuncts reference one variable appearing in

another; *i.e.* $x[0] > x[1]$, $x[1] > x[2]$. *etc.* As such, the building block schemata are contending. The domain size for each array element is -500 to 499, as for case study 1.

Q3 - Case Study 3 (Figure 5)**.** The target branch of case study 3 is executed under exactly the same circumstances as case study 1, but instead of using a `count` variable, each conjunct of the input condition is nested.

Q4 - Case Study 4 (Figure 6)**.** Case study 4 was deliberately designed as an example to thwart local hill climbing searches. The code is as for case study 1, except that `count` is reset to zero immediately before the array is full of elements that are in-range. This results in a local optimum, as depicted in Figure 6B. ET can overcome the local optimum through crossover, as shown in Figure 6C.

Q5 was assessed using data from each of the above studies.

C. Search Types and Variants Investigated

ET with variants of crossover. ET was applied using the setup described in Section II (with discrete recombination as the crossover operator). To compare ET with and without crossover, ET was also applied without a crossover operator. Parents selected for recombination simply become their offspring. In order to answer Q5, ET was also applied with uniform and one-point crossover.

ET with the Headless Chicken Test (HCT). The 'Headless Chicken Test' (HCT) [17] is used to assess whether GA progress is not merely the result of crossover simply functioning as a macro-mutation operator. The HCT in this paper is ET with discrete recombination, but using a new, randomly-generated individual not drawn from the population, as one of the parents. If ET cannot perform better than the HCT, the search is not actually benefiting from the random exchange of genes between individuals.

Alternating Variable Method (AVM). The AVM was applied as described in Section II.

Random Mutation Hill Climbing (RMHC). With Random Mutation Hill Climbing (RMHC), an input vector is initially selected from the search space at random. Mutations are then by replacing genes with a new value selected at uniform random, with genes mutated at a probability that is the inverse of the chromosome's length. The mutated individual replaces the current individual if it is of improved fitness. When used by Forrest and Mitchell [2], it was found to outperform GAs on Royal Road fitness functions. RMHC is the equivalent of a (1+1) EA.

Random Mutation AVM (RM-AVM). The Random Mutation-Alternating Variable Method (RM-AVM) is a new search novel to this paper, combining the AVM with random mutation re-starts. When the AVM becomes 'stuck', random mutations are made until a better solution is found. RM-AVM therefore incorporates the best features of AVM and RMHC; the ability of the AVM to accelerate down gradients with the ability of RMHC to escape certain local optima.

```
void case_study_2(int x[SIZE]) {
    int count = 0, i;
    for (i=0; i < SIZE-1; i++) {
        if (x[i] > x[i+1])
            count ++;
    }
    if (count == SIZE-1) {
      // target branch
    }
}
```

Figure 4. Case study 2: code to assess the impact of *contending* constraint-schemata. The target branch is executed if the array elements are sorted in descending order. Results with this case study can be found in Table II

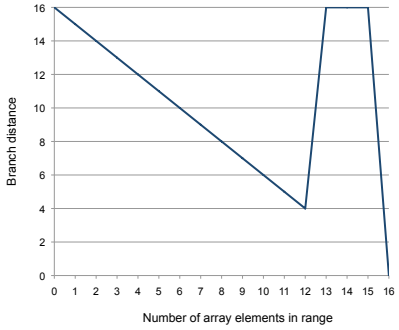
```
void case_study_3(int x[SIZE]) {
  if (x[0] >= MIN_RANGE && x[0] <= MAX_RANGE) {
    if (x[1] >= MIN_RANGE && x[1] <= MAX_RANGE) {
        ...
        // target branch
```

Figure 5. Case study 3: code to assess the impact of crossover on nested structures. The target is executed under exactly the same conditions as case study 1 (Figure 3). Results with this case study can be found in Table III

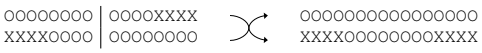
```
void case_study_4(int x[SIZE]) {
  int count = 0, i;
  for (i=0; i < SIZE; i++) {
    if (x[i] >= MIN_RANGE && x[i] <= MAX_RANGE)
      count ++;
  }
  if (count > SIZE-4 && count < SIZE)
    count = 0;

  if (count == SIZE) {
      // target branch
  }
}
```

A. CODE



B. FITNESS LANDSCAPE COLLAPSED TO ONE DIMENSION



C. CROSSOVER OF TWO CHROMOSOMES TO REACH THE GLOBAL OPTIMUM

Figure 6. Case study 4: a branch designed to defeat hill climbing but for which ET can succeed with crossover. Part A shows the program code. All integer values in an array must be in a given range to execute the branch, which uses the variable `count` to record the number of array elements for which this is true. However, just before the global optimum is reached, the value of `count` is set to zero, forming a local optimum depicted in part B, a graph plotting branch distance against the number of array values in range (for an array size of 16). ET can successfully execute the branch through crossover of two individuals at the local optimum as illustrated in part C, where an '`O`' represents an array value in range, while an '`X`' represents a value out of range. Results with this case study can be found in Table IV

Table I

RESULTS FOR THE PROGRAM OF FIGURE 3. AS ARRAY LENGTH INCREASES, SO DO THE NUMBER OF INPUT CONDITION CONJUNCTS AND POTENTIAL BUILDING BLOCK CONSTRAINT-SCHEMATA. THE IMPACT OF CROSSOVER IS GREATER AS THE POTENTIAL NUMBER OF BUILDING BLOCKS INCREASES AND THE PROBABILITY OF THEIR RANDOM GENERATION THROUGH MUTATION DECREASES

Success rate (percentage of times the branch was covered over the 50 runs) is reported unless the branch was covered with 100% success, in which case the average number of fitness evaluations is reported. A figure appears in bold if ET (using discrete recombination) was shown to be significantly better than another search when the one-sided Wilcoxon rank-sum test was applied to numbers of fitness evaluations over the respective sets of 50 trials. Conversely, a figure appears in italics if an alternative search was significantly better than ET. Statistical tests were not performed if the success rate for one of the searches fell below 60% (*i.e.* 30 runs).

A. FLAT FITNESS LANDSCAPE

Build. block probability	Search	Array length 2	4	8	16	32	64	128
0.5	ET	5	14	209	1,247	2,674	4,947	8,660
	ET, uniform crossover	5	14	208	1,228	2,684	*4,882*	*8,062*
	ET, 1-point crossover	5	14	202	**1,498**	**3,902**	**8,985**	48%
	ET, headless chicken test	5	14	215	**4,540**	6%	0%	0%
	ET, no crossover	5	14	2,803	38%	2%	0%	0%
	AVM	**29**	**327**	**10,963**	4%	0%	0%	0%
	RMHC	**6**	14	*56*	*123*	*318*	*775*	*1,819*
	RM-AVM	**22**	**91**	**662**	**2,742**	**13,816**	**98%**	0%
0.2	ET	24	408	1,381	3,103	5,612	9,919	88%
	ET, uniform crossover	24	408	1,417	*2,969*	*5,304*	*8,879*	*14,638*
	ET, 1-point crossover	24	411	**1,713**	**4,339**	**98%**	14%	0%
	ET, headless chicken test	24	431	**7,239**	2%	0%	0%	0%
	ET, no crossover	24	**88%**	24%	0%	0%	0%	0%
	AVM	**214**	**10,709**	0%	0%	0%	0%	0%
	RMHC	21	*74*	*163*	*499*	*1,200*	*2,756*	*6,338*
	RM-AVM	**77**	425	**1,663**	**8,875**	**40,426**	0%	0%
0.1	ET	106	921	2,186	4,339	7,502	96%	10%
	ET, uniform crossover	103	894	2,072	*3,993*	*6,840*	*11,564*	58%
	ET, 1-point crossover	103	919	**2,698**	**6,367**	**70%**	0%	0%
	ET, headless chicken test	106	**1,713**	**29,371**	0%	0%	0%	0%
	ET, no crossover	178	**60%**	6%	0%	0%	0%	0%
	AVM	**617**	56%	0%	0%	0%	0%	0%
	RMHC	*50*	*115*	*394*	*1,027*	*2,702*	*6,116*	14,375
	RM-AVM	**172**	*651*	**3,645**	**17,211**	**78%**	0%	0%
0.01	ET	2,587	94%	78%	46%	2%	0%	0%
	ET, uniform crossover	2,293	94%	*78%*	52%	10%	0%	0%
	ET, 1-point crossover	5,076	94%	**74%**	36%	0%	0%	0%
	ET, headless chicken test	**1,727**	**21,255**	0%	0%	0%	0%	0%
	ET, no crossover	**82%**	16%	0%	0%	0%	0%	0%
	AVM	**86%**	0%	0%	0%	0%	0%	0%
	RMHC	*446*	*1,438*	*4,439*	11,204	27,885	94%	2%
	RM-AVM	1,342	**6,861**	**98%**	16%	0%	0%	0%

B. GRADIENT FITNESS LANDSCAPE

Build. block probability	Search	Array length 2	4	8	16	32	64	128
0.5	ET	5	14	205	1,388	3,366	6,697	12,143
	ET, uniform crossover	5	14	206	1,319	*3,171*	*6,396*	*11,376*
	ET, 1-point crossover	5	14	215	**1,672**	**5,171**	**12,512**	**36,438**
	ET, headless chicken test	5	14	219	**6,025**	0%	0%	0%
	ET, no crossover	5	14	709	**10,324**	**26,387**	**63,863**	0%
	AVM	**11**	**21**	*42*	*86*	*173*	*345*	*693*
	RMHC	**6**	**15**	*56*	*127*	*339*	*787*	*1,980*
	RM-AVM	**11**	**21**	*42*	*86*	*173*	*345*	*693*
0.2	ET	24	450	1,850	4,536	8,540	16,197	33,393
	ET, uniform crossover	24	394	1,942	*4,245*	*8,147*	*14,528*	*26,256*
	ET, 1-point crossover	24	434	**2,142**	**6,105**	**14,808**	**36,349**	**90%**
	ET, headless chicken test	24	431	**10,666**	2%	0%	0%	0%
	ET, no crossover	24	**1,747**	**9,100**	**24,779**	**57,088**	0%	0%
	AVM	19	*38*	*76*	*157*	*318*	*651*	*1,340*
	RMHC	23	*75*	*192*	*529*	*1,283*	*3,030*	*7,120*
	RM-AVM	19	*38*	*76*	*157*	*318*	*651*	*1,340*
0.1	ET	104	1,227	3,460	7,025	14,393	26,569	53,286
	ET, uniform crossover	107	1,114	*3,092*	*6,696*	*12,113*	*22,483*	*43,717*
	ET, 1-point crossover	105	1,188	**4,033**	**10,435**	**22,862**	**50,760**	6%
	ET, headless chicken test	106	**2,216**	**88%**	0%	0%	0%	0%
	ET, no crossover	129	**4,153**	**12,117**	**31,605**	**71,617**	0%	0%
	AVM	*25*	*49*	*98*	*200*	*401*	*805*	*1,624*
	RMHC	*48*	*126*	*390*	*1,289*	*2,818*	*6,652*	*16,164*
	RM-AVM	*25*	*49*	*98*	*200*	*401*	*805*	*1,624*
0.01	ET	1,876	5,255	10,723	19,746	35,979	66,253	0%
	ET, uniform crossover	1,910	5,054	*9,970*	*17,593*	*32,711*	*57,981*	6%
	ET, 1-point crossover	1,788	5,415	**11,988**	**24,792**	**49,411**	**60%**	0%
	ET, headless chicken test	**2,575**	**37,259**	0%	0%	0%	0%	0%
	ET, no crossover	**3,172**	**9,755**	**23,919**	**56,270**	0%	0%	0%
	AVM	*38*	*76*	*152*	*315*	*632*	*1,285*	2,632
	RMHC	*618*	*1,894*	*4,859*	*13,597*	34,947	**82%**	0%
	RM-AVM	*38*	*76*	*152*	*315*	*632*	*1,285*	2,632

D. Experimental setup

Each experiment was run 50 times using an identical list of random seeds. Each search method was given a budget of 100,000 fitness evaluations. If test data were not found within this limit, the search was deemed to have failed. The **success rate** is the reported percentage of the 50 runs for which a search found test data for a branch. Where success rate is 100% the **average number of fitness evaluations** is reported in each table of results. This figure is the mean number of fitness evaluations that were taken to find test data. Statistical tests were performed with the non-parametric Wilcoxon rank-sum test to compare search performance using numbers of fitness evaluations obtained from each of the 50 runs.

E. Answers to Research Questions

Q1: Number and difficulty of input condition conjuncts. Case study 1 (Figure 3) was run with different array sizes; 2, 4, 8, 16, 32, 64 and 128. A domain of -500 to 499 for each array element, coupled with settings of (`MIN_RANGE`, `MAX_RANGE`) as (-250, 249), (-100, 99), (-50, 49), (-5, 4) and (0, 0) respectively, enabled the testing of different probabilities of finding an array element in range from 0.5 down to 0.01. Each setup was run with both variants of the code to produce flat and gradient landscape types.

The results can be found in Table IA. They show that as the length of the array increases (and the number of input condition conjuncts and potential building block constraint-schemata), the search not only becomes harder (as evidenced by higher average numbers of fitness evaluations and lower success rates) but also the impact of crossover increases.

At a building block generation probability of 0.5, ET with crossover is always 100% successful at generating test data. With small array sizes, test data is generated easily at random, the average number of fitness evaluations is not greater than 300, *i.e.* inputs were found within the first generation on average.

For larger array sizes, the HCT and ET without crossover were not always 100% successful. Where they were, ET with crossover had statistically significantly better performance, with the difference in performance becoming greater and in favour of ET with crossover as array size increases. For large array sizes, the HCT and ET without crossover always

Table II
RESULTS FOR THE PROGRAM OF FIGURE 4 WITH 'CONTENDING' CONSTRAINT-SCHEMATA*

Search	Array length								
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ET	22	125	399	1,202	96%	88%	66%	44%	32%
ET, uniform crossover	22	126	391	1,074	2,023	*84%*	78%	48%	38%
ET, 1-point crossover	22	119	385	*977*	*98%*	*96%*	*68%*	56%	36%
ET, headless chicken test	22	122	434	**2,072**	**9,984**	**90%**	**62%**	26%	6%
ET, no crossover	22	896	64%	26%	10%	10%	2%	0%	0%
AVM	**461**	**2,421**	**19,497**	42%	12%	0%	0%	0%	0%
RMHC	**32**	*75*	*198*	*324*	*810*	*2,669*	*4,300*	9,947	12,685
RM-AVM	**229**	**585**	**1,753**	**3,075**	**8,240**	**98%**	**84%**	70%	48%

Table III
RESULTS FOR THE PROGRAM OF FIGURE 5 WITH NESTING*

Build. block prob.	Search	Array length						
		2	**4**	**8**	**16**	**32**	**64**	**128**
0.5	ET	5	14	220	2,308	9,985	52%	0%
	ET, uniform	5	14	200	2,174	*8,334*	76%	0%
	ET, 1-point	5	14	211	**3,014**	**35,493**	0%	0%
	ET, HCT	5	14	221	**8,480**	0%	0%	0%
	ET, no crossover	5	14	1,289	**22,449**	**84%**	0%	0%
	AVM	**11**	**21**	*42*	*86*	*173*	345	693
	RMHC	**6**	**19**	*99*	*302*	*1,312*	5,182	21,281
	RM-AVM	**11**	**21**	*42*	*86*	*173*	345	693
0.01	ET	3,139	14,778	45,149	0%	0%	0%	0%
	ET, uniform	3,102	*12,528*	43,668	0%	0%	0%	0%
	ET, 1-point	3,268	14,056	**52,174**	0%	0%	0%	0%
	ET, HCT	*2,365*	**22,237**	0%	0%	0%	0%	0%
	ET, no crossover	**4,796**	**17,050**	**58,131**	0%	0%	0%	0%
	AVM	*55*	*137*	*394*	1,300	4,598	17,052	65,592
	RMHC	*526*	*2,578*	*10,818*	45,337	2%	0%	0%
	RM-AVM	*55*	*137*	*394*	1,300	4,598	17,052	65,592

Table IV
RESULTS FOR THE PROGRAM OF FIGURE 6 WITH LOCAL OPTIMUM*

Search	Array length			
	16	**32**	**64**	**128**
ET	2,923	4,087	6,201	10,485
ET, uniform crossover	2,912	3,761	6,010	*9,455*
ET, 1-point crossover	**70%**	54%	22%	2%
ET, headless chicken test	**20,236**	0%	0%	0%
ET, no crossover	0%	0%	0%	0%
AVM	4%	0%	0%	0%
RMHC	6%	2%	0%	0%
RM-AVM	0%	0%	0%	0%

*Success rate (percentage of times the branch was covered over the 50 runs) is reported unless the branch was covered with 100% success, in which case the average number of fitness evaluations is reported. A figure appears in bold if ET (using discrete recombination) was shown to be significantly better than another search when the one-sided Wilcoxon rank-sum test was applied to numbers of fitness evaluations over the respective sets of 50 trials. Conversely, a figure appears in italics if an alternative search was significantly better than ET. Statistical tests were not performed if the success rate for one of the searches fell below 60% (*i.e.* 30 runs).

failed to find test data.

As the probability of building block generation decreases, a similar pattern emerges; ET with crossover has an increasingly higher success rate than the HCT and ET without crossover, or, the average number of fitness evaluations for ET with crossover is significantly better, with the difference becoming greater in favour of ET with crossover.

The search becomes easier for all variants of ET when the landscapes has a downward gradient, as seen in part B of Table I, as opposed to when it is flat (part A). However, the effect on crossover follows the same pattern. While average numbers of fitness evaluations are lower (and success rates higher), ET with crossover is increasingly better than the HCT and ET without crossover, or, it achieves an increasingly higher success rate.

The empirical results therefore confirm the theoretical prediction that the higher number of input condition conjuncts, the more useful crossover can be. Also, as predicted, crossover becomes more useful as a search operator the greater the difficulty of generating building blocks.

Q2: Input condition conjuncts with non-disjoint sets of variables. Case study 2 (Figure 4) was run with a domain size of -500 to 499 for each array element (the probability of generating a building block at random is thus fixed at approximately 0.5), with array sizes of 4-12. Table II reports the results. All searches struggle as array size increases, due to the problem becoming more tightly constrained. Despite the presence of contending schemata, ET still outperforms the HCT and ET without crossover with an increasing margin as array size increases (*i.e.* the number of building blocks increase), and is statistically significantly better in many cases.

Thus, in conclusion, contending schemata may reduce the effectiveness of crossover, but the inclusion of the crossover operator may still result in a significantly better evolutionary search.

Q3: Nested conditionals. Case study 3 (Figure 5) was run with the same array sizes and building block generation probabilities as study 1. Table III reports the results for probabilities of 0.5 and 0.01. Although the input condition for reaching the target branch is identical to that of study 1, all searches clearly have difficulty in finding test data, due to nesting. This can easily be seen by comparing the results in Table III with those for study 1 reported in Table I.

Perhaps surprisingly, crossover still has a discernible impact on the search that is statistically significant in many cases. Crossover seems to allow the population to be filled quickly with many 'good' solutions for the current approach level, which increase the probability of it being penetrated through later mutations. When building block probability is low, there are fewer building blocks in existence, and crossover cannot fill the population so quickly. This leads to a smaller margin of increased performance for ET, in contrast with crossover's behaviour when conditionals are not nested (as with the answer to research question 1). At a probability level of 0.01, the aggressive mutation involved in the HCT leads it to significantly outperform ET at the small array size of 2.

In conclusion, although nesting limits the crossover operator, crossover still has a useful role to play in finding test data by causing 'good' genes to proliferate in the population.

Q4: Performance of ET compared to Hill Climbing. As expected, the AVM performs poorly whenever the fitness

landscape is flat, but is significantly superior to all other searches when a gradient exists. The only exception is the descending sort-check branch of case study 2, where the search becomes stuck due to poor starting positions, coupled with an inability to change the value of more than one input at a time. RMHC, on the hand, performs well on flat landscapes and case study 2, and is significantly better when compared to ET. The RM-AVM, being a combination of RMHC and the AVM, always performs somewhere between the two. For studies 1-3, it is always the case that at least one of the hill climbers outperforms ET with crossover.

Case study 4 was specifically designed to contain a local optimum (Figure 6), as a 'proof of concept' that cases can exist where ET will outperform both the AVM and RMHC (whether such code exists for 'real' is another issue). The results are shown in Table IV with domain size as for case study 1 and a building block generation probability of 0.5. As illustrated in Figure 6C, it is possible for ET to reach the global optimum through crossover of two individuals on the edge of the local optimum. Conversely, the chasm between optima is bridged by mutation alone with an extremely low probability, resulting in a low success rate for RMHC.

The conclusion for this research question, therefore, is that test data generation problems for 'crossover-friendly' programs are not necessarily better solved by ET than a hill climber. Cast study 4, however, does show that test data generation problems can exist where ET can find test data, but which are highly challenging for hill climbers.

Q5: Crossover type. Over all of the case studies, uniform crossover is significantly better than discrete recombination on exactly 35 occasions. Conversely, discrete recombination never outperforms uniform crossover. It seems that discrete recombination is responsible for destroying building blocks, slowing down the progress of the search. Whereas with uniform crossover, genes are always preserved in the offspring, discrete recombination may opt to copy a gene into both children and destroy the other.

Discrete recombination usually outperforms one-point crossover, with the exception of four configurations of case study 2 and the sort-check branch. Adjacent array values are dependent on one another, due to 'contending' building block constraint-schemata. As such discrete recombination can destroy some of the local context through the exchange of short sequences of genes, some of which can be preserved by less-disruptive one-point crossover.

In conclusion, the results strongly indicate that uniform crossover is a better choice of crossover operator than discrete recombination for test data generation. One-point crossover is best suited for test data generation problems involving a high degree of cohesion across input variables.

V. Conclusions

This paper has investigated the types of program structure that cause the crossover operator to progress the search for Evolutionary Testing (ET), both theoretically and empirically. The paper found that program structures executed by an input condition with a high number of conjuncts, each of which are hard to satisfy, result in fitness landscapes that are more easily exploitable by crossover than through mutation alone. This lends ET to programs that process large data structures or have internal states reached through sequences of function or method calls. Although nesting hinders a test data search, crossover may still be useful. Hill climbers can also be efficient on these programs, however program structures exist that result in entrapping local optima, for which local searches are not very effective. Finally, it was found that the discrete recombination operator of ET does not represent the best crossover operator for test data searches, as it is frequently outperformed by uniform crossover.

Acknowledgement. Phil McMinn is supported in part by EPSRC grants EP/G009600/1 and EP/F065825/1.

References

[1] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," *Proc. 1st European Conference on Artificial Life*. MIT Press, 1992, pp. 245–254.

[2] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," *Proc. Foundations of Genetic Algorithms*. Morgan Kaufmann, 1993, pp. 109–126.

[3] R. A. Watson, G. S. Hornby, and J. B. Pollack, "Modeling building-block interdependency," *Proc. PPSN V*. Springer, 1998, pp.97–106.

[4] T. Jansen and I. Wegener, "Real royal road functions - where crossover provably is essential," *Discrete Applied Mathematics*, vol. 149, pp. 111–125, 2005.

[5] J. N. Richter, A. Wright, and J. Paxton, "Ignoble trails - where crossover is provably harmful," *Proc. PPSN X*. Springer, 2008, pp. 92–101.

[6] A. Arcuri, P. K. Lehre, and X. Yao, "Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem," *SBST workshop 2008, Proc. ICST 2008*. IEEE, 2008, pp. 161–169.

[7] A. Arcuri, "Longer is better: On the role of test sequence length in software testing," *Proc. ICST 2010*. IEEE, 2010, pp. 469–478.

[8] M. Harman and P. McMinn, "A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation," *Proc. ISSTA 2007*. ACM, 2007, pp. 73–83.

[9] ——, "A theoretical and empirical study of search-based testing: Local, global and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, pp. 226–247, 2010.

[10] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[11] B. Korel, "Automated software test data generation," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 870–879, 1990.

[12] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," *Proc. ICGA 1989*. Morgan Kaufmann, 1989, pp. 116–121.

[13] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I. continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[15] P. McMinn, D. Binkley, and M. Harman, "Empirical evaluation of a nesting testability transformation for evolutionary testing," *ACM Transactions on Software Engineering Methodology*, vol. 3, 2009.

[16] A. Baresel, H. Sthamer, and M. Schmidt, "Fitness function design to improve evolutionary structural testing," *Proc. GECCO 2002*. Morgan Kaufmann, 2002, pp. 1329–1336.

[17] T. Jones, "Crossover, macromutation and population-based search," *Proc. ICGA '95*. Morgan Kaufmann, 1995, pp. 73–80.

A Novel Mask-Coding Representation for Set Cover Problems with Applications in Test Suite Minimisation

Shin Yoo
Centre for Research on Search, Evolution and Testing
King's College London
London, UK
Email: shin.yoo@kcl.ac.uk

***Abstract*—Multi-Objective Set Cover problem forms the basis of many optimisation problems in software testing because the concept of code coverage is based on the set theory. This paper presents Mask-Coding, a novel representation of solutions for set cover optimisation problems that explores the problem space rather than the solution space. The new representation is empirically evaluated with set cover problems formulated from real code coverage data. The results show that Mask-Coding representation can improve both the convergence and diversity of the Pareto-efficient solution set of the multi-objective set cover optimisation.**

***Keywords*-set-cover representation; search-based software engineering; test suite minimisation**

I. Introduction

Multi-Objective Set cover Problem forms an important basis for optimisation problems in software testing. The set cover problem itself is essential to software testing because the concept of code coverage is based on the set theory. For example, the problem of reducing redundancy in test suite can be formulated as a set cover problem [1]–[3]. Multi-Objective approach to set cover problem highlights the trade-offs between the coverage and the cost (often execution time of test cases). Instead of obtaining the maximum set cover, the Multi-Objective Set Cover aims to identify all the maximum coverage possible values for any given budget [4], [5]. While the traditional greedy heuristic is very effective to single-objective set cover problems, its inability to cope with additional objectives facilitated the use of Multi-Objective Evolutionary Algorithms (MOEAs) [5], [6].

Two important factors when applying a meta-heuristic algorithm to a software engineering problem is the fitness function and the representation of a candidate solution [7]. It is known that the choice of the representation of a solution can have a significant impact on the performance of a meta-heuristic algorithm [8], [9]. For example, Rothlauf et al. report that the use of Random NetKey representation reduced the distance to the optimal solution by almost 13% compared to the conventional Characteristic Vector (CV) representation for network design problem [9].

The *de-facto* standard representation for set cover problem is to encode the selection of subsets as a binary string: the ith digit of the binary string is 1 if the test case t_i is included in the solution and 0 otherwise. Following this, the neighbouring solutions for local search algorithms are often defined as solutions with a single digit different from the original solution. Genetic operators work similarly on the binary string representation; cross-over mixes the choice of test cases, while a single bit-flip mutation adds or subtracts a test case. While the bit-string representation is innocuous in itself, it may not be the ideal representation for the set cover problems that deal with code coverage. The empirical study presents some evidence that the bit-string representation may actually be sub-optimal for Multi-Objective Set Cover problem based on code coverage data.

This paper presents a novel representation for set cover problems called Mask-Coding representation. The main idea behind the new representation is to replace the *solution* space with the *problem* space, following the approach of Storer et al. to Job-Shop Scheduling Problem [10]. Mask-Coding representation still uses binary strings, but an instance of Mask-Coding representation would denote an alternative set cover problem in the problem space. The evaluation of this instance would require an efficient and effective domain specific construction heuristic: in case of set cover problem, this would be the greedy algorithm. The solution to the alternative problem, obtained by the construction heuristic, is then evaluated against the original problem.

One potential strength of problem space exploration is that it can be free of the challenges that arise from the features of the solution search landscape, such as a large plateau. The empirical evaluation of the new representation on widely studied code coverage data shows that it can indeed improve the performance of multi-objective meta-heuristic algorithms both in terms of convergence and diversity of the resulting Pareto-front.

The contributions of this paper are as follows:

1) This paper introduces a novel representation for set cover problems, Mask-Coding representation, based on the idea of problem space exploration.
2) The paper presents empirical evidence that the widely used binary string representation may not be ideal for set cover problems based on code coverage data.

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.12

3) The paper presents an empirical evaluation of Mask-Coding representation using multi-objective test suite minimisation problems with real-world coverage data. The results show that the new representation can improve both the convergence and diversity of the results. In the context of software testing, this means more efficient regression testing with more insightful cost-benefit analysis of regression test suites.

The rest of the paper is organised as follows: Section II describes the problems in traditional binary string representation with empirical evidence. Section III introduces Mask-Coding representation and the idea of problem space exploration. Section IV presents the research questions and describes the settings for the empirical study based on test suite minimisation problems, the result of which is discussed in Section V. Section VI presents related work and Section VII concludes.

II. Problems in Traditional Binary String Representation

A. Set Cover Problem

Single-objective set cover problem is NP-hard [11] and can be described as follows: given several sets that share some common elements, the goal is to select the minimum number of these sets so that the selected sets contain all the elements that are contained in any of the input sets. More formally,

Set Cover Optimisation Problem
Given a universe $\mathcal{U}$ of n elements and a family $\mathcal{S}$ of m subsets of $\mathcal{U}$, a *cover* is a subfamily $\mathcal{C} \subseteq \mathcal{U}$ whose union is equal to $\mathcal{U}$. The problem is to find a cover of $\mathcal{U}$ that uses the fewest sets.

It may not be possible to cover $\mathcal{U}$ completely. For example, assume that $\mathcal{U}$ is the set of all program statements in SUT (System Under Test) and $\mathcal{S}$ is the collection of execution traces of test cases: any unreachable code in SUT will not be covered by any combination of traces in $\mathcal{S}$. In this case, the goal of set cover optimisation becomes to achieve the highest coverage possible with the fewest sets. Coverage is defined as the ratio between the size of the cover and the size of $\mathcal{U}$, i.e.:

$$\text{coverage}(\mathcal{C}) = \frac{|\bigcup_{S_i \in C} S_i|}{|\mathcal{U}|}$$

Multi-Objective Set Cover optimisation assigns cost to each set in $\mathcal{S}$ and adopts an additional objective to actively minimise the cost. More formally,

Multi-Objective Set Cover Optimisation Problem
Given a universe $\mathcal{U}$, a family $\mathcal{S}$ of subsets of $\mathcal{U}$ and a cost function $cost : \mathcal{S} \rightarrow \mathbb{R}$, the problem is to find a cover $\mathcal{C}$ that maximises coverage($\mathcal{C}$) and minimises $\sum_{S_i \in C} cost(S_i)$.

While the definition of the multi-objective formulation appears similar to that of the single-objective set cover problem, the Pareto-optimisation [12] of both objectives shows the trade-off between coverage and cost of the set cover, which has application to software testing [5].

B. Binary String Representation and Dimensional Plateau

When applying meta-heuristic optimisation to set cover problem, the most commonly used representation of an individual solution is the binary string representation [4]–[6]. The length of the binary string is equal to the number of subsets in family $\mathcal{S}$. If the member S_i of $\mathcal{S}$ is included in the solution, the ith digit of the binary string is 1; if it is not included, 0.

While the definition of the binary string representation is innocuous in nature, there is a specific problem that arises when it is used for set cover problems based on code coverage data. It is known that different paths in a program get executed with different frequency. For example, the initialisation code will be executed with every execution, while a procedure that deals with a very rare situation, e.g. exception handling code, will be executed less frequently. Since code coverage data represent recorded execution traces, this difference in execution frequency will be reflected in the data. More formally, this means that a significantly large number of members in family $\mathcal{S}$ (i.e. test cases) may cover similar sets of elements in $\mathcal{U}$ that take up the majority of $\mathcal{U}$.

This redundancy in coverage has an important implication for the traditional binary string representation, namely, a large plateau in the coverage dimension. Intuitively, if a large number of members in $\mathcal{S}$ covers largely similar sets of the majority of elements in $\mathcal{U}$, the chance for any mutation on an arbitrary digit i of the binary string representation to make any impact on coverage significantly decreases. This is because there is a high probability that S_j such that $i \neq j, S_j \in \mathcal{S}$ will cover the same or very similar set of elements in $\mathcal{U}$. This would result in a large plateau in the coverage dimension of the search space. This problem will be referred to as the *dimensional plateau problem*:

Dimensional Plateau: in multi-objective search landscape, if one of the objective value remains the same while the other objectives changes, this creates a dimensional plateau for the dimension of the unchanging objective.

When applied to the multi-objective test suite minimisation problem, the existence of a dimensional plateau would mean that the search algorithm may fail to find any solutions with low-coverage and low-cost. If the search algorithm fails to escape the coverage dimensional plateau, it will only optimise the cost of the subset of test cases that will achieve the maximum coverage. Reaching the part of the search

landscape where solutions with lower coverage/lower cost exist may be very challenging.

III. Mask-Coding Representation

A. Problem Space Exploration

The idea of problem space exploration was first introduced by Storer et al. in order to design a new neighbourhood definition for stochastic local search for sequencing problems [10]. A similar idea was also introduced under the name of 'noising method' by Charon et al [13]. The key idea lies in the observation that often there exists a fast and deterministic heuristic for many combinatorial optimisation problems. Given such a heuristic h, and a problem instance p, it is possible to calculate a solution s very efficiently and, therefore, the pair (h, p) can be read as an encoding for a solution $s = h(p)$. By *perturbing* the heuristic h, the problem p, or both, a subset of solutions can be generated, which forms the neighbourhood for the local search. The *representation* used by the local search is not an encoding of the solution for the original problem p, but an encoding of the *perturbation* d. The problem p can be perturbed by changing the original problem data, whereas the heuristic h can be perturbed by changing its configuration. In the context of the paper, let us focus on the problem perturbation.

The problem space exploration can be powerful when the search landscape in the solution space for the original problem p presents challenges such as a large plateau. Problem space exploration can be used for any class of problems for which there exists a fast and deterministic construction heuristic. Since Storer et al. demonstrated the idea with Job Shop Scheduling Problem, it has been successfully applied to various combinatorial optimisation problems including 0-1 Multiple Knapsack Problem [14], Graph Partitioning Problem [15], Routing Problem [16] and Travelling Salesman Problem [17].

B. Mask-Coding Representation

The representation for problem perturbation is sometimes called 'Weight-Coding' because the perturbation is represented by a vector of weights that is applied to the original problem data [14], [16]. For example, the perturbation vector for a 0-1 knapsack problem would be a collection of weights that will be multiplied to the value of each item in the 0-1 Knapsack Problem.

For set cover problems, a vector of real numbers is not suitable for perturbation as the data consist of sets. Mask-Coding representation introduced in the paper uses bit-masking to perturb either the universe, $\mathcal{U}$, or the family of subsets, $\mathcal{S}$, or both. A genotype representation of a solution encoded with Mask-Coding would still be a binary string, but it does not depict the selection of members in $\mathcal{S}$ as in the traditional binary string representation. Depending on where the masking is applied, there are three different ways to apply Mask-Coding representation to the genotype representation for multi-objective set cover problem: $\mathcal{U}$-mask, $\mathcal{S}$-mask and $\mathcal{US}$-mask.

1) $\mathcal{U}$-Mask Representation: An $\mathcal{U}$-mask perturbs the original problem p by masking a subset of elements in $\mathcal{U}$. An instance of $\mathcal{U}$-mask representation is a binary string, $d = d_1 d_2 \dots d_n$, whose length equals the size of the original universe, $\mathcal{U}$. Without losing generality, let $\mathcal{U}$ be an ordered set with n elements, $\{e_1, \dots, e_n\}$. The perturbed (i.e. masked) universe, $\mathcal{U}^d$ only contains elements e_i such that $d_i = 1$. More formally,

$$\mathcal{U}^d = \mathcal{U} - \{e_i | d_i = 0\}$$

Similarly, the subsets of $\mathcal{U}$ in $\mathcal{S}$ are also masked using d:

$$\mathcal{S}^d = \{S_j^d | \forall S_j \in \mathcal{S}, S_j^d = S_j - \{e_i | d_i = 0\}\}$$

The pair of $(\mathcal{U}^d, \mathcal{S}^d)$ denotes the perturbed problem. Let x be the traditional binary string representation of the solution to the perturbed problem $(\mathcal{U}^d, \mathcal{S}^d)$, which is obtained using a construction heuristic h. It follows that x can also be a solution to the original problem $(\mathcal{U}, \mathcal{S})$ because the length of x remains equal to $|\mathcal{S}|$ and is irrelevant to neither the length nor the cardinality of d. Therefore, it is possible to measure coverage or cost of the set cover expressed with x using the original problem data, $(\mathcal{U}, \mathcal{S})$. Algorithm 1 illustrates the process of measuring coverage and cost of a solution encoded with $\mathcal{U}$-mask representation using the greedy algorithm as the construction heuristic (see Section IV-D for details of the construction heuristic).

Algorithm 1: Fitness evaluation for Multi-Objective Set Cover optimisation using $\mathcal{U}$ Mask-Coding representation and greedy heuristic

Input: the original universe, $\mathcal{U}$, the original family of subsets, $\mathcal{S}$, a solution encoded with $\mathcal{S}$-mask, d
Output: a coverage of d, $coverage_d$, and a cost of d, $cost_d$

FitnessEvaluationForUMask($\mathcal{U}$, $\mathcal{S}$, d)
(1) $\mathcal{U}^d = \mathcal{U} - \{e_i | d_i = 0\}$
(2) $\mathcal{S}^d = \{S_j^d | \forall S_j \in \mathcal{S}, S_j^d = S_j - \{e_i | d_i = 0\}\}$
(3) $x \leftarrow greedy(\mathcal{U}^d, \mathcal{S}^d)$
(4) $coverage_d \leftarrow coverage(x, \mathcal{U}, \mathcal{S})$
(5) $cost_d \leftarrow cost(x, \mathcal{U}, \mathcal{S})$
(6) **return** $coverage_d$, $cost_d$

2) $\mathcal{S}$-Mask Representation: An $\mathcal{S}$-mask perturbs the original problem by masking a subset of family members in $\mathcal{S}$. An instance of $\mathcal{S}$-mask is a binary string, $d = d_1 d_2 \dots d_m$, whose length equals the size of the original $\mathcal{S}$. Without losing generality, let $\mathcal{S}$ be an ordered set with m subsets of $\mathcal{U}$, $\{S_1, \dots, S_m\}$. After perturbation, the original $\mathcal{U}$ remains the same. However, the perturbed $\mathcal{S}$ is defined as follows:

$$\mathcal{S}^d = \mathcal{S} - \{S_j | d_j = 1\}$$

The pair of $(\mathcal{U}, \mathcal{S}^d)$ forms the perturbed problem. Unlike $\mathcal{U}$-mask, the solution x of the perturbed problem cannot be

accepted as a solution to the original problem as the length of x would be equal to the size of $\mathcal{S}^d$ rather than $\mathcal{S}$. Therefore, the solution x to $(\mathcal{U}, \mathcal{S}^d)$ needs to be decoded into a solution $x' = x'_1 \dots x'_m$ for the original problem $(\mathcal{U}, \mathcal{S})$. Each digit of x' is defined as follows w.r.t. the mask d:

$$x'_i = \begin{cases} 1 \text{ if } d_j = 1 \text{ and } S_j \text{ is selected by } x \\ 0 \text{ otherwise} \end{cases}$$

Algorithm 2 shows the process of measuring coverage and cost of a solution encoded with $\mathcal{S}$-mask representation using greedy algorithm as the construction heuristic.

Algorithm 2: Fitness evaluation for Multi-Objective Set Cover optimisation using $\mathcal{S}$ Mask-Coding representation and greedy heuristic

Input: the original universe, $\mathcal{U}$, the original family of subsets, $\mathcal{S}$, a solution encoded with $\mathcal{S}$-mask, d
Output: a coverage of d, $coverage_d$, and a cost of d, $cost_d$
FITNESSEVALUATIONFORSMASK($\mathcal{U}$, $\mathcal{S}$, d)
(1) $\mathcal{S}^d = \mathcal{S} - \{S_j | d_j = 1\}$
(2) $x \leftarrow greedy(\mathcal{U}, \mathcal{S}^d)$
(3) $x' \leftarrow decode(x, d)$
(4) $coverage_d \leftarrow coverage(x', \mathcal{U}, \mathcal{S})$
(5) $cost_d \leftarrow cost(x', \mathcal{U}, \mathcal{S})$
(6) **return** $coverage_d$, $cost_d$

3) $\mathcal{US}$-Mask Representation: It is also possible to perturb both $\mathcal{U}$ and $\mathcal{S}$ simultaneously. An instance of $\mathcal{US}$-mask is a binary string of length $(n+m)$, $d = d_1 d_2 \dots d_{n+m}$. The first n digits of d form the $\mathcal{U}$ mask, du, whereas the following m digits form the $\mathcal{S}$ mask, ds. The perturbation of $\mathcal{U}$ remains the same as in the case of $\mathcal{U}$ masking:

$$\mathcal{U}^{du} = \mathcal{U} - \{e_i | du_i = 0\}$$

However, the perturbation of $\mathcal{S}$ requires a different approach. First, the members of $\mathcal{S}$ that are masked by ds should be removed from $\mathcal{S}^{ds}$. Second, the masked elements of $\mathcal{U}$ should be also masked in each member of $\mathcal{S}^{ds}$. More formally,

$$\mathcal{S}^{ds} = \{S_j^{ds} | \forall S_j \in \mathcal{S} - \{S_j | ds_j = 1\}, \\ S_j^{ds} = S_j - \{e_i | du_i = 0\}\}$$

Since $\mathcal{S}$ has also been perturbed, the solution x from greedy algorithm needs to be decoded following the description in Section III-B2.

IV. EXPERIMENTAL SET-UP

A. Research Questions

The aim of the empirical study is to evaluate the impact of Mask-Coding representation on the optimisation of set cover problems based on code coverage data. The empirical study compares 4 different representations for Multi-Objective Set Cover: the traditional binary string representation, the $\mathcal{U}$-mask representation, the $\mathcal{S}$-mask representation and the $\mathcal{US}$-mask representation. In comparing these representations, the paper asks the following research questions:

- **RQ1. Convergence:** how well do the solutions from each representation converge to the optimal Pareto-frontier?
- **RQ2. Diversity:** how diverse are the solutions from each representation?
- **RQ3. Efficiency:** what is the impact of using Mask-Coding representation on the running time of the algorithm?

RQ1 and **RQ2** are answered by analysing the Pareto-fronts produced by different representations. Ideal measurement of convergence and diversity would require the knowledge of the true Pareto-fronts. Since it is not available, reference Pareto-fronts are formed by combining all the available results. **RQ3** concerns the additional computation resource required when using Mask-Coding representation, i.e. that of greedy algorithm. It is answered by measuring the execution time of each representation.

B. Subjects

Table I shows the subject test suites used in the empirical study. The test suites are obtained from Software Infrastructure Repository [18]. The set cover problem is instantiated with statement coverage data. That is, the universe $\mathcal{U}$ corresponds to the set of all statements in programs. In turn, the family of subsets $\mathcal{S}$ corresponds to the set of all execution traces of all the test cases in test suites. Two different types of test suites were deliberately chosen: ones with a small $\mathcal{U}$ and a large $\mathcal{S}$ (`printtokens` and `tcas`) and ones with a large $\mathcal{U}$ and a small $\mathcal{S}$ (`flex` and `gzip`). The level of redundancy in $\mathcal{S}$ (i.e. test suites) is much higher in the test suites that belong to the first class than the second class. Note that, while the test suites in the first class have already been studied for multi-objective test suite minimisation [5], only smaller subsets of the entire test suite have been considered. This paper deliberately uses the entire pool of test cases in order to force the high level of redundancy.

Table I
SIZES OF FOUR SUBJECT TEST SUITES OBTAINED FROM SIR

Subject	No. of statements	Test Suite Size
`printtokens`	189	4,115
`tcas`	65	1,608
`flex`	3,965	103
`gzip`	2,007	213

The coverage for each test has been measured using `gcov`, a widely used code profiling tool from the `gcc` compiler suite. The cost of executing each test has been measured using `valgrind` profiling tool [19]; for the

execution of each test, the number of CPU instructions has been measured and used as the execution cost.

While the empirical study is based on code coverage data, its aim is to analyse the impact of Mask-Coding representation rather than to show their benefits in the context of software testing. Therefore, the impact of Multi-Objective Test Suite Minimisation on fault detection capability lies beyond the interest of this paper and will not be considered.

C. MOEA Algorithm

The four representations are evaluated using a Multi-Objective Evolutionary Algorithm (MOEA) called Two-Archive Algorithm [20]. Two Archive Algorithm maintains two separate memorisation archive for convergence and diversity respectively. Algorithm 3 shows the high-level outline of Two-Archive Algorithm.

The key idea behind Two-Archive Algorithm lies in the algorithm that collects non-dominated solutions to two separate archives. A non-dominated solution from the population is first compared to both archives. If the new solution is dominated by a solution from archives, it is discarded. If the new solution is not dominated, there are two possibilities: 1) the new solution dominates a solution in archives, in which case the dominated solution is removed from the archive and the new solution is added to the convergence archive, and 2) the new solution is not dominated by and does not dominate any solution in archives, in which case the new solution is added to the diversity archive. In order to control the size of the archive, solutions are removed from the diversity archive when the size goes over a predefined limit: the solution in diversity archive that has the shortest distance to any solution in the convergence archive is removed. For more details, readers are encouraged to refer to Praditwong and Yao [20].

Algorithm 3: Outline of Two-Archive Algorithm

(1) Initialise the population
(2) Initialise archives to the empty set
(3) Evaluate initial population
(4) **while** stopping criterion is not met
(5) Collect non-dominated individuals to archives
(6) Select parents from archives
(7) Generate a new population from parents
(8) Evaluate the new population

When using Mask-Coding, the individual solutions in the population represent the masking, i.e. the input d of Algorithm 1-**??**, rather than the actual solution. The selection operator for Two-Archive Algorithm selects two parents from both archives with uniform probability distribution. It also uses the standard single-point crossover operator with the crossover rate of 0.9 and the single bit-flip mutation. The population size was set to 100. The stopping criterion was set to the maximum of 25,000 fitness evaluations.

D. Construction Heuristic

Fitness evaluation using problem space exploration requires an efficient and effective construction heuristic. For set cover, the greedy algorithm is known to produce results that are within $\ln n$ of the optimal cost [21]. Algorithm 4 describes the additional greedy algorithm used as the construction heuristic in the empirical study.

Algorithm 4: Outline of additional greedy algorithm ADDITIONALGREEDY($\mathcal{U}$, $\mathcal{S}$)

(1) $C \leftarrow \phi$ // covered elements in $\mathcal{U}$
(2) **repeat**
(3) $k \leftarrow min_k(cost_k/|S_k - C|)$
(4) add S_k to solution
(5) $C = C \bigcup S_k$
(6) **until** $C = \mathcal{U}$

E. Evaluation

In order to cater for the inherent randomness in population-based evolutionary algorithm, each experiment was repeated 30 times. The reference Pareto-fronts for convergence and diversity research questions were formed by combining solutions from all four representations and identifying a Pareto-front from the combined set of solutions, i.e. the results from 120 individual runs (4 representations, 30 runs per representation).

RQ1 and **RQ2** are answered by statistically analysing the number of solutions contributed to the reference Pareto-front by each representation. The hypothesis test is performed using t-test; while the distribution of the sample is not known, the central limit theorem dictates that the distribution approximates the normal distribution with a large enough sample size [22]. Additionally, Wilcoxon's rank-sum test, the non-parametric alternative, would not produce the precise p-value under the existence of ties, which have been frequently observed in the results.

V. RESULTS AND ANALYSIS

A. Convergence

Figure 1 shows the boxplots of the number of unique solutions contributed to the reference Pareto-fronts by each representation. The plot $u0s0$ represents neither $\mathcal{U}$ nor $\mathcal{S}$ mask, i.e. the traditional binary string representation. Respectively, $u1$ and $s1$ represent $\mathcal{U}$- and $\mathcal{S}$-mask being used.

One surprising finding is that the traditional binary string representation failed to produce any solution on the reference Pareto-front in the cases of `printtokens` and `tcas`. Additionally, even though the redundancy is not so severe in the test suite of `gzip`, the traditional binary string representation contributes very little. This shows that the traditional binary string representation may not be effective if the search landscape contains a large plateau (which, in the case of `printtokens` and `tcas`, is incurred by the high level of redundancy in the test suites).

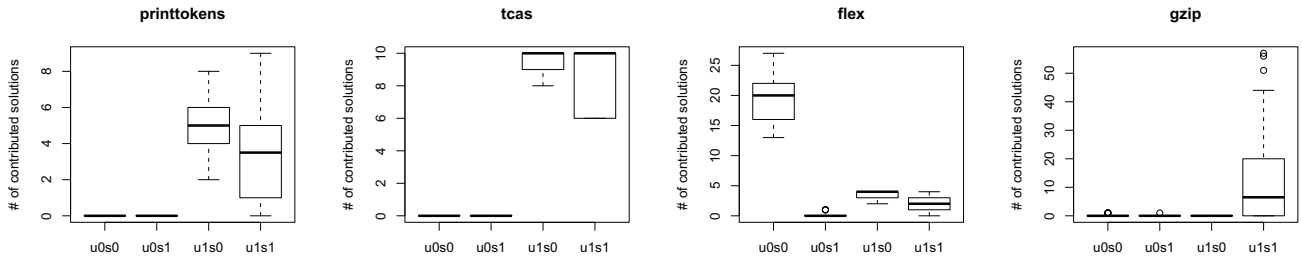


Figure 1. Number of unique solutions that are contributed to the reference Parefo-front by different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. While different combinations of Mask-Coding work best for different problems, $\mathcal{US}$- and $\mathcal{U}$-mask tend to produce a larger number of non-dominated solutions for 3 out of 4 datasets.

Table II
MEAN AND STANDARD DEVIATION OF THE NUMBER OF UNIQUE SOLUTIONS CONTRIBUTED TO THE REFERENCE PARETO-FRONT

Subject	Traditional		$\mathcal{U}$-Mask		$\mathcal{S}$-Mask		$\mathcal{US}$-Mask	
	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ
printtokens	0.0	0.0	4.83	1.55	0.0	0.0	3.57	2.63
tcas	0.0	0.0	9.60	0.55	0.0	0.0	8.80	1.83
flex	19.47	1.26	3.56	0.56	0.13	0.34	2.23	1.26
gzip	0.20	0.40	0.0	0.0	0.03	0.18	14.83	18.49

Table III
THE p-VALUES OF THE STATISTICAL HYPOTHESIS TEST BETWEEN $\bar{n}_{\mathcal{U}}$ AND $\bar{n}_{\mathcal{US}}$. THE HYPOTHESIS IS TESTED WITH ONE-TAILED t-TEST WITH THE SIGNIFICANCE LEVEL OF 95%.

Subject	p-value ($\bar{n}_{\mathcal{U}} > \bar{n}_{\mathcal{US}}$)
printtokens	0.015
tcas	0.015
flex	< 0.001
gzip	1.0

In printtokens, tcas and gzip, $\mathcal{U}$-mask tends to contribute the most to the number of solutions contributed to the reference Pareto-front. Interestingly, $\mathcal{U}$-mask seems to be more effective than $\mathcal{US}$-mask for printtokens and tcas. In all programs, $\mathcal{S}$-mask alone did not perform very well. Table II shows the statistical details of the results presented in Figure 1.

Table III shows the results of statistical hypothesis test of the data presented in Figure 1. The comparison between the traditional binary string representation and Mask-Coding representations does not require any statistical analysis: the results from the traditional representation is either almost always 0 (printtokens, tcas and gzip) or completely surpasses other representations (flex). Rather, the statistical analysis was performed to see whether $\mathcal{U}$-mask is more effective than $\mathcal{US}$-mask with statistical significance. The *null* hypothesis is that $\bar{n}_{\mathcal{U}}$ and $\bar{n}_{\mathcal{US}}$ are the same. The *alternative* hypothesis is that $\bar{n}_{\mathcal{U}}$ is greater than $\bar{n}_{\mathcal{US}}$. The hypothesis is tested with one-tailed t-test with 95% significance level.

In printtokens, tcas and flex, the null hypothesis is rejected with statistical significance, meaning that $\mathcal{U}$-mask produces more unique solutions on the reference Pareto-front than $\mathcal{US}$-mask. From Table II and Table III, **RQ1** is answered as follows: Mask-Coding representation results in higher convergence compared to the traditional binary string representation if the search landscape contains a large dimensional plateau. This claim is backed by the observation of larger number of unique solutions contributed to the reference Pareto-fronts for subjects printtokens and tcas.

B. Diversity

In order to answer **RQ2**, the number of unique solutions produced by each representation is compared statistically. Unlike Section V-A, all solutions produced by each representation are considered, regardless of whether they are on the reference Pareto-fronts or not.

Table IV
MEAN AND STANDARD DEVIATION OF THE NUMBER OF UNIQUE SOLUTIONS PRODUCED BY EACH REPRESENTATION.

Subject	Traditional		$\mathcal{U}$-Mask		$\mathcal{S}$-Mask		$\mathcal{US}$-Mask	
	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ	$\bar{n}$	σ
prttkn	1.03	0.18	10.7	1.83	1.0	0.0	11.03	3.40
tcas	1.00	0.0	9.60	0.55	1.0	0.0	8.80	1.83
flex	26.73	2.34	8.66	1.81	26.77	6.13	41.07	8.57
gzip	1.10	0.30	16.77	3.87	47.30	8.47	126.77	21.07

Figure 2 shows the boxplots of the number of unique solutions produced by each representation. When compared to Figure 1, it can be observed that $\mathcal{US}$-mask had produced some solutions for printtokens that were dominated by the solutions produced by $\mathcal{U}$-mask. Another interesting observation is that $\mathcal{US}$-mask has produced a much larger number of solutions for flex compared to the traditional binary string representation, but most of those additional

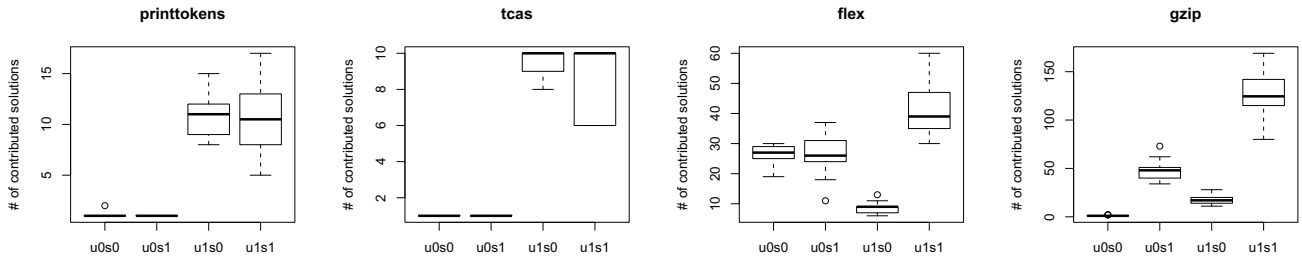


Figure 2. Number of unique solutions that are produced by different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. Overall, $\mathcal{US}$-mask tends to produce a larger number solutions compared to other representations.

Table V
THE p-VALUES OF THE STATISTICAL HYPOTHESIS TEST BETWEEN $\bar{n}_{\mathcal{US}}$, $\bar{n}_{\mathcal{U}}$, $\bar{n}_{\mathcal{S}}$ AND $\bar{n}_0$ (TRADITIONAL BINARY STRING REPRESENTATION) FOR **RQ2**. THE HYPOTHESIS IS TESTED WITH ONE-TAILED t-TEST WITH THE SIGNIFICANCE LEVEL OF 95%.

Subject	p-value		
	$\bar{n}_{\mathcal{US}} > \bar{n}_0$	$\bar{n}_{\mathcal{U}} > \bar{n}_0$	$\bar{n}_{\mathcal{S}} > \bar{n}_0$
`flex`	$< 10^{-9}$	1.0	0.48
`gzip`	$< 10^{-19}$	$< 10^{-19}$	$< 10^{-19}$

solutions were at the same time dominated by the solutions produced by the binary string representation.

Since the boxplots for `printtokens` and `tcas` largely reproduce the results in Figure 1 for which the traditional binary string representation and $\mathcal{S}$-mask do not produce almost any solution at all, the statistical analysis of diversity results focuses on the cases of `flex` and `gzip`. For each representation, the *null* hypothesis is that there is no difference in the number of unique solutions produced by the traditional binary string representation and the corresponding Mask-Coding representation. The alternative hypothesis is that the corresponding Mask-Coding representation produces a larger number of unique solutions. The results from the traditional binary string representation are denoted with $\bar{n}_0$.

Table V shows the result of the statistical hypothesis test. For `flex`, the alternative hypothesis is only accepted for $\mathcal{US}$-mask at the significance level of 95%. For `gzip`, all three Mask-Coding representations produce a larger number of unique solutions compared to the traditional binary string representation.

Figure 3 shows the shape of Pareto-fronts produced by different representations in order to facilitate more qualitative analysis of the results. The plot for each representation consists of non-dominated solutions collected from the combined results of the 30 repeated runs. For `printtokens` and `tcas`, the Pareto-fronts from both $\mathcal{US}$- and $\mathcal{U}$-mask covers the widest range of solutions. In contrast, the traditional binary string representation fails to escape the dimensional plateau and produces only one solution.

With `flex` and `gzip`, it can be observed that all three types of Mask-Coding largely fail to produce solutions with low cost and low coverage. This may be explained by the differences in redundancy in test suites. Solutions with low cost and low coverage will in turn require the inclusion of test cases with extremely low cost and coverage. These test cases are more likely to represent less frequent usage pattern of the program, e.g. error handling routines, and, therefore, the proportion of such test cases in the entire test suite is likely to be small. If the test suite has a very high level of redundancy that can lead to a dimensional plateau, the probability for the masking to hide these low cost/low coverage test cases is relatively low. On the other hand, if the level of redundancy is low, the probability for the masking to hide these test cases increase. This in turn may prevent the optimisation algorithm to produce solutions with low cost and low coverage.

Overall, **RQ2** is answered as follows: if the set cover problem contains a high level of redundancy in $\mathcal{S}$ that can lead to a dimensional plateau, Mask-Coding representation can help escaping the dimensional plateau to produce a Pareto-front with high diversity.

C. Impact on Performance

Since the fitness evaluation for Mask-Coding representation involves using a separate construction heuristic, it requires additional computation resources. Figure 4 shows the boxplots of the wall-clock execution time measured for the runs of different representations. While all three Mask-Coding representations require more computational resources than the traditional binary string representation, the amount of resources additionally required differs depending on the type of masking.

For `printtokens` and `tcas`, $\mathcal{U}$- and $\mathcal{S}$-mask requires similarly large amounts of additional computational cost whereas $\mathcal{US}$-mask requires significantly less. The combined use of both types of masking has reduced the size of problem instances for the construction heuristic significantly enough

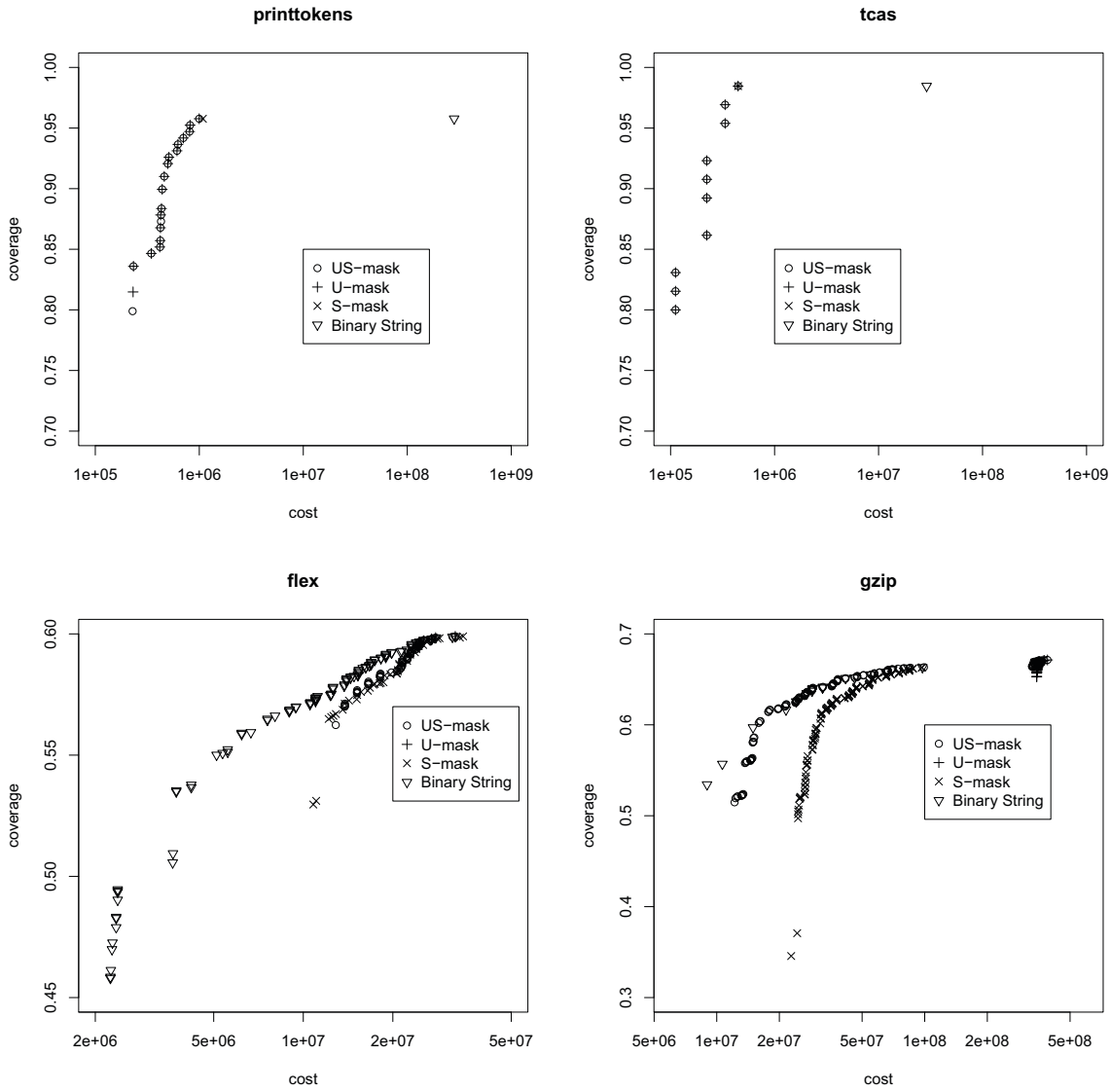


Figure 3. The shape of Pareto-fronts produced by different representations. Each Pareto-front consists of non-dominated solutions collected from the combined results of the 30 repeated runs. The x-axis is in a logarithmic scale.

to have an impact on the overall execution time. However, for `flex` and `gzip`, only $\mathcal{S}$-mask seems to make any difference in the amount of additionally required computational resources. This is probably because $|\mathcal{U}|$ is much larger than $|\mathcal{S}|$. Masking one element in $\mathcal{U}$ saves $|\mathcal{S}|$ steps for the construction heuristic, and vice versa. Therefore, if $|\mathcal{U}| \gg |\mathcal{S}|$, the impact of $\mathcal{S}$-mask is much bigger than that of $\mathcal{U}$-mask.

For all subjects, all three Mask-Coding representation require a significantly large amount of additional computation power. This partially answers **RQ4**. However, it should be noted that the data presented in Figure 4 are the measurements of execution time for fixed number of fitness evaluations. That is, the algorithms may have continued to run even after they have converged. Therefore, these data should not be read as the true cost of the use of Mask-Coding, which can be only measured by the time it took to converge to the Pareto-front. However, since the use of convergence as a stopping criterion requires the knowledge of the true Pareto-front *a priori*, here only the additional overhead of using Mask-Coding is studied and presented.

D. Threats to Validity

There are a few threats to validity regarding the generalisation of the results presented in this paper. First, most

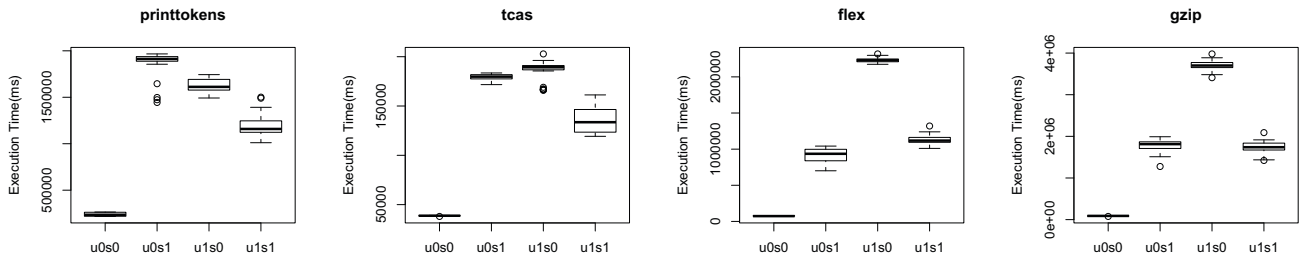


Figure 4. Execution time required by the use of different representations. Boxplot $u0s0$ represents both $\mathcal{U}$ and $\mathcal{S}$ masking turned off, i.e. the traditional binary string representation, whereas $u1$ and $s1$ represent $\mathcal{U}$ and $\mathcal{S}$ masking turned on respectively. For `printtokens` and `tcas`, $\mathcal{US}$-mask requires the least amount of time as the use of both types of masking reduces the computational cost for the construction heuristic by reducing the problem size. For `flex` and `gzip`, since $|\mathcal{U}|$ is much larger than $|\mathcal{S}|$, $\mathcal{U}$-masking requires less computational cost for the construction heuristic than $\mathcal{S}$-masking.

of existing work on problem space exploration has been done with single objective optimisation. The implications of applying the same idea to multi-objective optimisation problems are not clear and Mask-Coding representation may perform differently when applied to single-objective problems. This can only be answered with further empirical evaluation of the new representation. However, this paper chooses to evaluate the new representation with respect to the multi-objective optimisation because, in the context of Search-Based Software Engineering, the multi-objective version of set cover problem provides much more value to practitioners compared to the single-objective version of the same problem [5]. Second, there is no evidence that the additional greedy algorithm is the ideal choice of construction heuristic for the approach presented here. However, the additional greedy algorithm was selected due to its known effectiveness for set cover problem and it fits the profile of an ideal construction heuristic.

Threats to construct validity arises when the measurement used in the study does not reflect the concepts they represent. It should be noted that the research question on performance only evaluates the additional computational resource required by the masking. It does not reflect the savings in fitness evaluation that could have been gained if the stopping criterion was set differently. For example, if the true reference Pareto-front had been known, the stopping criterion could have been set with respect to the distance to the reference Pareto-front. If the new representation converges faster than the traditional representation, it would require less fitness evaluations. However, without the knowledge of the true Pareto-fronts, it was not possible to set the stopping criterion with respect to convergence.

VI. Related Work

Problem space exploration was first suggested by Storer et al. in an attempt to improve the optimisation for Job-Shop Scheduling problem [10]. A similar approach was also introduced as *noising* by Charon and Hudry [13]. Storer et al. discussed two different approaches of exploring the problem space: by perturbing the problem and by perturbing the construction heuristic (e.g. changing parameters of the construction heuristic). Since the additional greedy algorithm does not require any parameter tuning, the heuristic perturbation has not been considered in this paper.

The idea was applied to various combinatorial optimisation problems including 0-1 Multiple Knapsack Problem [14], Graph Partitioning Problem [15], Routing Problem [16] and Travelling Salesman Problem [17]. For all of these problems, the problem perturbation is represented as a vector of real numbers, which are usually weights that are multiplied to the numbers in the original problem. Therefore, these representations are often called *weight-coding*. However, no existing work uses bit-masking to perturb problem data expressed as sets.

Set-cover problem formed the basis of the widely studied test suite minimisation problem [1], [2], [23], [24]. Recently, formulating the test suite minimisation problem as a multi-objective set cover optimisation is an emerging trend found in search-based software testing [4]–[6]. This is because shorter development cycle often require the precise knowledge of how much testing is feasible given a budget on time. All of the existing work rely on the traditional binary string representation and, therefore, potentially suffer from the existence of dimensional plateau.

VII. Conclusions and Future Work

The paper introduces Mask-Coding, a novel representation for solutions of multi-objective set cover problem based on the concept of problem space exploration and problem perturbation. Mask-Coding uses bit-masks to perturb instances of set-cover problems. The empirical evaluation of the novel representation has shown that it can outperform the traditional binary string representation, especially under the existence of the dimensional plateau. Future work will consider evaluation of the representation with wider problem instances.

ACKNOWLEDGEMENT

The author is grateful to Xin Yao for an insightful discussion on the significance of the right choice of genotype representation.

REFERENCES

[1] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 3, pp. 270–285, 1993.

[2] J. Offutt, J. Pan, and J. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*. ACM Press, June 1995, pp. 111–123.

[3] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Information Processing Letters*, vol. 60, no. 3, pp. 135–141, 1996.

[4] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time aware test suite prioritization," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2006)*. ACM Press, July 2006, pp. 1–12.

[5] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2007)*. ACM Press, July 2007, pp. 140–150.

[6] C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T. de Souza, "A multi-objective approach for the regression test case selection problem," in *Proceedings of Anais do XLI Simpòsio Brasileiro de Pesquisa Operacional (SBPO 2009)*, 2009, pp. 1824–1835.

[7] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, Dec. 2001.

[8] J. Gottlieb, B. A. Julstrom, F. Rothlauf, and G. R. Raidl, "Prüfer numbers: A poor representation of spanning trees for evolutionary search." in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001, pp. 343–350.

[9] F. Rothlauf, D. E. Goldberg, and A. Heinzl, "Network random keys—a tree representation scheme for genetic and evolutionary algorithms," *Evolutionary Computation*, vol. 10, no. 1, pp. 75–97, 2002.

[10] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Journal of Management Science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[11] R. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, 1972.

[12] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1983.

[13] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133–137, 1993.

[14] C. Cotta and J. Troya, "A hybrid genetic algorithm for the 0-1 multiple knapsack problem," in *Artificial Neural Networks and Genetic Algorithms*. Springer-Verlag, 1997, pp. 251–255.

[15] V. Sudhakar and C. Siva Ram Murthy, "A modified noising algorithm for the graph partitioning problem," *Integration, the VLSI Journal*, vol. 22, no. 1-2, pp. 101–113, 1997.

[16] M. J. Morgan and C. L. Mumford, "A weight-coded genetic algorithm for the capacitated arc routing problem," in *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009)*. New York, NY, USA: ACM, 2009, pp. 325–332.

[17] B. Codenotti, G. Manzini, L. Margara, and G. Resta, "Perturbation: An efficient technique for the solution of very large instances of the euclidean tsp," *INFORMS JOURNAL ON COMPUTING*, vol. 8, no. 2, pp. 125–133, 1996.

[18] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact." *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[19] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," in *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2007)*. ACM Press, June 2007, pp. 89–100.

[20] K. Praditwong and X. Yao, "A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm," in *Proceedings of Computational Intelligence and Security, International Conference*, ser. Lecture Notes in Computer Science, vol. 4456, November 2006, pp. 95–104.

[21] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Proceedings of the 5th annual ACM Symposium on Theory of Computing (STOC 1973)*. ACM Press, May 1973, pp. 38–49.

[22] J. Rice, *Mathematical Statistics and Data Analysis*. Duxbury Press, 2nd Ed. 1995.

[23] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*. ACM Press, May 2004, pp. 106–115.

[24] G. Rothermel, M. Harrold, J. Ronne, and C. Hong, "Empirical studies of test suite reduction," *Software Testing, Verification, and Reliability*, vol. 4, no. 2, pp. 219–249, December 2002.

Test Case Selection Method for Emergency Changes

Fábio de A. Farzat
Postgraduate Information Systems Program – UNIRIO
Av. Pasteur 458, Urca – Rio de Janeiro, RJ – Brazil
fabio.farzat@uniriotec.br

***Abstract* —Software testing is an expensive task that significantly contributes to the total cost of a software development project. Among the many strategies available to test a software project, the creation of automated test cases that can be enacted after building a release or resolving a defect is increasingly used in the industry. However, certain defects found in the system operation may block major business operations. These critical defects are sometimes resolved directly in the production environment under such a restricted deadline that there is not enough time to run the complete set of automated test cases upon the patched version of the software. Declining to run the test case suite allows a quicker release of the software to production, but also allows other defects to be introduced into the system. This paper presents a heuristic approach to select test cases that might support emergency changes aiming to maximize the coverage and diversity of the testing activity under a strict time constraint and given the priority of the features that were changed.**

Keywords-*Software Testing; Genetic Algorithm; Time Constraint; Process development; Risk Management*

I. INTRODUCTION

One of the most costly activities of the development process, sometimes covering up to 50% of the total project cost [10], testing aims to reveal defects introduced into the software by former activities of the development process. Although most test teams see the activity as a way to prove that a particular piece of software works correctly, testing activities cannot prove that a software system is correct: it can only prove the opposite, by appointing errors in the software.

Due to their high cost, testing activities are often given less effort than it would be required to completely validate the software. In extreme cases, they are almost eliminated from the development process for one or more releases of the system. Even in companies that have an institutionalized process, testing is sometimes left to a secondary role for some releases due to business pressure. This behavior is even more common when errors are found in production, thus blocking the perfect execution of business transactions.

In such situations, corrective maintenance activities are executed directly in the production environment, releasing the patched version of the software without proper test. Given the need to resolve the failure, the test team performs the change and immediately deploys it. Moreover, the risk of inserting more defects into the system and causing collateral damage to the operation is even worse than not testing an emergency change.

To minimize the risk of an incorrect patch or one that introduces more defects, some sort of guarantee must be defined. For instance, one may test all the code referring to features affected by the patch made as part of the emergency release. However, to manually separate test cases related to a specific set of changes can be time consuming. Since this is an emergency, time is a decisive criterion. Another disadvantage is that, even if it takes time, covering only the code changed cannot assure that other essential parts are still working was pretended.

Thus, we need to select a set of test cases that can run in a given time frame and cover the most important features that were affected by the emergency change, according to the priority of each feature. Selecting such an appropriate subset of test cases is an optimization problem and, in systems with a large test suite, there may be no exact method capable to find the optimal solution in reasonable time. This paper presents a formal model for this problem, a data collection procedure to acquire the information to properly address it, and a heuristic method to propose good-enough solutions for it.

II. MODELING THE EMERGENCY TEST CASE SELECTION PROBLEM

Given an emergency release, we address the emergency test case problem as the selection of a set of test cases that can be executed within a short time limit and that maximize coverage of the changes that were made within the release. Changes have different priorities, according to the features that they affect. We restrict our problem to unit tests, which are fine-grained tests developed to validate a particular code module. The source code modules, on the other hand, take part in the implementation of one or more features. Figure 1 presents the relationships among these elements.

This problem is related to the NP-Complete problem that determines the items to include in a collection, each item being described by a weight and a value, so that the total weight of the collection is less than a given limit and its total value is as large as possible. In this work, an item is mapped to a test case, an item's weight is related to the test case's execution time, the weight limit is the time constraint, and the total value depends on an alignment between the features affected by recent changes and the test case coverage upon these features.

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.13

A **feature** (F) represents a set of functional requirements provided by the software. Usually, each specific transaction or operation performed by the software is considered a feature. In our context, a feature will always represent a set of functional requirements that are implemented by a set of source code modules (analysis and design artifacts are not handled in our test case selection method). Each feature has a property indicating its **priority** to the operation of the software system. The more important the feature to support the business, the highest will be its priority value. Features with maximum priority will always be marked as changed, thus making certain that they will be tested.

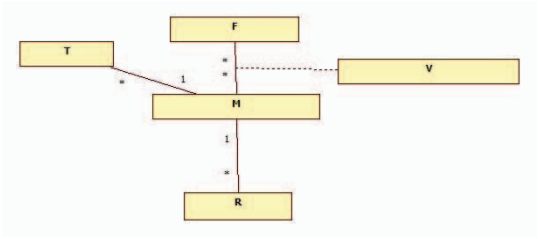


Figure 1. Emergency test case selection model.

A **source code module** (M) is a file that contains part of source code comprising the system. Each source code module participates in the implementation of one or more features. Thus, there is a relationship (V) between features and source code modules, so that each module implements a certain percentage of the feature.

Each module has a set of **revisions** (R) and **test cases (**T) associated to it. Each revision associated with a given source code module mark represents a particular change suffered by the module. It is strictly related to the version control system used to support software development. Each test associated with a given code module covers certain pieces of code. Since features are implemented by source code modules, the test cases indirectly cover certain features of software.

Software **tests** can be defined as a treatment composed by a set of entry values, an action, and an expected result. The action is executed using the entry values to derive an output, which is compared to the expected result. Tests can be classified as positive or negative. Nyman [6] conceptualizes these tests as follows:

- **Positive Tests**: tests that seek to demonstrate that a particular source code module does what it should do, according to its design specification. For instance, a positive test can check whether an application yields a certain result when particular parameter values are submitted to a calculation;
- **Negative Tests**: tests that seek to demonstrate that a particular source code module does nothing that it should not do, according to restrictions specified in its design documents. These tests are developed with the intention to "break" the system. The unexpected behavior obtained from a negative test is when the system does not display an error message when it should or displays an error message when it should not.

In our proposed approach to select emergency test cases we allow the manager to assign a weight to positive (W_p) and negative tests (W_n) in a way that these weight sum up to 100%. The weights allow the manager to decide whether more positive or negative tests should be selected. Such a decision may depend on the types of changes suffered by the system during the last emergency update. While selecting the test cases, our proposed method takes into account these weights.

III. A DATA COLLECTION PROCEDURE TO SUPPORT EMERGENCY TEST CASE SELECTION

To provide the information required by the test case selection method, we propose using data about the latest changes suffered by the source code modules comprising the system. Such information determines the set of modules that need to be tested. If available tests cases related to these modules require more time to be executed than the desired time restriction, an optimization procedure is enacted to select a subset of these tests cases according to their coverage and the priority of the features implemented by the source code modules that they validate.

Version control is the task of keeping software systems consisting of many versions and configurations well organized [5]. SubVersion (or SVN) is one of many systems that are used in software projects for version management, being much diffused in the open source community because of its distribution under an open source license. Projects such as Apache and Debian use SubVersion to support their configuration management process. In order to collect information about the source code modules that were changed as part of the last release, we collect information from the change history log maintained by SubVersion. By reading the change history log, we can identify the source code modules that comprise the application and their structure in the file system (the directories under which they reside). For each module, the change history log records the dates when changes were made, the developer who modified the code, and a comment describing the changes.

While the source code modules comprising the system can be identified by investigating the change history log, the features provided by the system cannot be directly retrieved from a tool commonly used in the industry or open source projects. Some requirement engineering tools allow the user to maintain a set of features. Tools supporting product lines and domain oriented engineering also register this kind of information. So, if such tools are available, the features can be retrieved from them. In a general scenario, we intend to develop a tool that reads the change history log, allows the user to enter the features offered by the software, and to associate these features to source code modules. This tool will feed information to the optimization process, describing

which source code modules implement each feature and the percentage of the feature related to each module.

Since we restrict our approach to unit tests, each test case is directly linked to one and only one source code module. Test management tools allow developers to maintain a repository of test cases, sometimes relating them to the source code modules that they validate. Even if such tools are not available, unit test cases usually follow strict development rules, that include naming conventions and restrictions on the directories under which they need to be saved. Thus, the set of test cases that were developed for a system can easily be retrieved.

Finally, we need to collect data about the coverage of each test case. This data is automatically acquired from unit testing execution tools which are also commonly used in the industry. Tools like JUnit[1] (for Java) or Nunit [2](for .Net) can run a set of unit tests and collect data on their execution time. Other tools, like JCoverage[3] or NCover[4], can determine the coverage of a set of test cases. Such information is available in log files that can be fed into the optimization process.

IV. THE OPTIMIZATION PROCESS

After collecting the information from the version control system and complementing this information with data about the relationships between source code modules and features, and between source code modules and test cases, our proposed method uses a genetic algorithm to determine the best combination of test cases covering the changed features and that can be executed under a restricted time span.

The usage of heuristic optimization procedures in the software testing field is not innovative, even in the test case selection area. However, the emergency release problem imposes several particularities that were not addressed in previously published works.

Krishnamoorthi and Sahaaya [1] propose a genetic algorithm that selects a set of test cases which can run within a predefined time constraint and maximize the coverage of the whole source code comprising the system. However, their optimization process does not take into account relevant information for emergency patches, like functionalities that were affected by the changes, their priority, or the types of tests that compose the available test suite. Therefore, the test cases selected by their proposed method may cover important parts of the system, but parts that may not have been recently changed.

Walcott [11] proposes a heuristic method that intends to organize the execution order of the test suite to quickly detect faults during the testing process. By increasing the rate of fault detection earlier in the test suite, errors can be found earlier during the testing phase.

V. FORMAL MODEL

The emergency test case selection problem maximizes the coverage and diversity (positive and negative tests) of the testing activity, respecting a strict constraint upon the time required to execute a set of test cases and the priority of the features changed in the last release. Formally, we have:

- Let F be a set of features. Each element $F_i \in F$ is described by a name and a priority;

  $F = \{F_i\}$

  $F_i = [name_i, priority_i]$

- Let Q = {P1, P2, P3, P4, P5} be the set of feature priorities and allow each element $q_i \in Q$ to be associated with a unique positive weight w_i where $w_1 > w_2 > w_3 > w_4 > w_5$;

- Let M be a set of source code modules. Each element $M_i \in M$ is described by a name and a set of revisions;

  $M = \{M_i\}$

  $M_i = [name_i, R_i, T_i]$

- Let R_i be the set of revisions associated to a given source code module M_i. Each $R_{ij} \in R_i$ is described by a number, a date, a description, and the author responsible for the change that created the revision;

  $R_i = \{R_{ij}\}$

  $R_{ij} = [number_{ij}, date_{ij,} description_{ij,} author_{ij}]$

- Let T_i be the set of test cases associated to a given source code module M_i. Each $T_{ij} \in T_i$ is described by a name, a type (positive or negative), an execution time, and a coverage percentage;

  $T_i = \{T_{ij}\}$

  $T_{ij} = [name_{ij}, type_{ij}, time_{ij}, coverage_{ij}]$

- Let V be a set of relationships between source code modules and features. Each element $V_i \in V$ is described by a module, a feature, and an ownership percentage;

  $V = \{V_i\}$

  $V_i = [M_i, F_i, perc_i]$

  $\therefore M_i \in M,$

  $F_i \in F,$

  $0 < V_i.percent \leq 100\%,$

  $$\forall F_k \in F \rightarrow \sum_{v \in V | v.F = F_k} v.perc = 100\%$$

[1] http://www.junit.org
[2] http://www.nunit.org/
[3] http://www.jcoverage.com/
[4] http://ncover.org

- Let S be an emergency test suite, that is, the set of test cases selected to support the release of a software patch under strict time restrictions and covering the features affected by the most recent changes;

$$S = \{T_1, T_2, T_3 \dots T_n\}$$

The optimization process aims to select a good-enough emergency test suite S* so that it can be executed in a given time span (L) and maximize a reward function.

$$S^* \subseteq S \mid time(S^*) \le L \wedge \nexists\, s \subseteq S \therefore s \neq S^* \wedge time(s) \le L \wedge reward(s) \ge reward(S^*)$$

$$time(s) = \sum_{t \in s} t.time$$

$$reward(s) = cov(s) * (1 - penalty(s))$$

$$\mathrm{cov}(s) = \sum_{t \in s} t.\mathrm{cov}erage * \sum_{m \in \mathrm{mod}(t)} \sum_{v \in rel(m)} v.perc * weigth(v.F.priority)$$

$$\therefore \mathrm{mod}(t) = \{ m \in M \mid t \in m.T \wedge m.R \neq \emptyset \}$$

$$rel(m) = \{ v \in V \mid v.M = m \}$$

$$penalty(s) = \sqrt{((poscount(s) - Wp)^2 + (negcount(s) - Wn)^2)} \Big/ 2$$

$$poscount(s) = \frac{count(t \in s \mid t.type = positive)}{count(t \in s)}$$

$$negcount(s) = \frac{count(t \in s \mid t.type = negative)}{count(t \in s)}$$

The former equations impose restrictions for the selection of S*: (a) S* must be a subset of the set of emergency tests S; (b) S* must take less than L time units to execute; and (c) no other subset of S taking less than L time units to execute should yield a higher value for the reward function than S*.

The reward function calculates an overall coverage factor for a test suite, further adjusting this value by a penalty function. The overall coverage factor for a given test suite is the sum of the coverage factor for each test case composing the suite. Finally, the coverage factor for a given test case is calculated by multiplying its observed coverage by the product of the priority weights of features evaluated by the test (according to the modules addressed by the test case).

To calculate the penalty function, the manager must first determine the desired balance between positive and negative tests in the test suite to be selected by establishing values for W_p and W_n. The penalty function uses Euclidean distance to calculate the difference between the desired proportion for each kind of test and the effective proportion of test cases in a given suite. It yields values residing in the [0, 1] interval. Higher values represent higher distances between the desired and the observed proportion in a given suite, thus making this suite less appealing for selection.

VI. CASE STUDY

To validate the proposed method, a case study will be conducted with real data from the insurance industry. A Brazilian insurance company agreed to provide information about a real software project for research purposes. This data will be collected according to the procedure presented in section II and fed into our genetic algorithm. The company has a defined software development process and uses a set of tools to support the corrective and evolutive maintenance processes. They use Subversion as a version control system and Trac to support release management. The project was developed using C# programming language, uses Microsoft Visual Studio as development environment, and NUnit to automate unit testing.

VII. CONCLUSION

This paper proposed an optimization strategy to select a set of test cases that can run in a given time frame and cover the most important features that were affected by an emergency change, according to the priority of each feature. We presented an information collection procedure to support the acquisition of data required by the optimization process, considering that the project under interest is supported by a set of commonly used tools. Finally, we presented a formal model for the problem, our first impressions about a genetic algorithm to implement the optimization process and about a case study to validate the proposed model, using information provided by a Brazilian insurance company.

This work is part of a Master degree dissertation and it is a work in progress. Currently, I am concluding the bibliographical research and the formal modeling of the optimization problem. The next steps include implementing the concepts presented in the formal model, customizing an already existing implementation of a genetic algorithm to support the proposed method, evaluating the implementation with toy data, collecting information provided by the already contacted insurance company, and evaluating the proposed method using such data.

REFERENCES

[1] R. Krishnamoorthi, S.A.Sahaaaya Arul Mary. Regression Test Priorization using Genetic Algorithms, International Journal of Hybrid Information Technology. Vol.2, No.3, July, 2009.

[2] Myers, Glenford. The Art of Software Testing. John Wiley & Sons, 1979. Primeira edição.

[3] Rothermel, G.; R. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng., vol. 27, no. 10, pp. 929-948, Oct. 2001.

[4] SWEBOK – Guide to the Software Engineering Body of Knowledge, 2004, www.swebok.org, acesso em 14/10/2009

[5] TICHY, WALTER. RCS – A System for Version Control, Software - Practice & Experience 1.1 ed., p 3, July 2005.

[6] NYMAN, Jeff, Positive and Negative Testing, GlobalTester, TechQA, http://www.sqatester.com/methodology/PositiveandNegativeTesting.htm, last access in March 29th, 2010.

[7] IEEE Standards Collection - Software Engineering. IEEE, New York - NY, 1994

[8] Sami Khuri, Thomas BÄack, and JÄorg HeitkÄotter. The zero/one multiple knapsack problem and genetic algorithms. In Proceedings of the ACM Symposium on Applied Com-puting, pages 188{193. ACM Press, 1994.

[9] PRESSMAN, S., ROGER (1997), Software Engineering - A Practitioner's Approach, 4 ed., New York, McGraw-Hill.

[10] HARROLD, M. J.; SOFFA, M. L. Selecting and using data for integration test. IEEE software, v. 8, n. 2, p. 58-65, Mar 1991.

[11] Walcott; Performing Regression Test Prioritization for Time-Constrained Execution Using a Genetic Algorithm, Master of Science Thesis, 2004.

Sophisticated Testing of Concurrent Programs

Zdeněk Letko
Brno University of Technology
Božetěchova 2, Brno
CZ 612 66, Czech Republic
Email: iletko@fit.vutbr.cz

Abstract

Search-based techniques were successfully applied to many different areas of testing but according to our knowledge there are no works that applies search-based techniques to testing of concurrent software, yet. This PhD paper describes plans and already achieved preliminary results with applying search-based techniques to testing of concurrent software. In particular, we plan to combine noise injection techniques for testing of concurrent software, various concurrency coverage measures, and several dynamic analyses with search-based optimization techniques.

1. Introduction

As concurrent programming is far more demanding than sequential, its increased use in the last decade have led to a significantly increased number of defects that appear in commercial software due to errors in synchronization. This stimulates a more intensive research in the field of detecting of such defects. Despite persistent efforts of wide community of researchers, a satisfiable solution of this problem for common programming languages like Java does not exist.

Finding concurrency defects in concurrent programs is a challenging task. The traditional testing approach does not work well here because concurrency defects often appear only under special conditions, e.g., when a specific interleaving of threads occurs. Concurrent programs are also much harder to analyze then sequential programs because one has to consider all possible interleavings and interactions of involved threads and therefore such analyzes suffer from exponential time requirements.

In this paper, we propose an approach that combines *dynamic analysis* which is able to precisely detect a concurrency defect along a given execution path, *noise injection technique* which allows us to influence thread interleaving, and *search-based technique* that is used to control noise generator with intention to maximize number of different interleavings we see during multiple executions. Our approach is sound if a sound dynamic analysis which does not produce false alarms is used and incomplete because usually we do not analyze all possible interleavings. On the other hand, our approach is able to analyze more different interleavings than traditional testing having similar time requirements and if suitable search technique is used, we analyze mostly those interleavings which are interesting to analyze.

The paper is organized as follows. The related works are briefly mentioned in the next section. The third section introduces the way how search-based techniques can be applied to concurrency testing. Then, the main directions of our research are pinpointed and finally, the already achieved preliminary results are shortly mentioned.

2. Related Works

Search-based Testing. Search-based testing applies metaheuristic search techniques to the problem of test data generation. In principle, the test adequacy criterion is encoded as a fitness function, and a search technique uses this function as a guide to achieve a maximal test adequacy. The search-based approach is very generic because different fitness functions can be defined to capture different test objectives.

Search techniques can be divided into local, global and hybrid techniques. The *local search techniques* [9], [5] seek for a better solution of the problem in the neighborhood of the currently known best solution. Such techniques provide a relatively good performance but can get stuck in a local optimum. The *global search techniques* [9], [5] seek for a better state in the whole state space. Such methods overcome the problem of local optima but usually need many more

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.14

states to be explored. Global search techniques are often implemented using evolutionary algorithms [9]. The third class of techniques, referred to as *hybrid search techniques* [5], tries to combine the previous two approaches to get rid of their disadvantages.

There have been published many works in the area of search-based testing in the past two decades. The approach has so-far been successfully applied to many different testing approaches covering functional, non functional, and state-based properties (c.f., e.g., structural, exception, stress, regression, and integration testing) [4], [5].

Finding Bugs in Concurrent Software. One of the most common approaches used for detecting concurrency bugs is *testing*, typically extended in some way to cope with the fact that concurrency bugs often appear only under very special scheduling. To increase chances of spotting a concurrency bug, various ways of *influencing the scheduling* are used (apart from running the same test many times). An example of this approach is random or heuristic noise injection used in the ConTest tool [2] or a systematic exploration of all schedules up to some number of context switches as used in the CHESS tool [10].

Testing can be improved by specific *dynamic analyses* which try to extrapolate the behaviour seen within a testing run and to warn about a possible defect even if such defect was not in fact seen in the testing run. Many dynamic analyses have been proposed for detecting special classes of bugs, such as data races [3], atomicity violations [7], and deadlocks [6]. These techniques may find more bugs than classical testing but, on the other hand, their computational complexity is usually higher and some of them can produce false alarms.

An alternative to testing and dynamic analyses is the use of various *static analyses* [11] or *model checking* [1]. Static analyses try to avoid execution of the given program or to execute it on a highly abstract level only. Model checking (sometimes viewed as a heavy-weight static analysis) tries to systematically explore all possible states of a model representing the analyzed program. These approaches are usually not precise enough and therefore produce false alarms or are too demanding and do not scale well.

Search-based Testing of Concurrent Software. According to our knowledge, search-based techniques have not been applied to concurrency testing yet. But, recently published work [12] describes application of a search technique to control state space exploration within a model checker with intention to explore areas that are more likely to contain errors. Our approach share the same idea of applying a search technique to focus on scenarios that more likely lead to an error but we focus on testing. Therefore, we do not need to construct model and handle state space of the tested application.

3. Search-based Concurrency Testing

The success of search-based software testing is based on two facts. First, the input space within which test data is sought is typically well defined but so large that it is usually infeasible to explore the whole input space. Second, the test goal can be expressed as a fitness function. The reason why the search-based testing has probably not been used for testing of concurrency is that there were not suitable definitions of input spaces and fitness functions. In the rest of this section, specifics of concurrency testing are briefly described and then definitions of input spaces and fitness functions are provided.

Specifics of Concurrency Testing. Testing is usually based on the idea that a test is executed and results (or partial results) are compared with the expected results. If the results obtained from the test correspond to the expected results, the test passes. If not, the test fails. Concurrency introduces non-determinism into the execution and one has to check that a program produces the same results for the same inputs regardless of which of the many legal orderings of operations within the program execution occurred. Therefore, testing of concurrent software is known to be very difficult since a test that has already passed in many executions may suddenly fail just because the order of operations during the execution differs from all previous executions. Concurrency testing tools therefore focus on observing many different legal orderings of program operations during *multiple executions of the same test* with the intention to find an ordering that makes the test fail.

Different legal orderings during multiple executions of a test can be achieved either by using a modified scheduler that schedules operations in an order that has not been seen yet or by the noise injection technique. The biggest problem of the modified scheduler approach is compatibility. There exist various implementations of Java and implementations of schedulers they use may differ, e.g., due to differences in underlying platforms. Therefore, we focus on the noise injection approach. In this approach, the application is instrumented and calls to a *noise maker* are injected at selected places. The noise maker tries to cause a delay (generally referred to as *noise*) when called. The selected places are those whose relative order among the threads can impact the result; such as entrances and exits from synchronized blocks, accesses to shared

variables, and calls to various synchronization primitives. The decisions whether to insert a noise can be random so different interleavings are attempted at each run or based on heuristics that try to reveal typical bugs for a particular synchronization construction.

Input Space. In our proposed approach, the input space for concurrency testing combines classic test inputs, noise maker inputs, noise parameter inputs, and an input dimension for the number of iterations to be considered. The task of identifying *classic test inputs* is the same as, for instance, for the structural testing and therefore existing approaches can be used to identify them.

The *noise maker inputs* influence the behavior of the applied noise maker. There could be two different kinds of such inputs: direct and indirect. The *direct noise maker inputs* tell the noise maker where exactly to put a noise. In this case, the search technique is used to identify a subset from the set of all possible locations in the code where it is suitable to cause a noise. The *indirect noise maker inputs* do not tell the noise maker where exactly to put a noise but set parameters influencing the procedure that decides whether to cause a noise at a particular location (e.g., a probability of causing the noise, enabling usage of some specific heuristic, etc.).

The *noise parameter inputs* define what kind of noise to use. The kind of noise is given by the type of language construction that is used by the noise maker to produce the noise and the strength of the produced noise. Finally, the *iteration count input* determines how many times the test must be performed because as was stressed above, a repeated execution of the same test with the same parameters can lead to different interleavings.

It is evident that what we do here is that we enlarge the classic input space by adding dimensions needed by the noise maker. This makes the input space much larger and hence more suited for advanced search-based techniques.

Fitness Functions. A fitness function is an objective function that prescribes the optimality of a solution and is highly dependent on the goal that a search technique tries to reach. Identifying suitable fitness functions for testing concurrency is one of our research goals and therefore we describe here only concurrency related measures on top of which the fitness functions can be built. As the most promising candidates for the inputs of fitness functions, we consider concurrency coverage measures, outputs of dynamic analyses, and classical measures like duration of the test and the number of bugs detected during a test execution.

To evaluate how well the concurrent behavior is tested, one needs some suitable coverage measures like those described in [13] (referred here as *concurrency coverage measures*). There exist several concurrency coverage models. These models usually combine code and concurrency coverage information. For instance, the *synchronization coverage* model [13] consists of tasks that describe all possible "interesting" behaviors of selected synchronization primitives. For instance, in the case of a synchronized block (defined using the Java keyword `synchronized`), the related tasks are: *synchronization visited*, *synchronization blocking*, and *synchronization blocked*. The synchronization visited task is basically just a code coverage task. The other two are reported when there is an actual contention between synchronized blocks—when a thread t_1 reached a synchronized block A and stopped because another thread t_2 was inside a block B synchronized on the same lock. In this case, block A is reported as blocked, and block B as blocking (both, in addition, as visited).

Further input information (and derived measures) can be obtained from outputs of various concurrency-related *dynamic analyses* executed along the test. Besides detecting bugs, these analyses can produce interesting data like, for instance, the set of variables that were really accessed by several different threads, the set of variables over which a race condition was detected, etc.

4. Proposed Research

A search-based testing techniques. The first direction of our research considers designing suitable fitness functions that are suitable for the needs of concurrency testing. The common use cases are to *detect a concurrency bug* (find a set of locations where to put a noise to cause an ordering of operations that enables detection of a bug), to *make the detected bug reproducible* (find a set of locations where to put a noise that cause an ordering making the bug to reveal), etc.

Reduction of input space. As was described in Section 3, the input space for concurrency testing is much larger than, for instance, in structural testing. However, since putting noise to some program locations does not make sense (e.g., locations that do not influence concurrency) and putting noise to some program locations can have the same effect as having the noise somewhere else, there is an open space for various optimizations that reduce the input space. Of course, most of the reductions must be problem-specific.

Propose a suitable incorporation of dynamic and/or static analyses. Finally, the third direction

combines testing with dynamic and static analyses in order to detect concurrency bugs. As was mentioned in Section 3, dynamic analyses can be used not only to detect concurrency bugs but also to produce inputs for the fitness function. Static analysis can be used to further analyze tests and its outputs can be used in search algorithms.

5. Already Achieved Results

We have implemented a generic platform for search-based testing called SearchBestie [8][1] that provides an environment for experimenting with search-based techniques as well as applying these methods in the area of testing. We instantiate the generic platform for concurrency testing, by linking it to the concurrency testing tool ConTest [2] from IBM. ConTest provides us with a tool for instrumenting Java bytecode, a configurable noise maker, code and concurrency coverage generators, and a listeners infrastructure that can be used to develop and apply dynamic analyses.

Our experiments with SearchBestie are in a preliminary stage. So far, we have mainly shown the concept of interconnection of SearchBestie and ConTest. However, in our experiments [8] focused on various specifics of concurrency testing, we have also obtained certain evidence that search-based techniques can be useful in concurrency testing. To prove our ideas, we compared random testing with a search-based approach which uses the simplest greedy search algorithm for finding a configuration of ConTest (indirect noise maker inputs) that provides as high concurrency coverage as possible. Since we get in some cases better results by such a primitive search than by the random approach, we believe that there is a big potential in applying better search algorithms and various heuristics in this area.

Acknowledgment

This work was partly supported by the Czech Science Foundation (proj. P103/10/0306 and 102/09/H042) and the internal BUT FIT grant FIT-10-1.

References

[1] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[2] O. Edelstein, E. Farchi, E. Goldin, Y. Nir, G. Ratsaby, and S. Ur. Framework for testing multi-threaded java programs. *Concurrency and Computation: Practice and Experience*, 15(3-5):485–499, 2003.

[3] T. Elmas, S. Qadeer, and S. Tasiran. Goldilocks: a race and transaction-aware java runtime. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 245–255, New York, NY, USA, 2007. ACM Press.

[4] M. Harman. Automated test data generation using search based software engineering. In *AST '07: Proceedings of the Second International Workshop on Automation of Software Test*, page 2, Washington, DC, USA, 2007. IEEE Computer Society.

[5] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering*, 99(RapidPosts):226–247, 2009.

[6] P. Joshi, C.-S. Park, K. Sen, and M. Naik. A randomized dynamic program analysis technique for detecting real deadlocks. In *PLDI '09: Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, pages 110–120, New York, NY, USA, 2009. ACM.

[7] Z. Letko, T. Vojnar, and B. Křena. Atomrace: data race and atomicity violation detector and healer. In *PADTAD '08: Proceedings of the 6th workshop on Parallel and distributed systems*, pages 1–10, New York, NY, USA, 2008. ACM.

[8] Z. Letko, T. Vojnar, B. Křena, and S. Ur. A platform for search-based testing of concurrent software, to appear in proceedings of PADTAD '10.

[9] P. McMinn. Search-based software test data generation: a survey: Research articles. *Softw. Test. Verif. Reliab.*, 14(2):105–156, 2004.

[10] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, and I. Neamtiu. Finding and reproducing heisenbugs in concurrent programs. In *OSDI*, pages 267–280. USENIX Association, 2008.

[11] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[12] J. Staunton and J. A. Clark. Searching for safety violations using estimation of distribution algorithms. *Software Testing Verification and Validation Workshop, IEEE International Conference on Software Testing, Verification, and Validation*, 0:212–221, 2010.

[13] E. Trainin, Y. Nir-Buchbinder, R. Tzoref-Brill, A. Zlotnick, S. Ur, and E. Farchi. Forcing small models of conditions on program interleaving for detection of concurrent bugs. In *PADTAD '09: Proceedings of the 7th Workshop on Parallel and Distributed Systems*, pages 1–6, New York, NY, USA, 2009. ACM.

1. The submitted version of the paper can be downloaded from: http://www.fit.vutbr.cz/~iletko/pub/padtad10.pdf

An optimization-based approach to software development process tailoring

Andréa Magalhães Magdaleno
[1]COPPE/PESC - System Engineering and Computer Science Program - UFRJ
[2]NP2Tec - Research and Practice Group in Information Technology - UNIRIO
Rio de Janeiro, Brazil
andrea@cos.ufrj.br

***Abstract*—** A major activity performed by the manager before starting a software project is tailoring its development process. Such activity requires information about the context under which the project will be executed, including organizational, project, and team characteristics. In addition, it also requires pondering many factors and evaluating all existing constraints. In this scenario, we claim that a balance between collaboration and discipline can be the drivers to tailor software development processes in order to meet project and organization needs. With the purpose of facilitating this balancing, it is possible to automate some of the steps to solve the problem, reducing the effort required to execute this task and improving the obtained process. Therefore, this work presents an optimization-based approach where the balancing in process tailoring is defined, modeled and briefly analyzed. This approach uses collaboration and discipline as utility functions to select the most appropriate process for a software development project, considering its current context.

Keywords- *process tailoring; optimization problem; constraint satisfaction problem*

I. INTRODUCTION

Software organizations engage in a wide variety of projects with different characteristics and no single process can be adopted in all of them. Therefore, software process tailoring is necessary. Process tailoring is the act of particularizing a general process description to derive a new process applicable to a specific situation [1].

Despite being one of the main tasks to be executed by the project manager, process tailoring is not simple. The complexity of software development and the variety of existing models [2-4] make this task arduous and inaccurate. It also requires pondering many factors and evaluating a large set of constraints. Due to this complexity, the manager usually is not able to evaluate all available combinations and chooses a process in an ad-hoc manner, based on his/her own experience, possibly selecting one that is not the best alternative for the current project.

The negative consequences of inappropriate process tailoring in organizations may involve [5]: unnecessary activities that lead to a waste of time and money; the omission of those activities that are necessary, which may affect the product quality; and the failure to comply with the organizational standard process or with international standards such as ISO or CMMI [3].

In this scenario, we claim that process tailoring should consider organization, project and team contexts, using collaboration and discipline as main drivers [6]. Discipline refers to plan and direct process during design phase, while collaboration focuses on people interaction during process execution. Both are complementary and essential in any project, but in different proportions, depending on project characteristics. Therefore, they need to be balanced.

In order to facilitate process tailoring, it is possible to support the project manager by automating some of the steps to solve the balance problem, possibly reducing the effort required to execute this activity and improving the quality and adequacy of the obtained process. Therefore, this work presents an optimization-based approach that searches a solution space composed of several candidate processes, selecting an appropriate process for a software project according to the manager's perception on the best balance between collaboration and discipline.

II. BPT PROBLEM

Aiming to offer a systematic approach to support process tailoring, a process reuse structure, based on process lines, was created. Process lines correspond to the application of software product lines [7] idea to processes. In particular, to dynamically compose and tailor processes, a Context-Based Process Line Engineering (CBPLE) [8] was proposed. It corresponds to a set of process components, organized to represent common and variants parts within a specific domain that can be reused and combined according to composition rules also based on changes in the context.

This dynamism is important because collaboration and discipline vary during throughout the software development lifecycle. It is impossible to predict, at the early stages of a project, a process that is appropriate to all situations. Therefore, process tailoring must be continuously performed as more information is available during project execution, i.e, considering the process enactment context.

The CBPLE approach is organized in two main phases: Process Domain Engineering (PDE) and Process Application Engineering (PAE) [8]. In the PDE phase, the organization creates the process line with a generic set of processes that capture the commonalities and variabilities in a domain [9]. In PAE phase, the process is tailored by the project manager from the process line, based on context information, and is ready to run. Since the aim of this work is to support the project manager in the project's process tailoring, the Balance in Process Tailoring (BPT) problem starts in the PAE phase and its definition and modeling are presented in

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.15

the next sections.

A. Problem Definition

In PDE, during process line construction, two main artifacts are produced: process line features diagram and architecture. The former represents domain characteristics and models the common and variable properties of process line members. The second one is composed of process components containing variabilities, internal properties and interfaces. These two artifacts are related since each feature has a set of components that implement it.

Both these artifacts are used in process tailoring. They are selected, according to the current context information, to indicate features and components applicable to the project. Some examples of context information are: organizational structure, organizational culture, knowledge management, problem size, client relationship, requirements stability, team experience, group working experience, and team proximity. This process line selection is out of the scope of this work because this BPT optimization problem actually focuses on project specific features and components.

Context information (CI) and process component selection rules (PCSRI and PCSRE) act as filters to select the most appropriate process components for the project (Figure 1a). Thereafter, process components are combined according to combination rules (PCCRI and PCCRE) (Figure 1b). These combinations will be evaluated considering their collaboration and discipline potential in order to determine the best combination for the project (Figure 1c). Finally, the process is composed by connecting the formerly selected components to the activities of the base process (BP), according to their interfaces (Figure 1d).

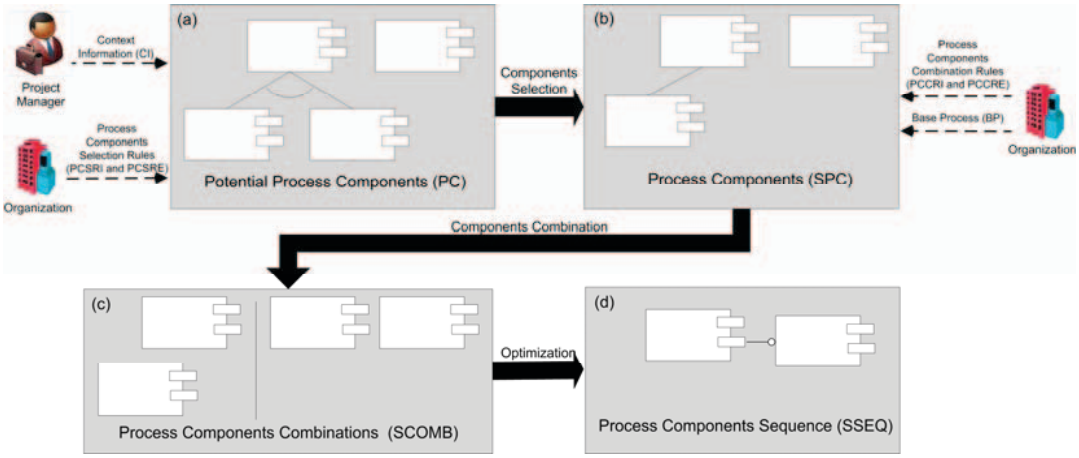


Figure 1. BPT problem phases overview

B. Problem Modeling

BPT problem has three main steps (Figure 1), detailed in next sections, and each of them adopt different techniques, such as selection/filter, combination and association/sequence.

1) Process Components Selection

The selection stage of the process components compatible with the project context is modeled as follows:

- Let PC be a set of all process components of the process architecture (Figure 1a). Each process component $PC_i \in PC$ is described by a name, a collaboration potential, a discipline potential, a set of required interfaces (input artifacts), and a set of provided interfaces (output artifacts).

  $PC = \{ PC_i \}$
  $PC_i = [name_i, pcolab_i, pdisc_i, INTRPC_i, INTPPC_i]$

- Let $INTRPC_i$ and $INTPPC_i$ be, respectively, a set of required interfaces (input artifacts) and a set of provided interfaces (output artifacts) from a given process component. Each required interface $INTRPC_{ii} \in INTRPC_i$ has a name. The definition for $INTPPC_i$ is similar to $INTRPC_i$.

  $INTRPC_i = \{ INTRPC_{ij} \}$
  $INTRPC_{ij} = [name_{ij}]$

- Let CI be a set of current context information. Each element $CI_i \in CI$ is described by a name, and a value.

  $CI = \{ CI_i \}$
  $CI_i = [name_i, value_i]$

- Let PCSRI be a set of implication rules for process components. These rules specify components that must be present depending on the context information. Each rule $PCSRI_i \in PCSRI$ is described by an identifier, an antecedent (an expression of context information), and a consequent (a set of process components).

  $PCSRI = \{ PCSRI_i \}$
  $PCSRI_i = [id_i, ANTCI_i, SEQCI_i]$
  $\therefore\ ANTCI_i = ecx$
  $SEQCI_i = \{ PC_k \}$, where $PC_k \in PC$
  $ecx = (ecx\ AND\ ecx) \vee (ecx\ XOR\ ecx) \vee (ecx\ OR\ ecx) \vee CTX$
  where, $CTX = name\ op\ value$
  where, $\exists\ CI_i \in CI \rightarrow CI_i.name = name$
  and $op \in \{ =, \neq, >, \geq, <, \leq \}$

- Let PCSRE be a set of exclusion rules for process components. These rules specify components that must not be present depending on the context information. Each rule $PCSRE_i \in PCSRE$ is described by an identifier, an antecedent (an expression of context information), and a consequent (a set of components).

  $PCSRE = \{ PCSRE_i \}$
  $PCSRE_i = [id_i, ANTCE_i, SEQCE_i]$
  $\therefore ANTCE_i = ecx$
  $SEQCE_i = \{ PC_k \}$, onde $PC_k \in PC$

To solve this first step of the problem, it is necessary to select the set of process components available for the project, which are indicated by context implication rules and are not indicated by context exclusion rules. This selection is formally expressed below:

$SPC = \{ SPC_i \in PC \mid \alpha (SPC_i) \wedge \beta (SPC_i) \}$
$\alpha(s) \leftarrow \exists PCSRI_i \in PCSRI \mid s \in PCSRI_i.SEQCI \wedge$ *evaluate*$(PCSRI_i.ANTCI, CI) = true$
$\beta(s) \leftarrow \nexists PCSRE_i \in PCSRE \mid s \in PCSRE_i.SEQCE \wedge$ *evaluate*$(PCSRE_i.ANTCE, CI) = true$

where, *evaluate*(ecx, CI) yields a boolean according to the evaluation of the *ecx* expression under context values *CI*.

2) Process Components Combination

This step takes as input a list of process components compatible with the project according to their context information (SPC) (Figure 1b) and generates as a result a list of potential combinations of these components that can fill in the activities of the base process (SCOMB) (Figure 1c). This step can be modeled as follows:

- Let $SCOMB_1$ be a set of all possible combinations of process components coming from SPC.

  $SCOMB_1 = \{ SPC_i \}$
  $SPC_i \subseteq SPC \wedge \nexists s \subseteq SPC \mid s \notin SCOMB_1$

- Let PCCRI be a set of implication rules for process components. These rules specify components that must be included depending on the presence of other components. Each rule $PCCRI_i \in PCCRI$ is described by an identifier, an antecedent (an expression of process components), and a consequent (a set of components).

  $PCCRI = \{ PCCRI_i \}$
  $PCCRI_i = [id_i, ANTI_i, SEQI_i]$
  $\therefore ANTI_i = ecp$
  $SEQI_i = \{ PC_k \}$, where $PC_k \in SPC$
  $ecp = (ecp \text{ AND } ecp) \vee (ecp \text{ XOR } ecp) \vee (ecp \text{ OR } ecp) \vee COMP \therefore COMP \in PC$

- Let PCCRE be a set of exclusion rules for process components. These rules specify components that must not be included depending on the presence of other components. Each rule $PCCRE_i \in PCCRE$ is described by an identifier, an antecedent (an expression of process components), and a consequent (a set of components).

  $PCCRE = \{ PCCRE_i \}$
  $PCCRE_i = [id_i, ANTE_i, SEQE_i]$
  $\therefore ANTE_i = ecp$
  $SEQE_i = \{ PC_k \}$, where $PC_k \in SPC$

To select the list of all feasible combinations of process components according to combination rules, it is necessary to filter the set $SCOMB_1$ and remove the elements that do not conform to these rules:

$SCOMB = \{SCOMB_i \in SCOMB_1 \mid \alpha (SCOMB_i) \wedge \beta (SCOMB_i) \}$
$\alpha(t) \leftarrow \forall PCCRI_i \in PCCRI \mid$ *eval* $(PCCRI_i.ANTI, t) = true$
$\Rightarrow \nexists PC_k \in PCCRI_i.SEQI \mid PC_k \notin t$
$\beta(t) \leftarrow \forall PCCRE_i \in PCCRE \mid$*eval* $(PCCRE_i.ANTE, t) = true$
$\Rightarrow \nexists PC_k \in RCCPE_i.SEQE \mid PC_k \in t$

where, *eval*(ecp, SCOMB) yields a boolean according to whether the components in $SCOMB_i$ attend to *ecp*.

3) Process Components Sequence

The sequencing of process components to fill in the activities of a base process can be modeled as follows:

- Let BP be a base process defined by the organization, such as lifecycles (waterfall, iterative, etc.). The base process is described by a name and a set of activities.

  $BP = [name, ATIV]$

- Let ATIV be a set of activities belonging to a given base process. Each activity $ATIV_i \in ATIV$ is described by a name, a set of predecessor activities, a set of required interfaces (input artifacts), and a set of provided interfaces (output artifacts).

  $ATIV = \{ ATIV_i \}$
  $ATIV_i = [name_i, PATIV_i, INTRATIV_i, INTPATIV_i]$

- Let $INTRATIV_i$ and $INTPATIV_i$ be, respectively, a set of required interfaces and a set of provided interfaces from a given activity. Each required interface $INTRATIV_{ii} \in INTRATIV_i$ has a name. The definition for $INTPATIV_i$ is similar to $INTRATIV_i$.

  $INTRATIV_i = \{ INTRATIV_{ij} \}$
  $INTRATIV_{ij} = [name_{ij}]$

To solve the problem it is necessary to combine the process components that will be part of each activity:

$SCPATIV = \{ SCPATIV_i \}$
$SCPATIV_i = [AT_i, \{ C_{ij} \}]$
$\therefore AT_i \in ATIV$
$\forall ATIV_i \in ATIV \subset BP \mid \exists SCPATIV_i.AT = ATIV_i$
$C_{ii} \leftarrow$ *components*$(AT_i, SCOMB_i)$

The *components* method returns a set of process components sets, where each process component set satisfies activity (AT_i) interfaces and can be used to implement this activity. This method is detailed below:

components$(AT_i, SCOMB_i)$
$SUB=\{ SUBi \}$
$\therefore SUB_i \subseteq SCOMB_i \wedge \nexists s \subseteq SCOMB_i \mid s \notin SUB$
$\Phi \leftarrow \phi$; selected components
For each $SUB_i \in SUB$
$\forall r \in AT_i.INTRATIV \rightarrow \exists c \in SUB_i \mid r \in c.$ INTRCP

$\forall s \in AT_i.INTPATIV \rightarrow \exists c \in SUB_i \mid s \in c.\ INTPCP$
$\forall c \in SUB_i \rightarrow \forall r \in c.\ INTRCP \mid$
$r \in \bigcup SUB_i.INTPCP \vee r \in AT_i.INTRATIV$
If all these conditions are satisfied
$\Phi \leftarrow \Phi \cup SUB_i$

C. Utility Functions

There may be several solutions to the proposed problem. These solutions have different qualities, defined by utility functions. Since BPT involves the balancing of collaboration and discipline, they can be used as quality factors for the solution. Thus, we considered two utility functions that are responsible for determining the selection of components that satisfies all problem constraints and maximizes, respectively, the collaboration or discipline potential to the project.

Let S be a set of possible solutions that satisfy all constraints. Let $Colab(S_k)$ (or $Disc(S_k)$) be a function that returns the collaboration (or discipline) from a given process component S_k. The selected solution $S^* \in S$ is chosen according to the corresponding utility equations.

- ***Greatest Collaboration:*** calculated as the sum of the collaboration potential for each component included in the tailored process.

$$\nexists\ S_j \in S \mid Colab(S_j) > Colab(S^*)$$

- ***Greatest Discipline*:** calculated as the sum of the discipline potential for each component included in the tailored process.

$$\nexists\ S_j \in S \mid Disc(S_j) > Disc(S^*)$$

D. BPT Complexity and Search Space Analysis

The BPT's complexity is influenced by the number of process components (p) and activities (a). Considering these factors, the problem complexity is $O(p!^a)$. However, when constraints are added to the problem formulation, the effort required to find the solutions tends to decrease, due to the number of paths that do not attend the constraints or have less utility than the best solution. Therefore, the number of possible process components to fill in each activity reduces to p', where $p' < p$.

Thus, in practice, the search space grows in proportion to $(p'!^a)$. For instance, considering that a base process, like waterfall lifecycle, has an average of 5 activities and taking OpenUP as an example, we can suppose that a development model has an average of 20 components. Since the process line includes components for almost 5 development models, the complete search space has approximately $(100!^5)$ possible solutions.

Since an exact solution to the problem cannot be found in feasible time, it points to the possibility of using a heuristic algorithm, like genetic algorithms, but this implementation has not been done so far and it is included in thesis plan.

III. Conclusion

This paper presented an approach to balance collaboration and discipline in process tailoring. This balancing was defined and modeled as a three-step model. This modeling concludes the second year of PhD research.

The results obtained in the first two years of work concerns: (i) characterization of software development models reconciliation, by conducting a systematic review of the literature that showed the importance of a dynamic process tailoring; (ii) elaboration of the proposal to balance collaboration and discipline in process tailoring using context management [6]; (iii) establishment of CBPLE approach [8]; (iv) definition of the initial set of context information relevant to software development processes.

For future work, we intend to more clearly define how the utility functions must be calculated, computationally model the problem, choose which technique is most suitable to be applied to solve this problem, and develop a tool that runs an optimization algorithm and generates a suggestion of a tailored process, according to the given parameters.

Furthermore, it is also important to validate: if a heuristic technique will find better solutions with less computational effort than a random search; if the resulting process is better than one that would be adapted without the support of the solution; investigate the degree of influence that each of the input parameters has on the quality of the generated solution and response time, through a sensitivity analysis; and verify the solution applicability in real software developments by conducting some case studies.

Acknowledgment

This work is funded by CNPq under grant nº. 142006/2008-4 and is part of FAPERJ's research project nº E-26/103.049/2008. The author also would like to thank Prof. Marcio Barros for his collaboration in this work.

References

[1] M. Ginsberg e L. Quinn, *Process Tailoring and the Software Capability Maturity Model*, SEI-CMU, 1995.

[2] K. Beck, et al., "Manifesto for Agile Software Development," 2001.

[3] M.B. Chrissis, et al., *CMMI: Guidelines for Process Integration and Product Improvement*, Boston, MA, USA: Addison-Wesley, 2006.

[4] E.S. Raymond, *The Cathedral & the Bazaar*, O'Reilly Media, 2001.

[5] O. Pedreira, et al., "A systematic review of software process tailoring," *SIGSOFT Software Engineering Notes*, vol. 32, 2007, pp. 1-6.

[6] A.M. Magdaleno, "Balancing Collaboration and Discipline in Software Development Processes," *Doctoral Symposium of International Conference on Software Engineering (ICSE)*, Cape Town, South Africa, 2010, pp. 331-332.

[7] L. Northrop, "SEI's software product line tenets," *IEEE Software*, vol. 19, 2002, pp. 32-40.

[8] V.T. Nunes, et al., "Context-Based Process Line," *International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Portugal: 2010, pp. 277-282.

[9] D. Rombach, "Integrated Software Process and Product Lines," *Unifying the Software Process Spectrum*, Berlin/Heidelberg: Springer-Verlag, 2006, pp. 83-90.

Search Based Optimization of Requirements Interaction Management

Yuanyuan Zhang
CREST Centre
King's College London
London, UK
yuanyuan.zhang@kcl.ac.uk

Mark Harman
CREST Centre
King's College London
London, UK
mark.harman@kcl.ac.uk

***Abstract*—There has been much recent interest in Search Based Optimization for Requirements Selection from the SBSE community, demonstrating how multi-objective techniques can effectively balance the competing cost and value objectives inherent in requirements selection. This problem is known as release planning (aka the 'next release problem). However, little previous work has considered the problem of Requirement Interaction Management (RIM) in the solution space. Because of RIM, there are many subtle relationships between requirements, which make the problem more complex than an unconstrained feature subset selection problem. This paper introduces and evaluates archive-based multi-objective evolutionary algorithm, based on NSGA-II, which is capable of maintaining solution quality and diversity, while respecting the constraints imposed by RIM.**

***Keywords*-Requirements, RIM, NSGA-II, Search-based Software Engineering**

I. Introduction

In the release planning for software development, the requirements interdependency relationship is an important element which reflects how requirements interact with each other in a software system. Furthermore, it also directly affects requirements selection activity as well as requirements traceability management, reuse and the evolution process.

According to Carlshamre et al.,

> "The task of finding an optimal selection of requirements for the next release of a software system is difficult as requirements may depend on each other in complex ways" [1].

Some requirements might have technical, structural or functional correlations that need to be fulfilled together or separately, or one requirement might be the prerequisite of another. The analysis and management of dependencies among requirements is called Requirements Interaction Management (RIM) which is defined as

> "the set of activities directed towards the discovery, management, and disposition of critical relationships among sets of requirements" [2].

Robinson et al. [2] gave the definition of requirement interaction:

> "Two requirements R_1 and R_2 is said to interact if (and only if) the satisfaction of one requirement affects the satisfaction of the other."

RIM consists of a series of activities related to requirement dependencies which are complex and challenging tasks.

Few previous authors [3], [4], [5] focus on the role of requirements dependencies in the solution space. However, dependencies can have a very strong impact on the development process in a typical real world project. Bagnall et al. [3] only considered the *Precedence* dependency type, representing the relationship as a directed, acyclic graph. Its vertices are denoted as individual requirements and its edges, directed from one vertex to another, are denoted as the *Precedence* dependency between the requirements. Greer and Ruhe extended the work by adding the *And* dependency type together with *Precedence* as the constraints in their EVOLVE model. Franch and Maiden applied the *i** approach to model dependencies for COTS component selection.

In this paper, the Search-based Requirements Selection Optimization framework allows for the five most common types of requirements dependencies. The objectives are to investigate the influences of requirements dependencies on the automated requirements selection process for release planning and to validate the ability of the proposed framework to find the optimal balance in the solution space for release planning under different circumstances.

The study is based on the assumption that the dependence identification activity has been completed. Here we present the most common interaction types found in the requirements literature. These will be studied in this paper. The paper will show how multi-objective SBSE can be adapted to take account of RIM.

And	Given requirement R_1 is selected, then requirement R_2 has to be chosen.
Or	Requirements R_1 and R_2 are conflicting to each other, only one of R_1, R_2 can be selected (Exclusive OR).
Precedence	Given requirement R_1 has to be implemented before requirement R_2.
Value-related	Given requirement R_1 is selected, then this selection affects the value of requirement R_2 to the stakeholder;
Cost-related	Given requirement R_1 is selected, then this selection affects the cost of implementing

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.16

requirement R_2.

The rest of the paper is organized as follows: In Section II the problem is formalized as an SBSE problem, while Section III describes the data sets and algorithms used. Section IV presents the results for dependence aware requirements optimization and discusses the findings. Section V describes the context of related work in which the current paper is located. Section VI concludes the paper.

II. Fitness Function

In the context of Value/Cost-based requirements assignments analysis, the dependencies among requirements need to be accounted for within the fitness function. This section describes our fitness computation and how we incorporate RIM into this fitness.

Assume that the set of possible software requirements is denoted by:

$$\Re = \{r_1, \ldots, r_n\}$$

The requirements array R is defined by:

$$R = \begin{Bmatrix} r(1,1) & r(1,2) & \cdots & r(1,i) & \cdots & r(1,n) \\ r(2,1) & r(2,2) & \cdots & r(2,i) & \cdots & r(2,n) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ r(j,1) & r(j,2) & \cdots & r(j,i) & \cdots & r(j,n) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r(n,1) & r(n,2) & \cdots & r(n,i) & \cdots & r(n,n) \end{Bmatrix}$$

First, we formalise the RIM constraints that were listed informally in the introduction to this paper.

And Define an equivalence relation ξ on the requirements array R such that $r(i,j) \in \xi$ means that r_i is selected if and only if requirement r_j has to be chosen.

Or Define an equivalence relation φ on the requirements array R such that $r(i,j) \in \varphi$ (equivalently $r(j,i) \in \varphi$) means that at most one of r_i, r_j can be selected.

Precedence Define a partial order χ on the requirements array R such that $r(i,j) \in \chi$ means that requirement r_i has to be implemented before requirement r_j.

Value-related Define a partial order ψ on the requirements array R such that $r(i,j) \in \psi$ means that if the requirement r_i is selected, then its inclusion affects the value of requirement r_j for the stakeholder.

Cost-related Define a partial order ω on the requirements array R such that $r(i,j) \in \omega$ means that if the requirement r_i is selected, then its inclusion affects the cost of implementing requirement r_j.

In addition, the relations ξ, φ and χ should satisfy

$$\xi \bigcap \varphi = \emptyset \quad \wedge \quad \xi \bigcap \chi = \emptyset$$

in order to guarantee consistency in the requirements dependency relationship.

The fitness function with dependency constraints is defined as follows:

$$\text{Maximize } f_1(\overrightarrow{x}) = \sum_{i=1}^{n} score_i \cdot x_i$$

$$\text{Maximize } f_2(\overrightarrow{x}) = -\sum_{i=1}^{n} cost_i \cdot x_i$$

subject to

$$x_i = x_j \quad \text{for all pairs} \quad r(i,j) \in \xi \quad \text{(And constraints)}$$

$$x_i \neq x_j \ \vee \ x_i = x_j = 0 \quad \text{for all pairs} \quad r(i,j) \in \varphi \quad \text{(Or constraints)}$$

$$x_i = 1 \wedge x_j = 1 \quad \vee \quad x_i = 1 \wedge x_j = 0 \quad \vee \quad x_i = x_j = 0$$

$$\text{for all pairs} \quad r(i,j) \in \chi \quad \text{(Precedence constraints)}$$

In terms of Value-related and Cost-related requirements dependencies, they cannot be transformed from dependencies into constraints. So the fitness values of a solution are changed directly when there exists a Value-related or Cost-related dependency, as follows:

$$\text{If} \quad x_i = x_j = 1 \quad \text{for all pairs} \quad r(i,j) \in \psi \quad \Rightarrow$$

$$\text{Update Fitness Value of } f_1(\overrightarrow{x})$$

$$\text{If} \quad x_i = x_j = 1 \quad \text{for all pairs} \quad r(i,j) \in \omega \quad \Rightarrow$$

$$\text{Update Fitness Value of } f_2(\overrightarrow{x})$$

III. Experimental Set Up

To assess the likely impact of requirements dependencies on the automated requirements selection process, a set of empirical studies were carried out. This section describes the test data sets used and the search-based algorithm applied to the requirements interaction management.

A. Data Sets

The "27 combination random data sets" used in previous studies [6] were adopted in this studies. These are the basis of the data sets we will use in the empirical studies. The "27-random" data sets were generated randomly according to the problem representation. These synthetic test problems were created by assigning random choices for value and cost. The range of costs were from 1 through to 9 inclusive (zero cost is not permitted). The range of values were from 0 to 5 inclusive (zero value is permitted, indicating that the stakeholder places no value on this requirement).

Table I
27 COMBINATION RANDOM DATA SETS

	R_{small}	R_{medium}	R_{large}
C_{small}	$C_s\ R_s\ D_{low}$ $C_s\ R_s\ D_m$ $C_s\ R_s\ D_h$	$C_s\ R_m\ D_{low}$ $C_s\ R_m\ D_m$ $C_s\ R_m\ D_h$	$C_s\ R_l\ D_{low}$ $C_s\ R_l\ D_m$ $C_s\ R_l\ D_h$
C_{median}	$C_m\ R_s\ D_{low}$ $C_m\ R_s\ D_m$ $C_m\ R_s\ D_h$	$C_m\ R_m\ D_{low}$ $C_m\ R_m\ D_m$ $C_m\ R_m\ D_h$	$C_m\ R_l\ D_{low}$ $C_m\ R_l\ D_m$ $C_m\ R_l\ D_h$
C_{large}	$C_l\ R_s\ D_{low}$ $C_l\ R_s\ D_m$ $C_l\ R_s\ D_h$	$C_l\ R_m\ D_{low}$ $C_l\ R_m\ D_m$ $C_l\ R_m\ D_h$	$C_l\ R_l\ D_{low}$ $C_l\ R_l\ D_m$ $C_l\ R_l\ D_h$

Table II
SCALE RANGE OF '27-RANDOM' DATA SET

	Small	Medium	Large
No. of Stakeholders	2-5	6-20	21-50
No. of Requirements	1-100	101-250	251-600
	Low	Medium	High
Density of Matrix	0.01-0.33	0.34-0.66	0.67-1.00

This simulates the situation where a stakeholder ranks the choice of requirements (for value) and the cost is estimated to fall in a range: very low, low, medium, high, very high. The number of stakeholders and the number of requirements are divided into three situations, namely, small scale, medium scale and large scale; the density of the stakeholder-requirement matrix is defined as low level, medium and high level. Table I lists the combination of all cases schematically. As can be seen in Table II, the data set divides the range of a variable into a finite number of non-overlapping intervals of unequal width.

Any randomly generated, isolated data set clearly cannot reflect real-life scenarios. We do not seek to use our pseudo random generation of synthetic data as a substitute for real world data. Rather, we seek to generate synthetic data in order to explore the behavior of our algorithms in certain well defined scenarios. The use of synthetic data allows us to do this within a laboratory controlled environment. Specifically, we are interested in exploring the way the search responds when the data exhibits a presence or absence of correlation in the data. As well as helping us to better understand the performance and behavior of our approach in a controlled manner, this also allows us to shed light on the real world data, comparing results with the synthetic data.

To explore this, in the empirical studies, we generated four data sets exhibiting different scales and densities using the approach to data set generation depicted in Table I and Table II. We named the sets A, B, C and D. In the A data set, the parameter choices were chosen to be "medium" and the density of the stakeholder-requirement matrix was also chosen to be "medium". The parameters of the data set were randomly generated within the given scale intervals. More concretely, the number of requirements is 230, the number of stakeholders is 11 and the density of matrix is 0.53.

Table III
SCALE OF A, B, C AND D DATA SETS: EXPLORATION OF THE CONFIGURATION SPACE FOR RIM

	R_{small}	R_{medium}	R_{large}
C_{small}			**C:** $C_s\ R_l\ D_m$
C_{median}		**A:** $C_m\ R_m\ D_m$	
C_{large}	**B:** $C_l\ R_s\ D_m$		**D:** $C_l\ R_l\ D_h$

Table IV
A, B, C AND D DATA SETS: CHOOSING A VARIETY OF RIM DISTRIBUTIONS

Data Set	No. of Stakeholders	No. of Requirements	Density of Matrix
A	11	230	0.53
B	34	50	0.39
C	4	258	0.51
D	21	412	0.98

Following the same principle, the B, C and D data sets were generated. The scales and densities chosen and the specific parameters created are listed in Table III and Table IV.

In the four data sets that were generated, all the requirements were initially created to be independent. To introduce the dependency, relationships among requirements are added randomly but with respect to constraints. Five two-dimensional arrays $And(i,j)$, $Or(i,j)$, $Pre(i,j)$, $Val(i,j)$ and $Cos(i,j)$ ($1 \leq i \leq n$ and $1 \leq j \leq n$) are defined to represent the five requirements dependency types.

$$And(i,j), Or(i,j)\ and\ Pre(i,j) \in \{0,1\}$$

$And(i,j) = 1 \wedge And(j,i) = 1$ if requirement r_i and r_j have *And* dependency and 0 otherwise; $Or(i,j) = 1 \wedge Or(j,i) = 1$ if requirement r_i and r_j have *Or* dependency and 0 otherwise; $Pre(i,j) = 1 \wedge Pre(j,i) = 0$ if requirement r_i and r_j have *Precedence* dependency.

The above three dependency arrays are bit vectors which compactly store individual boolean values, as flags to indicate the relationship between requirements. As each random relationship is created, we check to ensure that the *And*, *Or* and *Precedence* dependence constraints are respected, thereby guaranteeing the generation of a valid instance.

In the $Val(i,j)$ and $Cos(i,j)$ arrays, the values are not 0 or 1, but rather the extent of impact of the *Value* or *Cost* which are expressed as a numerical percentage. $Val(i,j) \neq 0$ if requirements r_i and r_j have a *Value-related* dependency; $Cos(i,j) \neq 0$ if requirements r_i and r_j have a *Cost-related* dependency.

B. Algorithms

The search algorithms used in this work were NSGA-II [7] and a modified version we implemented that is specifically constructed to produce a good Pareto front for RIM-constrained problem. Our modification is inspired by Praditwong and Yao's Two-Archive multi-objective evolutionary optimization algorithm [8]. Results will be presented to compare the performance of two algorithms.

Keeping the final set of non-dominated solutions is good enough for general multi-objective optimization work. However, when we take account for requirements dependencies, the selected optimal non-dominated solutions might not respect the dependency constraints. As a result some solutions might have to be eliminated. This may mean rejecting otherwise 'optimal' solutions in favor of previously considered and otherwise less optimal solutions.

In order to preserve these potential candidate solutions, this paper introduces an archive-based variation of the NSGA-II algorithm to retain near optimal solutions (maintaining diversity and quantity of the solutions) based on constraints.

All search-based approaches were run for a maximum of 50,000 fitness function evaluations. The population was set to 500. We used a simple binary GA encoding, with one bit to code for each decision variable (the inclusion or exclusion of a requirement). The length of a chromosome is thus equivalent to the number of requirements. Each experimental execution of each algorithm was terminated when the generation number reached 101 (i.e after 50,000 evaluations). All genetic approaches used tournament selection (the tournament size is 5), single-point crossover and bitwise mutation for binary-coded GAs. The crossover probability was set to $P_c = 0.8$ and mutation probability to $P_m = 1/n$ (where n is the string length for binary-coded GAs). In the archive based NSGA-II algorithm, the total capacity of the archives was set to 500.

IV. Empirical Studies and Results

This section presents the experiments carried out to investigate the results in the presence of requirement dependencies and to compare the performance of two search algorithms.

There are two types of empirical study: a Dependency Impact Study (DIS) and a Scale Study (SS). Data set A is used for DIS and data sets B, C and D are used for SS.

In DIS, the experiment is designed for the purpose of evaluating the impacts of five different dependency types on the requirements selection process. Data set A is used throughout the DIS experiment in order to set up a uniform baseline for comparison. Three experiments were conducted in DIS, described as follows:

1) Applying the NSGA-II algorithm to data set A with and without dependencies separately, in order to carry out the comparison of the results of each dependency type (five types individually).
2) Applying the NSGA-II and archive based NSGA-II to data set A aiming to compare the performances of two algorithms under the dependency constraints (five types individually).
3) Considering dependence relationships as a whole in order to seek to investigate the difference among the solutions generated by the two algorithms.

In SS, we report results concerning the performance of the two algorithms as the data sets increase in size. There are three data sets B, C and D with the number of stakeholders ranging from 4 to 34 and the number of requirements ranging from 50 to 412.

In both studies, the five dependency types can be divided into two categories: fitness-invariant dependency (And, Or and Precedence) and fitness-affecting dependency (Value-related and Cost-related). Therefore we will discuss the two scenarios separately in each study. In addition, the same dependency density levels were used for all five dependencies. That is, we assume that they are equally common in the requirements correlations.

A. Dependency Impact Study

1) Aims: Three goals need to be achieved in DIS listed as follows:

1) The Pareto front should cover the maximum number of different situations and provide a set of well distributed solutions.
2) The solutions contained in the Pareto front should be as close as possible to the optimal Pareto front of the problem.
3) The solutions are required to pass the evaluation without failure to meet constraints.

2) And, Or and Precedence: In the first part of the section, we present the results of applying the NSGA-II and the archive based NSGA-II algorithms to handle *And*, *Or* and *Precedence* requirements dependencies. The results generated by the standard NSGA-II algorithm are shown in Figures 1, 2 and 3; and the results from the archive based NSGA-II algorithm are shown in Figures 4, 5 and 6 separately.

The '+', '◯' and '$*$' symbols plotted in the figures denote the final non-dominated solutions found. Each solution represents a subset of requirements selected. The '+' symbol represents the solutions found without regard to requirement dependencies. They are also marked in grey to

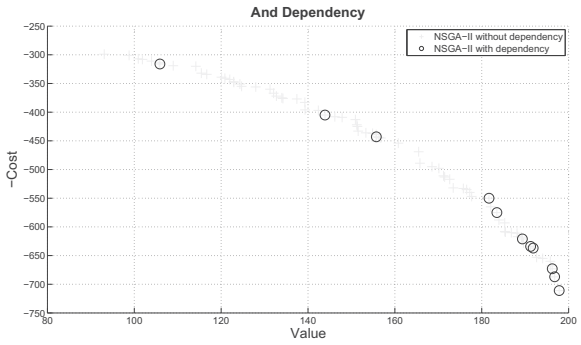


Figure 1. Results Comparison: with and without *And* dependency

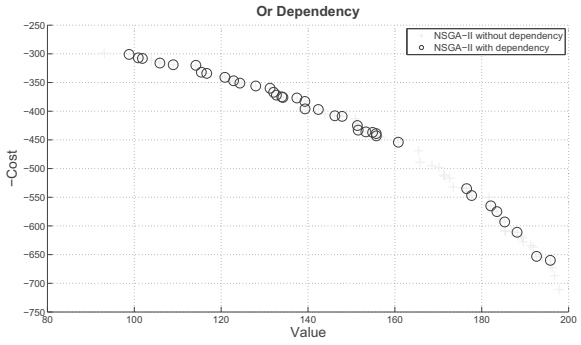


Figure 2. Results Comparison: with and without *Or* dependency

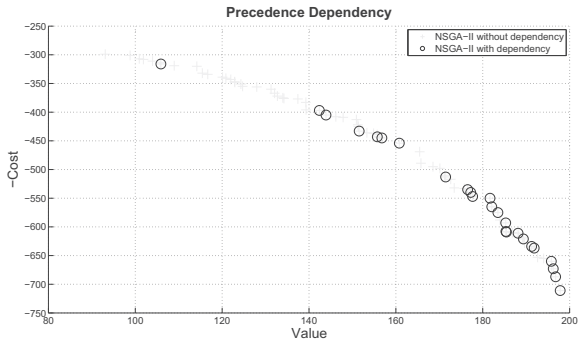


Figure 3. Results Comparison: with and without *Precedence* dependency

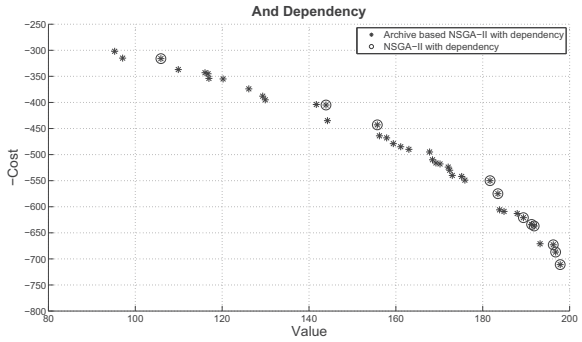


Figure 4. Results Comparison: Original and Degenerated Pareto fronts with *And* dependency

distinguish them from the others (which do take account of dependencies).

In Figures 1, 2 and 3, we observe that the shapes of Pareto fronts, consisting of a number of grey '+' symbols, are the same. These are solutions generated by the NSGA-II algorithm without consideration of requirements dependency relationship and so they are expected to be identical. They are used as the baseline to explore the impact of three types of dependencies on the requirements selection results.

We illustrate the results in Figure 1 when the *And* dependency relationships exist among the requirements. '◯' symbols indicate solutions which respect the *And* dependence. As can be seen from the graph, all the '◯' solutions still fall on the Pareto front composed of grey '+' symbols. However, there is a large decrease in the number of '◯' solutions compared to the number of '+' solutions. In other words, a few solutions survived and the rest were eliminated (from the selection) because of the failure to meet dependency constraints. Another obvious observation drawn from this graph is that the distribution of '◯' solutions is neither as smooth nor as uniform as the '+' solutions. That is, a certain number of big or small gaps exist among them. These two observations indicate that the algorithm can neither provide good solutions in quantity nor maintain a good diversity (quality) on the Pareto front under *And* dependency constraints.

Compared to the results of *Or* and *Precedence* dependencies in Figures 2 and 3, the downward trends in the number of '◯' solutions are roughly the same, but the extent is different. Similar observations can be made from the two figures: the results show a slight decrease in the number of the solutions. Moreover, the distribution is more continuous, exhibiting a few small gaps among the solutions.

In conclusion, the shapes of Pareto fronts (results) are affected by the different dependency constraints to a different extent. The *And* dependency problem appears to denote a tighter constraint than the *Or* and *Precedence* dependencies for search-based requirements optimization. The latter two denote problems for which it is relatively easy to find the solutions that satisfy the constraints.

To explore these findings in more detail, we designed a more robust adaptive algorithm for both tight and loose constraints; the archive-based NSGA-II algorithm. The results for three types of constraints are shown in Figures 4, 5 and 6 respectively. The '∗' denotes the solution generated by the archive-based version of the NSGA-II algorithm and the '◯' by NSGA-II.

From Figure 4, we can see the archive based '∗' solutions actually reach all the points on the previous '◯' Pareto front, sharing all the common points generated by NSGA-II. The

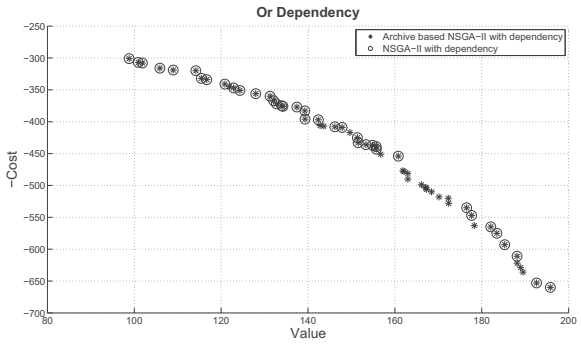


Figure 5. Results Comparison: Original and Degenerated Pareto fronts with *Or* dependency

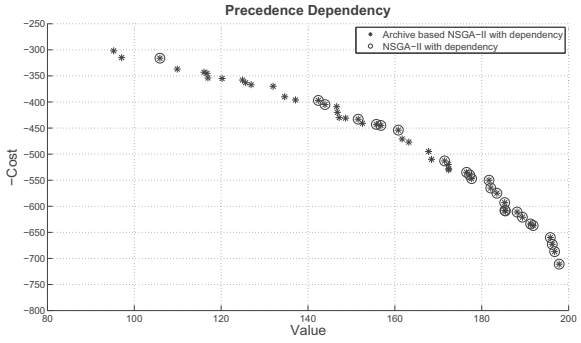


Figure 6. Results Comparison: Original and Degenerated Pareto fronts with *Precedence* dependency

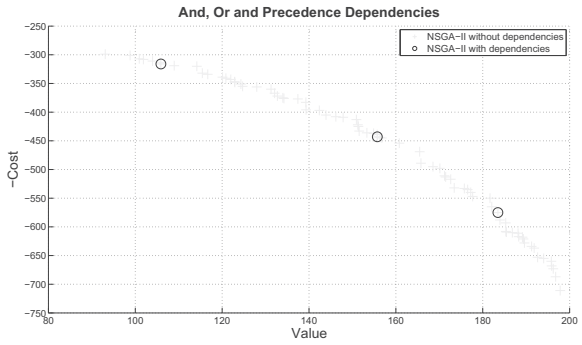


Figure 7. Results Comparison: with and without *And*, *Or* and *Precedence* dependencies

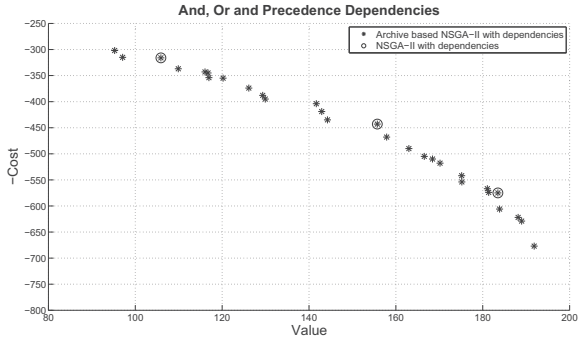


Figure 8. Results Comparison: Original and Degenerated Pareto fronts with *And*, *Or* and *Precedence* dependencies

Pareto front in this problem is orientated towards the upper right. The improved algorithm provided a *degenerated* '$*$' Pareto front.

The *degenerated* Pareto front means that the '$*$' front generated seems to become worse when compared to the grey '+' front, but it discovers a larger number of good solutions to fill the gaps while meeting the constraints. That is, the diversity of solutions is significantly improved and the number of solutions on the Pareto front is also increased. The algorithm generated similar results when dealing with *Or* and *Precedence* dependency constraints, as illustrated in Figures 5 and 6.

These results indicate that the archive is able to 'repair' the gaps which open up in the Pareto front when RIM constraints are imposed. In this way, the archive-based technique can provide stable and fruitful solutions, which are not merely 'good enough' but also 'robust enough' under the strict constraints that characterize the problem.

Finally, all three dependencies were taken into consideration to access their overall combined impact. In the Figure 7, the '◯' solutions denote the final results that satisfy all the dependencies constraints. Combining *And*, *Or* and *Precedence* dependencies together, the constraints become much tighter. It is easy to see that very few '◯' solutions remain on the Pareto front based on the NSGA-II algorithm. By contrast, Figure 8 shows a smooth, relatively non-interrupted Pareto front, consisting of '$*$' solutions, generated by archive based NSGA-II.

3) Value-related and Cost-related: In this section we focus on the last two types of requirement dependencies: *Value-related* and *Cost-related*. These two impose no constraint on the fitness function but have direct influence on the fitness value.

The results are illustrated in Figure 9. There are four subgraphs in the figure: (a) is the original Pareto front without dependency generated by NSGA-II; (b) and (c) show the results under *Value-related* and *Cost-related* dependencies respectively; (d) presents the changed Pareto front when combining these two dependencies.

We observe that the shapes of the four Pareto fronts produced are different. They are not like the previous results of the first three dependencies: eliminating solutions on the unconstrained Pareto fronts or using a 'degenerate' front are not viable for *Value/Cost-related* constraints. *Value/Cost-related* relationships among the requirements can directly contribute to an increase or a decrease in the fitness values obtained for a selected solution. In this way, the shape of the Pareto front is changed more than once without dependency.

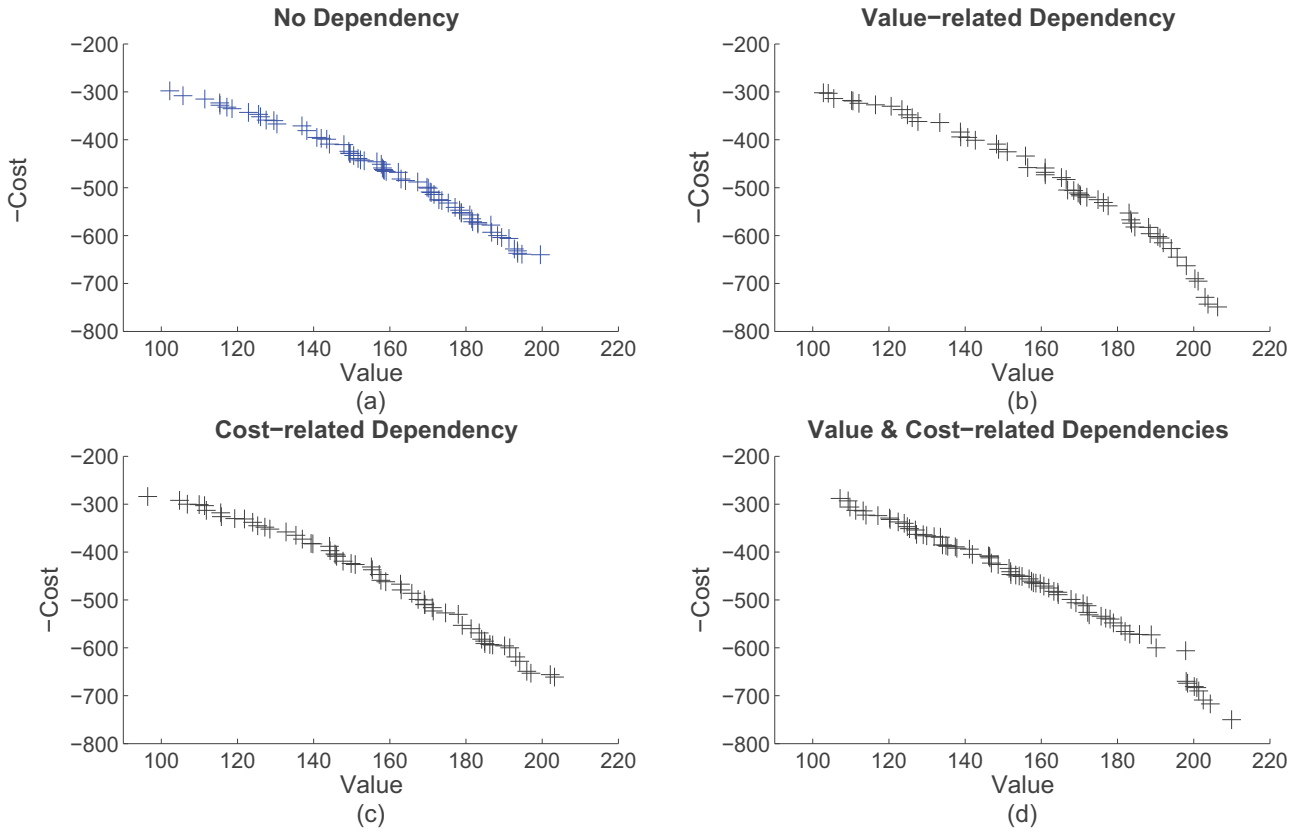


Figure 9. Results Comparison: Original and Changed Pareto front with Value-related and Cost-related dependencies

B. Scale Study

In this section, we report on the second empirical study – the Scale Study. The results are presented in the Figures 10, 11 and 12. As described at the beginning of Section IV, the techniques were applied to three data sets B, C and D generated from the smaller scale to a relatively larger one in terms of the number of stakeholders involved and the number of requirements fulfilled. The details are listed in Table III and Table IV.

In this study, all three dependency constraints are considered together. The results are plotted in one graph for each data set. In the figures, the grey Pareto front, consisting of a number of '+' solutions, denotes the results without handling dependencies generated by the NSGA-II algorithm; the '◯' solutions are the survivors after selection for meeting the constraint; the '∗' solutions which are produced by the archive based NSGA-II algorithm constitute the *degenerated* Pareto front.

When the problem is gradually scaled up, from the graphs we can see that the number of the '◯' solutions consistently and rapidly decreases. As illustrated in Figure 12, the '◯' solutions have a poor spread over the Pareto front. By contrast, the '∗' solutions still fill the gaps among the '◯' solutions and produce a relatively smooth and continuous Pareto front.

These three data sets B, C and D are considered using the same proportion of possible dependencies (6% of the number of requirements). Another observation from the three figures is that the distance between the original '+' Pareto front and the *degenerated* '∗' Pareto front is wider in Figure 12 than in Figure 10. The Pareto fronts move towards the lower left part of the solution space, in order to find near-optimal solutions that have a good spread as well as having (more than) enough candidate solutions.

V. Related Work

Dependence analysis is a part of the overall traceability problem for requirements engineering. The task of requirement traceability is to identify and document traceability links among requirements and between requirements and following SE activities in both a forwards and backwards direction [9]. Requirements traceability is crucial for the success of the system. It enables detection of the conflicting requirements and reduction of missing requirements. Furthermore, it can track the progress of a project, assess the impact of various changes and provide complete information

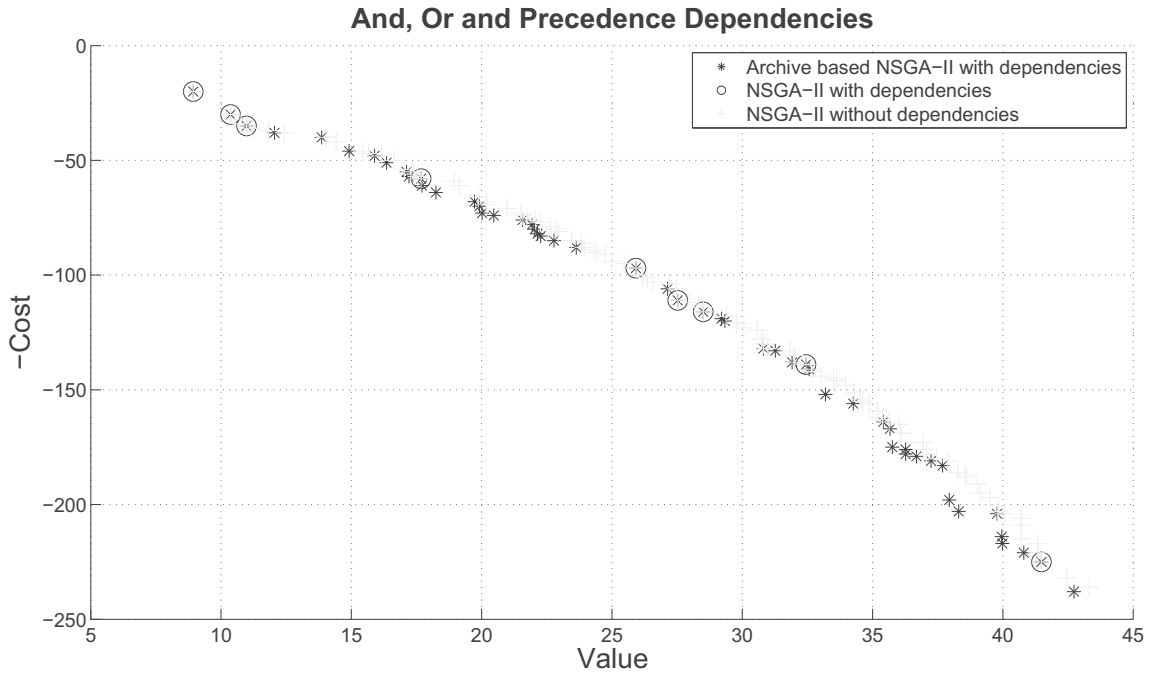


Figure 10. Results for Data Set B: 34 Stakeholders, 50 Requirements

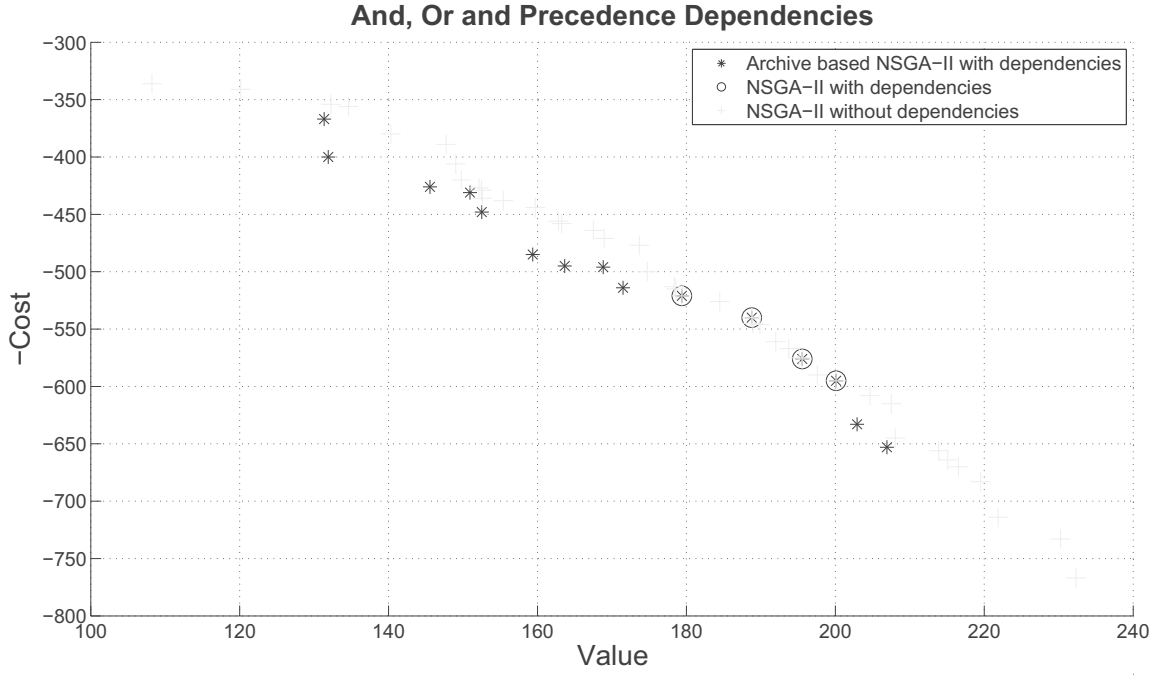


Figure 11. Results for Data Set C: 4 Stakeholders, 258 Requirements

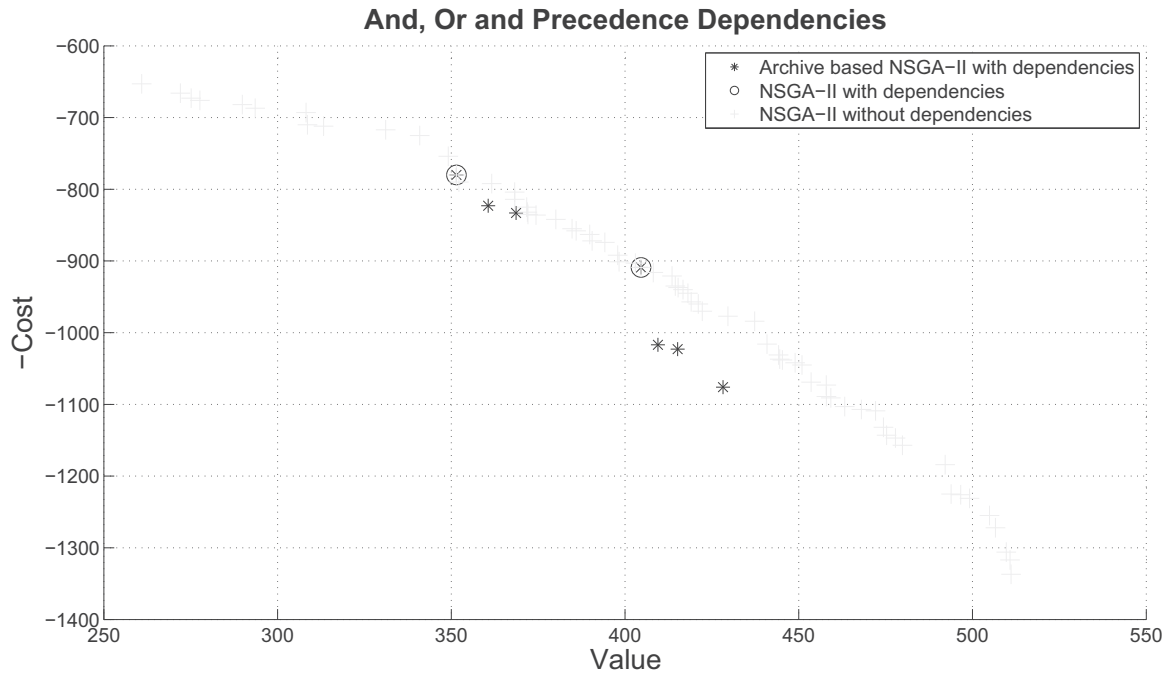


Figure 12. Results for Data Set D: 21 Stakeholders, 412 Requirements

in the SE lifecycle.

There are many ways to represent traceability links. The traceability matrix [10] and cross references [9] are both regarded as good practice which have been widely used in industry. In addition, a large number of requirements tools support traceability management [9]. One of the most famous tools is DOORS (Dynamic Object Oriented Requirements System) [11].

Pohl [12] proposed the *traceability meta model* to establish a traceability structure, which included dependence models aiming to describe the relations between trace objects. Karlsson et al. [13] opened up the discussion on supporting requirements dependencies in the requirements selection process. Robinson et al. [2] provided the basic concepts and scope of Requirements Interaction Management (RIM). They introduced RIM process in general and a historical perspective of RIM. Carlshamre and Regnell [14] described a two-dimensional (*scope* and *explicitness*) representation to investigate different types of dependencies. Subsequently Carlshamre et al. [1] extended their work and carried out an industrial survey of requirements interdependencies in software product release planning. A functional and value-related dependence classification scheme was proposed in detail. The survey also tried to find the possible relationship between the dependence types and development contexts. Dahlstedt and Persson [15], [16] provided an overview of research work concerning comparing and validating the different requirements dependencies classification frameworks.

There was some work which suggested that proper treatment of RIM should take account of different types of requirements interactions [1], [5], [13], [15], [17].

VI. Summary

This paper presented Requirements Interaction Management (RIM) and has taken RIM into consideration in the automated requirements selection process for the release planning problem. Five basic requirement dependencies were introduced. The first three types were considered to be constraints within the fitness functions; the latter two directly involved in the performance.

A“27 combination random data sets” model was generated to develop a procedure in order to better approximate real world situations. Two variable factors were considered in the data generation model. One is the different levels of data set scales which are related to the number of requirements and the number of stakeholders; the other is the density of the data sets.

To simulate the release planning selection process under requirements dependencies, two empirical studies were carried out; the Dependency Impact Study (DIS) which is designed to investigate the influences of five different dependency types and the Scale Study (SS) which concerns the performance of the two search techniques when the data sets scale up.

The same data set was used throughout the DIS experiment aiming to set up a uniform baseline for influence

comparison. The results of the empirical studies illustrated that the *And* dependency appears to denote a tighter constraint than the *Or* and *Precedence* dependencies for search-based requirements optimisation. When all three dependencies were taken into consideration to access their overall combined impact, the constraints in this case became much tighter. For *Value-related* and *Cost-related* dependencies, they directly contributed to an increase or a decrease in the fitness values and further changed the shape of the Pareto front.

In SS, three data sets from smaller scale to a relatively larger one were applied. The results showed that Archive based NSGA-II could produce a smooth, relatively non-interrupted Pareto front compared to NSGA-II. When the data set was gradually scaled up, the number of solutions generated by the latter consistently and rapidly decreases. Instead, Archive based NSGA-II could still find better solutions both in diversity and quantity.

RIM is of vital importance from a software release planning point of view. For instance, the certain optional requirements can be put into one release or be separated into several releases according to their dependency relationships in order to save implementation cost and increase revenue.

Aided by search-based automated RIM, the requirements engineer can faster and more easily address this problem. For any non-trivial problem, many factors need to be considered in the requirements selection process. It is always important to look at the requirements from different perspectives. Unlike human-based search, automated search techniques carry with them no bias. They automatically scour the search space for solutions that best fit the (stated) human assumptions in the fitness function.

ACKNOWLEDGMENT

The authors would like to thank Kathy Harman assisted with proof reading.

REFERENCES

[1] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An Industrial Survey of Requirements Interdependencies in Software Product Release Planning," in *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE '01)*, 2001.

[2] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements Interaction Management," Georgia State University, Tech. Rep. 99-7, August 1999.

[3] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley, "The Next Release Problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, December 2001.

[4] X. Franch and N. A. Maiden, "Modelling Component Dependencies to Inform Their Selection," in *Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS '03)*, ser. LNCS, vol. 2580. Ottawa, Canada: Springer, 10-12 February 2003, pp. 81–91.

[5] D. Greer and G. Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach," *Information & Software Technology*, vol. 46, no. 4, pp. 243–253, March 2004.

[6] Y. Zhang, E. Alba, J. J. Durillo, S. Eldh, and M. Harman, "Today/Future Importance Analysis," in *Proceedings of International Conference on Genetic and Evolutionary Computation (GECCO '10)*. Portland, Oregon, USA: ACM, 7-11 July 2010, to appear.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[8] K. Praditwong and X. Yao, "A New Multi-Objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm," in *Proceedings of he 2006 International Conference on Computational Intelligence and Security (CIS '06)*, vol. 1. Guangzhou, China: IEEE Press, 3-6 November 2006, pp. 286–291.

[9] O. C. Z. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceedings of the 1st International Conference on Requirements Engineering (RE '94)*. Colorado Springs, Colorado, USA: IEEE Computer Society, 18-21 April 1994, pp. 94–101.

[10] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards, "Implementing Requirements Traceability: A Case Study," in *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering (RE '95)*. York, UK: IEEE Computer Society, 27-29 March 1995, pp. 89–95.

[11] "IBM Rational DOORS (Dynamic Object Oriented Requirements System), http://www.telelogic.com/Products/doors/."

[12] K. Pohl, *Process-Centered Requirements Engineering*. Research Studies Press, 1996.

[13] J. Karlsson, S. Olsson, and K. Ryan, "Improved Practical Support for Large-scale Requirements Prioritizing," *Requirements Engineering Journal*, vol. 2, no. 1, pp. 51–60, 1997.

[14] P. Carlshamre and B. Regnell, "Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes," in *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA '00)*. London, UK: IEEE Computer Society, 4-8 September 2000, pp. 961–965. [Online]. Available: http://computer.org/proceedings/dexa/0680/06800961abs.htm

[15] Å. G. Dahlstedt and A. Persson, "Requirements Interdependencies - Moulding the State of Research into A Research Agenda," in *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (RefsQ '03)*, Klagenfurt/Velden, Austria, 16-17 June 2003.

[16] ——, *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg, 2005, ch. 5 Requirements Interdependencies: State of the Art and Future Challenges, pp. 95–116.

[17] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements Interaction Management," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 132–190, June 2003.

Using Interactive GA for Requirements Prioritization

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

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

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

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

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

I. Introduction

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

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

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

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

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

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

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

and future work are presented in Section VI.

II. Related Work

Several techniques used in the current prioritization approaches consist of assigning a rank to each requirement in a candidate set according to a specific criterion, such as value of the requirement for the customer or requirement development cost. The rank of a requirement can be expressed as its relative position with respect to the other requirements in the set, as in *Bubble sort* or *Binary search* procedures, or as an absolute measure of the evaluation criterion for the requirement, as in *Cumulative voting* [7]. Other, alternative techniques consist of assigning each requirement to one specific class among a set of predefined priority classes, as for instance in *Numerical Assignment* [7], [8] and in *Top-ten requirements* [9].

Among the pairwise techniques, CBRank [10] adopts a preference elicitation process that combines sets of preferences elicited from human decision makers with sets of constraints which are automatically computed through machine learning techniques; it also exploits knowledge about (partial) rankings of the requirements that may be encoded in the description of the requirements themselves as requirement attributes (e.g., priorities or preferences).

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

Among the methods exploiting genetic algorithms for requirements management, in [11] the EVOLVE method supports continuous planning for incremental software development. This approach is based on an iterative optimization method supported by a genetic algorithm. In [12], the authors focus on the specific Multi-Objective Next Release Problem (MONRP) in Requirements Engineering and present the results of an empirical study on the suitability of weighted and Pareto optimal genetic algorithms, together with the non-dominated sorting genetic algorithm (NSGA-II), presenting evidence to support the claim that NSGA-II is well suited to the MONRP. These two works overcome the problems of scalability of methods such as AHP, but they do not produce a total ordering of requirements. Rather, they group requirements for the planning of the next release.

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

III. Approach

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

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

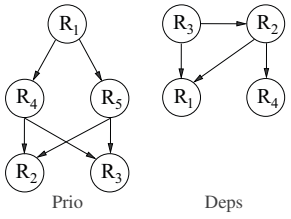


Table I
Requirements with priority and dependencies

Let us consider the five requirements listed in Table I. For each requirement we consider the priority expressed by the analyst who collected them and the dependencies the requirement may have with other requirements. For conciseness, in Table I we omit other important elements of a requirement (e.g., the textual description). Constraints and properties of requirements can be represented by means of a *precedence* graph. In the following, we always make the assumption that precedence graphs are actually DAGs (directly acyclic graphs). In fact, cyclic (sub-)graphs provide no useful information, since they represent precedences that are compatible with any requirements ordering. In a precedence graph, an edge between two requirements indicates that, according to the related constraint or property, the

requirement associated with the source of the edge should be implemented before the target requirement. Edges may be weighted, to actually quantify the strength or importance of such a precedence and an infinite weight may be used for precedence relations that must necessarily hold in the final ordering of the requirements. For simplicity, in this section we assume that each edge has a weight equal to one, but all the arguments, as well as the algorithm described in the next section, can be applied almost unchanged to weighted precedence graphs. Weights could be given to individual edges or, alternatively, to the whole precedence graph associated with a constraint or property.

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

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

Table II
PRIORITIZED REQUIREMENTS AND RELATED DISAGREEMENT

The interactive genetic algorithm we use for requirements prioritization evolves a population of individuals, each representing a candidate prioritization. Table II shows 6 individuals (i.e., 6 prioritizations). An individual is a permutation of the sequence of all requirements to be prioritized. In order to evolve this population of individuals, their fitness is first evaluated. We measure the fitness of an individual as the *disagreement* between the total order encoded by the individual and the partial orders encoded in the precedence graphs constructed from the requirement documents. Further precedence relationships are obtained from the user during the prioritization process. Such relations are also considered in the computation of the disagreement. They constitute the *elicited precedence graph*, which is initially empty. In our running example, it would be the third precedence graph, to be added to Prio and Deps.

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

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

Table III
PAIRWISE COMPARISONS TO RESOLVE TIES

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

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

has only one pair in the disagreement, (R_4, R_5).

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

After collecting user input in terms of pairwise comparisons, the existing elicited precedence graph is augmented with the new precedence edges. The appearance of cycles in the elicited graph indicates the existence of contradictory information, since a cycle is compatible with any ordering of requirements. Hence, whenever a cycle is introduced in the elicited graph, we ask the user to break it.

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

IV. Algorithm

In this section, we describe an interactive genetic algorithm that implements the approach presented in the previous section. Before introducing the algorithm, we provide a formal definition of the intuitive notion of disagreement(taken from [10]), which plays a fundamental role when evaluating the fitness of an individual and when deciding which pairwise comparisons to elicit from the user. We give the definition in the general case where two partial orders are compared. A special case, quite relevant to the proposed method, is when one or both orders are total ones.

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

The disagreement between two (partial or total) orders ord_1, ord_2, defined upon the same set of elements R, is the set of pairs in the transitive closure[1] of the first order, ord_1^*, that appear reversed in the second order closure ord_2^*. A measure of disagreement is given by the size of the set $dis(ord_1, ord_2)$.

Algorithm 1 Compute prioritized requirements

Input R: set of requirements
Input $ord_1, ..., ord_k$: partial orders defining priorities and constraints upon R ($ord_i \subseteq R \times R$ defines a DAG)
Output $< R_1, ..., R_n >$: ordered list of requirements

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

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

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

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

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

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

Inside the main evolutionary iteration (steps 11-22), the first operation to be performed is *selection* (step 11). While any selection mechanism could be used in principle with this algorithm, we experimented the best performance when using tournament selection. When evolutionary optimization gets stuck for G generations (i.e., a locally minimum disagreement with available constraints is reached), the resulting population is sorted by decreasing disagreement and ties are determined for the best (top fraction) individuals in the population. If there are ties, the user is resorted to in order to resolve them. Specifically, the pairs in the disagreement between equally scored individuals are submitted to the user for pairwise comparison; in this work we take care that each pair is not presented to the user multiple times during the process. The result of the comparison is added to the elicited precedence graph ($eliOrd$).

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

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

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

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

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

V. CASE STUDY

We applied the IGA algorithm to prioritize the requirements for a real software system, as part of the project ACube (Ambient Aware Assistance) [13]. ACube is a large research project funded by the local government of the Autonomous Province of Trento, in Italy, aiming at designing a highly technological smart environment to be deployed in nursing homes to support medical and assistance staff. In such context, an activity of paramount importance has been the analysis of the system requirements, to obtain the best trade off between costs and quality improvement of services in specialized centers for people with severe motor or cognitive impairments. From the technical point of view, the project envisages a network of sensors distributed in the environment or embedded in users' clothes. This technology should allow monitoring the nursing home guests unobtrusively, that is, without influencing their usual daily life activities. Through advanced automatic reasoning algorithms, the data acquired through the sensor network are going to be used to promptly recognize emergency situations and to prevent possible dangers or threats for the guests themselves. The ACube project consortium has a multidisciplinary nature, involving software engineers, sociologists and analysts, and is characterized by the presence of professionals representing end users directly engaged in design activities.

As a product of the user requirements analysis phase, 60 user requirements (49 technical requirements[2]) and three macro-scenarios have been identified. Specifically, the three macro scenarios are: (i) "localization and tracking to detect falls of patients", (ii) "localization and tracking to detect patients escaping from the nursing home", (iii) "identification of dangerous behaviors of patients"; plus (iv) a comprehensive scenario that involves the simultaneous presence of the previous three scenarios.

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

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

or

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

Table IV summarizes the number of technical requirements for each macro-scenario. Together with the set of technical requirements, two sets of technical constraints have been collected during requirements elicitation: *Priority* and *Dependency*, representing respectively the priorities among requirements and their dependencies. In particular, the *Priority* constraint has been built on the basis of the users' needs and it is defined as a function that associates each technical requirement to a number (in the range 1–500), indicating the priority of the technical requirement with respect to the priority of the user requirements it is intended to address. The *Dependency* feature is defined on the basis of the dependencies between requirements and is a function that links a requirement to the set of other requirements it depends on.

[2]We consider only functional requirements.

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

Table IV
THE FOUR MACRO-SCENARIOS AND THE NUMBER OF TECHNICAL REQUIREMENTS ASSOCIATED WITH THEM.

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

A. Research questions

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

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

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

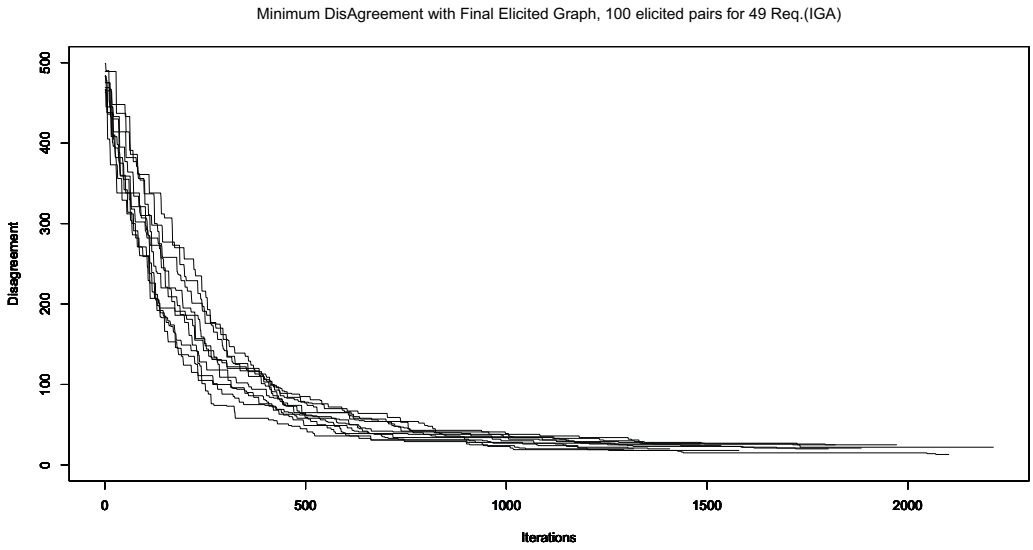


Figure 1. Disagreement of the best individual in each population across generations in the ALL scenario (the $timeOut$ is 1080 seconds)

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

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

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

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

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

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

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

GS) with probability p_e. We varied the probability of user error p_e from 5% to 20%.

B. Results

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

Figure 1 shows how the best individual in each population converges toward a low value of the final fitness function (i.e., disagreement with the final precedence graphs, including all elicited constraints). 30 executions of the IGA algorithm are considered, on all the 49 requirements of the case study. While different runs exhibit slightly different behaviors and the final value obtained for the minimum disagreement differs from run to run, we can observe that the trend is always a steep decrease, which indicates the algorithm is indeed optimizing (minimizing) the final fitness function value, even though this is only partially known during the optimization iterations. Similar plots have been obtained for the 3 separate scenarios, FALL, ESC, MON.

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

We can observe that the sanity check is passed (i.e., both GA and IGA outperform RAN by a large degree). We can also see how interaction improves the performance of non-interactive GA. While the improvement is minor with 25 elicited pairs, it gets quite substantial with 50 and it is definitely a major one with 100 elicited pairs. It should be noticed that state of the art algorithms for requirement prioritization, such as AHP, require exhaustive pairwise

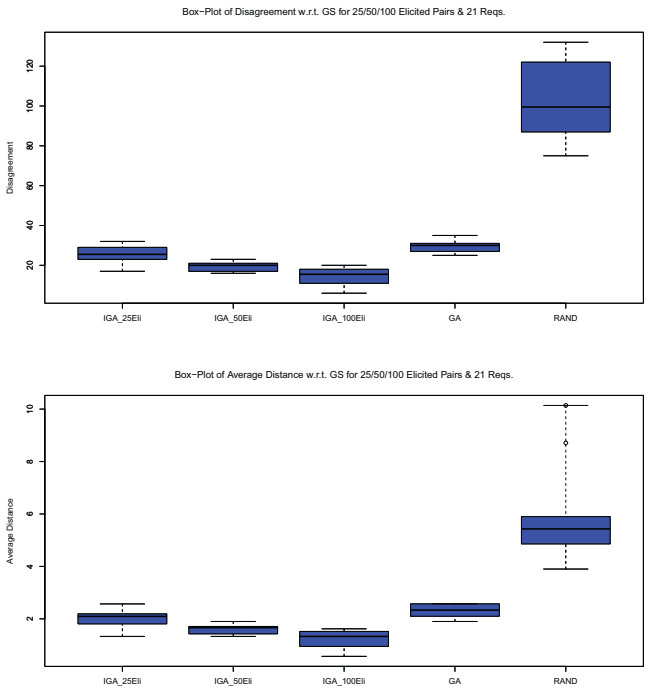


Figure 2. Disagreement with (top) and distance from (bottom) GS after eliciting 25/50/100 pairs from the user in the MON scenario

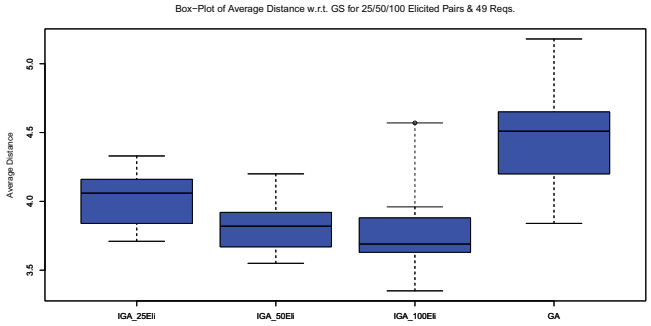


Figure 3. Average distance of each requirement position from the respective GS position after eliciting 25/50/100 pairs in the ALL scenario

comparisons, which in the ALL scenario means 49 * 48 / 2 = 1176 comparisons (210 in MON). Hence, even as many as 100 elicited pairs represent a small fraction (around 50% for MON; 10% for ALL) of the comparisons elicited by AHP.

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

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

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

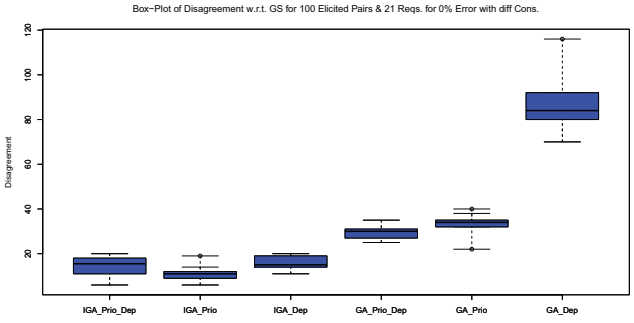


Figure 4. Performance of IGA and GA under different availability of precedence constraints (either Prio, Dep or both) in the MON scenario

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

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

Figure 5 shows how the performance of the IGA algorithm degrade at increasing user error rates. We show the results obtained for ALL, but similar plots are available for the

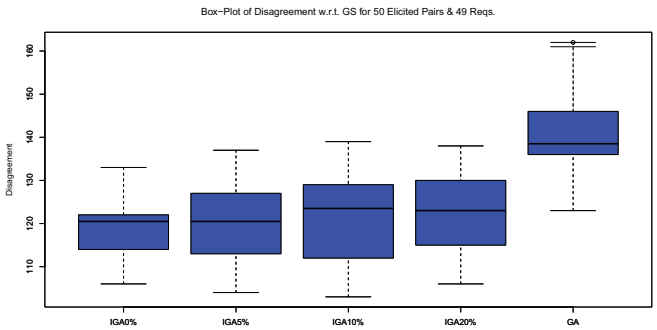


Figure 5. Performance of IGA at increasing user error rates in the ALL scenario

smaller, separate scenarios. As expected, the disagreement tends to increase as long as the user makes errors at increasing rate. However, such increase is not steep, indicating that the algorithm is quite robust with respect to the errors possibly affecting the user responses to pairwise comparisons. Even at a user error probability p_e as high as 20%, IGA keeps some margin of improvement over GA.

C. Discussion

Based on the data collected from our experiments (due to space limitations, we showed only a portion of them in this paper), we can answer positively to all our research questions. IGA converges even though the fitness function is known in its complete form only at the end of the elicitation process (RQ1). When comparing the prioritizations produced by the considered algorithms with GS, both in terms of disagreement and of position distance, IGA outperforms substantially GA (and RAN), especially when a higher number of pairwise comparisons can be carried out (RQ2). The improvement of IGA over GA is even higher when limited ranking information is available a-priori (RQ3). Moreover, the behavior of the algorithm is robust with respect to the presence of elicitation errors committed by the user (RQ4).

AHP, which represents the state of the art for requirement prioritization, requires exhaustive pairwise comparison. This is often impractical. It is definitely so in our case study, where it would be impossible to elicit 1176 comparisons from the user for the ALL scenario (in the subscenarios exhaustive elicitation is also impractical, requiring between 210 and 338 pairwise comparisons). In the ALL scenario, with an artificial user which makes no error, AHP would produce a final ordering which has zero disagreement with GS. By eliciting only a small fraction (at most 10%) of the pairs required by AHP, we cannot expect to be equally good in terms of disagreement. In fact, the final disagreement produced by IGA is not zero. However, if we look at the average distance of each requirement from the position it

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

D. Threats to validity

The main threat to the validity of our case study concerns the *external validity*, i.e., the possibility to generalize our findings to requirements collected for other systems and having different features. Since we conducted one case study, the natural way to corroborate our findings and make them applicable to other systems is by replicating this study on other, different cases. Although considering just one case, we did our best to exploit it as much as possible. Specifically, we considered four macro scenarios in addition to the complete one, and we considered the same set of requirements, but with different precedence constraints associated. This enlarges a bit the scope of generalizability of the results.

Other threats to validity regard the *construct validity*, i.e., the observations we made to test our research hypotheses. Specifically, we used disagreement and requirement position distance as the metrics that determine the algorithm's performance. Other metrics may be more meaningful or more appropriate. On the other hand, disagreement is widely used in the related literature and position distance looked like an interpretable alternative. Another construct validity threat might be related with to simple user error model we used to simulate a user who occasionally makes errors. We will experiment with more sophisticated models in the future.

VI. Conclusions and future work

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

Ant Colony Optimization for the Next Release Problem

A comparative study

José del Sagrado, Isabel María del Águila, Francisco Javier Orellana
Dept. of Languages and Computation
University of Almería
04120 Almería, Spain
e-mail: jsagrado@ual.es

Abstract— **The selection of the enhancements to be included in the next software release is a complex task in every software development. Customers demand their own software enhancements, but all of them cannot be included in the software product, mainly due to the existence limited resources. In most of the cases, it is not feasible to develop all the new functionalities suggested by customers. Hence each new feature competes against each other to be included in the next release. This problem of minimizing development effort and maximizing customers' satisfaction is known as the next release problem (NRP). In this work we study the NRP problem as an optimisation problem. We use and describe three different meta-heuristic search techniques for solving NRP: simulated annealing, genetic algorithms and ant colony system (specifically, we show how to adapt the ant colony system to NRP). All of them obtain good but possibly suboptimal solution. Also we make a comparative study of these techniques on a case study. Furthermore, we have observed that the suboptimal solutions found applying these techniques include a high percentage of the requirements considered as most important by each individual customer.**

Keywords: ant colony optimization; simulated annealing; genetic algorithm; requirement selection; next release problem

I. INTRODUCTION

The selection of the enhancements to be included in the next software release is a complex task in every software development. Enhancements to include cannot be randomly selected since there are many factors involved which can turn into a source of problems during software development if they are not treated in an appropriated way. Within this scenario, customers demand their own software enhancements, but all of them cannot be included in the software product, mainly due to the existence limited resources (availability of men-moth in a given software project). In most of the cases, it is not feasible to develop all the suggested new functionalities. Hence each new feature competes against each other to be included in the next release. The next release problem (NRP) [1] has as goal meet the customer's needs, minimizing development effort and maximizing customers satisfaction.

The task of selecting a set of requirements, which until now only appeared when defining new versions of widely distributed software products (e.g. new versions of commercial software products), becomes important within the new approaches of agile software development. In the Manifesto for agile software development [3] was defined a set of principles, focused on the improvement of the weak points of heavyweight methodologies: people versus processes, software development versus documentation development, customer collaboration versus contract negotiation [20]. Furthermore, agile approaches assume that development processes are complex and unpredictable, subject to many changes, usually related to the availability of resources, delivery time, or requirements to build. Hence one of the main points of agile methodologies is their fast adaptation to any change.

Agile methodologies are based on an iterative and incremental process, performing series of iterations or sprints which last from one to several weeks, where new functionality is developed. At the end of each sprint a deliverable, which can be used as a prototype, is obtained. Scrum [3, 21, 22] is one of the most polished agile methods. In Scrum every development handles a list or backlog where all requirements are gathered. Each one of these requirements represents software enhancements that are still to be implemented. Before a sprint is started (in scrum a sprint takes four weeks), backlog has to be updated and requirements have to be reprioritized according to their importance. Requirements to be included must always be agreed by customers and developers. The requirement selection is made based on several aspects such as: the profit acquired by its inclusion in the software product, its development effort, the availability of development resources, etcetera. Once the sprint starts, the list of requirements does not suffer any modification until it ends. An improved prototype is obtained as result of a sprint and it has to be evaluated again by the team in order to adjust backlog.

The next release problem is considered as an optimization problem [1, 15, 11] widely known in Search Based Software Engineering [14, 4, 13], so it is suitable for the application of meta-heuristic techniques. Different approaches can be found in the literature to tackle with requirement selection problem, for example, [1] and [2] apply greedy and simulated annealing techniques, [11] use genetic algorithms in software release planning and [18] propose the use of ant colony optimization (ACO).

The works of Salui and Ruhe [19] Zhang et al. [23], Finkelstein et al. [9, 10] and Durillo et al. [8], study the NRP

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.18

problem from the multi-objective point of view, either as an interplay between requirements and implementation constraints [19] or considering multiple objectives as cost-value [23] or different measures of fairness [9, 10], or applying genetic algorithm [8].

In this work we show how to apply ant colony optimization in NRP. First, we study how to adapt the original formulation of NRP, so that it can be addressed with the different meta-heuristic techniques applied (simulated annealing, genetic algorithms and ant colony systems). In particular for the case of genetic algorithms, we define a new crossover operation and we show that crossover and mutation operations are not closed for NRP. This is the reason for introducing a repair operation. In the ant colony system, we study the representation of the problem as a graph, so that it can be traversed by the ants in their search for a solution. Once the different techniques are adapted, we perform a simple visual parameter tuning based on the results obtained for a given instance of the problem of requirement selection taken from [11], using different parameters values. However, we do not make a deep study of the parameters tuning process. Also, we make a visual comparison of the results obtained by the different techniques making use of the best parameters configurations.

The rest of this paper is structured as follows. Section 2 gives a mathematical description of NRP. Section 3 describes the three meta-heuristics algorithms used. Section 4 is devoted to experimentation, paying special attention to parameters selection and techniques comparison. Finally, Section 5 draws conclusions and future works.

II. Problem Definition

Let $\boldsymbol{R} = \{r_1, r_2, \cdots, r_n\}$ be the set of requirements that are still to be implemented (i.e. the backlog). These requirements represent enhancements to the current system and are suggested by a set of m customers and are candidates to be included in the next sprint. Customers are not equally important. So, each customer i will have an associated weight w_i, which measures its importance. Let $\boldsymbol{W} = \{w_1, w_2, \cdots, w_m\}$ be the set of customers' weights.

Each requirement r_j in $\boldsymbol{R}$ has an associated development cost e_j, which represents the effort needed in its development. Let $\boldsymbol{E} = \{e_1, e_2, \cdots, e_n\}$ be the set of requirements' efforts. On many occasions, the same requirement is suggested by several customers. However, its importance or priority may be different for each customer. Thus, the importance that a requirement r_j has for customer i is given by a value v_{ij}. The higher the v_{ij} value, the higher is the priority of the requirement r_j for customer i. A zero value for v_{ij} represents that customer i has not suggested requirement r_j. All these importance values v_{ij} can be arranged under the form of an $m \times n$ matrix

$$\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{m1} & \cdots & v_{mn} \end{pmatrix} \quad (1)$$

The global satisfaction, s_j, or the added value given by the inclusion of a requirement r_j in the sprint, is measured as a weighted sum of the its importance values for all the customers and can be formalized as

$$s_j = \sum_{i}^{m} w_i \, v_{ij}. \quad (2)$$

In order to define a sprint, we have to select a subset of requirements $\hat{\boldsymbol{R}}$ included in the backlog $\boldsymbol{R}$, which maximize satisfaction and minimize development effort. The satisfaction and development effort of a sprint can be obtained, respectively, as:

$$sat(\hat{\boldsymbol{R}}) = \sum_{j \epsilon \hat{\boldsymbol{R}}} s_j, \quad (3)$$

$$eff(\hat{\boldsymbol{R}}) = \sum_{j \epsilon \hat{\boldsymbol{R}}} e_j, \quad (4)$$

where j is an abbreviation for requirement r_j. As each sprint has assigned limited resources, then development effort cannot exceed a certain bound B.

We can formulate the requirement selection problem for a given sprint with a particular effort bound as an optimisation problem:

$$\begin{aligned} &\text{maximise } \sum_{j \epsilon \hat{\boldsymbol{R}}} s_j \\ &\text{subject to } \sum_{j \epsilon \hat{\boldsymbol{R}}} e_j \leq B. \end{aligned} \quad (5)$$

This problem is known to be NP-hard [1, 2] as it can be represented as a 0-1 knapsack problem.

III. Algorithms Applied

This section describes in more detail the three different search algorithms applied for solving the next release problem: simulated annealing, genetic algorithms and ant colony optimization.

A. Simulated Annealing

Simulated annealing emulates the energy changes that occur in a system of particles when its temperature is reduced till the system reaches a state of equilibrium. At higher temperatures drastic changes in the system are allowed, whereas at lower temperatures only minor changes are allowed. This cooling scheduling has as goal to reduce the energy state of the system, taking the system from an arbitrary initial energy state to a final state with the minimum possible energy.

This algorithm was first used in optimization problems in [16], and it has been also applied in several works to solve the next release problem [1, 2]. In the case of the requirement selection problem simulated annealing searches for a subset of requirements $\hat{\boldsymbol{R}}$ within the set of all subsets of n requirements $\mathcal{P}(\boldsymbol{R})$. The size of this search space is $|\mathcal{P}(\boldsymbol{R})| = 2^n$. We can represent a subset of requirements $\hat{\boldsymbol{R}}$ in this

space as a vector $\{x_1, x_2, \cdots, x_n\}$, where $x_i \in \{0, 1\}$. If requirement $r_i \in \hat{\boldsymbol{R}}$, then $x_i = 1$ and otherwise $x_i = 0$.

The search starts from an initial solution $\boldsymbol{S}$ generated by selecting requirements r_j with highest satisfaction values s_j, until the maximum development effort B is reached, or until there are not available requirements whose development effort is less than or equal to the free development effort capacity (i.e. the difference between the maximum development effort B and the sum of the development efforts of selected requirements). Then we explore a set of solutions close to the current solution, i.e. its neighborhood. We say, following [1, 2], that a vector $\boldsymbol{S}' = \{x_1', x_2', \cdots, x_n'\}$ is a neighbor of a solution $\boldsymbol{S} = \{x_1, x_2, \cdots, x_n\}$, if they differ exactly in one element $x_i' \neq x_i$ and $eff(\boldsymbol{S}') \leq B$. So, the neighbourhood of a solution $\boldsymbol{S}$, denoted as $nei(\boldsymbol{S})$, will be the set of all its neighbours.

As the objective is to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, we use satisfaction, $sat(\boldsymbol{S})$, as *fitness* or evaluation function.

Let $\boldsymbol{S}$ be the current solution and $\mathbf{S}'$ be a new solution in the neighborhood of $\boldsymbol{S}$, $\boldsymbol{S}' \in nei(\boldsymbol{S})$. The probability of making the transition from the current solution S to the candidate solution $\boldsymbol{S}'$, i.e. the *acceptance probability*, is defined as

$$p(S, S', T) = \begin{cases} 1 & if\ \Delta E > 0 \\ e^{\frac{\Delta E}{T}} & otherwise \end{cases} \quad (6)$$

and depends on the energy difference between solutions

$$\Delta E = sat(\boldsymbol{S}') - sat(\mathbf{S}), \quad (7)$$

and on a global parameter T, the temperature. If the new solution $\boldsymbol{S}'$ improves the current one $\boldsymbol{S}$, then it is accepted and adopted as the current solution. If it is not better, its acceptance is determined by an exponential function, which becomes smaller as long as the temperature decreases. Hence, less fitted solutions will be more difficult to be accepted at lower temperatures.

Finally, for the cooling schedule we follow the standard geometric approach

$$T_{i+1} = \alpha \cdot T_i, \quad (8)$$

where α is a control parameter indicating how long will we stay at a given temperature.

B. Genetic Algorithm

A genetic algorithm [12] is a bio-inspired search algorithm based on the evolution of collections of individuals (i.e. *populations*) as result of natural selection and natural genetics. Starting from an initial population, their individuals evolve into a new generation by means of selection, crossover and mutation operators. This evolution (i.e. iteration of the algorithm) is performed selecting some individuals according to their quality from the population. Then some parents are chosen and combined using crossover to produce new individuals (children). Finally, all the individuals in the new population have a certain but very small probability of mutation, i.e. their hereditary structure may be altered. Observe that the size of the new population coincides with that of the initial population.

The crossover and mutation operators are in charge of producing new individuals and they are applied with different probabilities i.e. crossover probability and mutation probability. Each one of these operators plays a different role. Crossover should increase the quality of the population. Mutation allows the exploration of different areas of the search space and helps to avoid local optima. An adequate choice of the selection, crossover and mutation operators allows the genetic algorithm can find a near optimal solution in a reasonable number of iterations.

In our genetic algorithm to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, each individual (i.e. solution) is represented and evaluated, as in the case of simulated annealing, by a vector representation and the satisfaction function, $sat(\boldsymbol{S})$, respectively.

Consider a backlog with five requirements $\boldsymbol{R} = \{r_1, r_2, r_3, r_4, r_5\}$ and let $\boldsymbol{E} = \{3, 4, 2, 1, 4\}$ be the set of their associated development efforts. The development effort bound B is set to a value of 7. We have two requirements subsets $\boldsymbol{S} = \{r_1, r_2\}$ and $\boldsymbol{S}' = \{r_3, r_5\}$, that can be represented as vectors 11000 and 00101, respectively. Suppose that these subsets have been selected to crossover and that the crossover point chosen is between the second and the third bit of the vector representation. As result we obtain the offspring vectors 11101 and 00000, which corresponds to not valid subsets $\{r_1, r_2, r_3, r_5\}$ and Ø. Thus, crossover is not a closed operator.

Now, consider the subset $\boldsymbol{S} = \{r_1, r_2\}$ represented by the vector 11000. Suppose that the fourth bit is altered by mutation, obtaining the vector 11010, which corresponds to the subset of requirements $\{r_1, r_2, r_4\}$ with an associated development effort of 8, greater than the fixed effort bound B. Thus, mutation is not a closed operator either.

We see that new individuals do not correspond to valid subsets of requirements, because their associated development efforts exceed the fixed bound B. Thus, mutation and crossover operators are not closed operators. In this situation it makes sense to introduce a repair operator to ensure the closeness of these operators or to define new closed mutation and crossover operators. If we introduce a repair operator, it has to act in the genetic algorithm just after the mutation operator. The repair operator transforms invalid subsets of requirements into valid ones, by randomly eliminating requirements in the subset until its effort is within the fixed effort bound (i.e. the requirement subset effort is less than or equal to B).

The closed mutation operator alters a bit x_i in the vector representation of a subset $\boldsymbol{S}$ of requirements as follows:

- If $x_i = 1$, then it can always be altered by mutation and turned into 0. This corresponds to the deletion of a requirement r_i.
- If $x_i = 0$, then it can be altered by mutation and turned into 1, if and only if the addition of the development effort e_i associated with requirement r_i does not exceed the fixed development effort bound B. That is to say,

$$eff(\boldsymbol{S}) + e_i \leq B. \tag{8}$$

In the case of the crossover operator, it is not always possible to find a crossover point that gives valid offspring. For example, consider again the subsets represented by the vectors 11000 and 00101, for all possible crossover points the crossover operator obtains not valid individuals that represent requirements subsets with an associated development effort greater than B. With this fact in mind and in order to define a closed crossover operator as simple as possible, we decide to adopt the following strategy: use a classical single point crossover operator but returning an offspring formed only by valid individuals. Returning to the example above, if the crossover point chosen is between the third and the fourth bit, the offspring obtained is 11001 and 00100 with efforts 11 and 2, respectively. So, the closed crossover operator returns only one valid individual 00100. Although the complexity of computing the operator increases having to check the validity of individuals obtained in the offspring, this increase is less than having to find the existence of a crossing point for obtaining an offspring in which all the individuals are valid.

The genetic algorithm used to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, starts with the generation of an initial population. The initial population guarantees that each requirement in the backlog $\boldsymbol{R}$ is included in at least one individual while the rest of requirements are randomly selected until development effort bound B is reached. Thus, the population size is $|\boldsymbol{R}|$.

We use satisfaction function, $sat(\boldsymbol{S})$, as fitness function to evaluate the quality of a subsets of requirements.

In order to apply the crossover operator, each individual $\boldsymbol{S}$ is selected to be a parent with a probability, $p_{\boldsymbol{S}}$, proportional to its fitness value in the current population, $\boldsymbol{Q}$, and is defined as

$$p_{\boldsymbol{S}} = sat(\boldsymbol{S}) / \sum_{\boldsymbol{S}' \in \boldsymbol{Q}} sat(\boldsymbol{S}'). \tag{9}$$

Our aim is to choose requirements subsets with the highest satisfaction.

In the offspring production process, two requirements subsets are recombined by means of the closed single point crossover operator described above. The mutation of the offspring requirements subsets is done using the closed mutation operator previously described with a probability of $1 / (10 \cdot |\boldsymbol{R}|)$ near to zero. These offspring individuals are added to the new population together with its parents. Then the new population in the current iteration of the algorithm is reduced to the original size, selecting the $|\boldsymbol{R}|$ best requirements subsets. This selection process eliminates those less fitted individuals from the new population.

Elitism can help preventing the loss of good solutions once they are found. We can include elitism in our genetic algorithm, if before applying the crossover operator, we preserve a few of the best individuals of the actual population in the new one. Then the rest of individuals in the new population are added following the above described process applying crossover, mutation and resizing the new population.

We decide to stop the genetic algorithm after a fixed given number of evaluations (e.g. 10,000 evaluations of the fitness function), then, the best solution found during the last iteration is returned.

C. Ant Colony Optimization

Ant colony optimization is a meta-heuristic for combinatorial optimization problems proposed by Dorigo et al. [6, 5]. This technique emulates the behaviour of real ants in their task to find the shortest path from their colony to a source of food. This searching task is performed by means of a substance called pheromone, used by ants to communicate which each other. Ants leave a pheromone trail on the ground that marks the path found. If other ants find and follow the same path, this pheromone trail will be stronger, attracting other ants to follow it. One characteristic of pheromone is that it evaporates over time, making less desirable those trails less travelled. What at first seems a random behaviour for ants, when no pheromone trail is present on the ground, turns into a movement influenced by the substance left by other ants in the colony.

Ant Colony System (ACS) is one of the ant colony optimization algorithms [6]. In our ACS algorithm to find a subset of requirements $\boldsymbol{S}$ with maximum satisfaction within the effort bound B, the problem itself is encoded as a fully connected directed graph in which each vertex represents a requirement r_i (e.g. Figure 1(a) shows a fully connected directed graph for a set of five requirements). A value of pheromone τ_{ij} is associated to the edge joining requirements r_i and r_j. This value can be read and modified by the ants. Initially the pheromone value is set to a given value τ_0.

At an iteration of the ACS algorithm each ant k in the colony builds a solution $\boldsymbol{S}_k$ by traversing the graph vertex by vertex without visiting any vertex more than once. When ant k is in vertex i, the following vertex j is selected stochastically within the set of visible vertices from i, $vis_k(i)$. We say that for ant k a vertex j is visible from vertex i, if and only if j has not been previously visited and the inclusion of requirement r_j, with development effort e_j, in the solution $\boldsymbol{S}_k$ does not exceed the fixed development effort bound B. That is to say,

$$vis_k(i) = \{ j \mid eff(\boldsymbol{S}_k) + e_j \leq B \}. \tag{10}$$

During the construction process of a solution $\boldsymbol{S}_k$, when ant k is in vertex i, it has to select the following vertex j to visit. This is done applying the *pseudorandom proportional rule* [5]. Thus, based on a random variable q uniformly distributed on [0, 1] and a parameter q_0, if $q \leq q_0$, then ant k traverses from vertex i to a vertex j given by

$$j = \arg max_{l \in vis_k(i)} \{ \tau_{il} \cdot \eta_{il}^{\beta} \}, \tag{11}$$

where β is a parameter controlling the relative importance of the heuristic information η_{ij}, which is given by

$$\eta_{ij} = \mu \cdot (s_j / e_j), \tag{12}$$

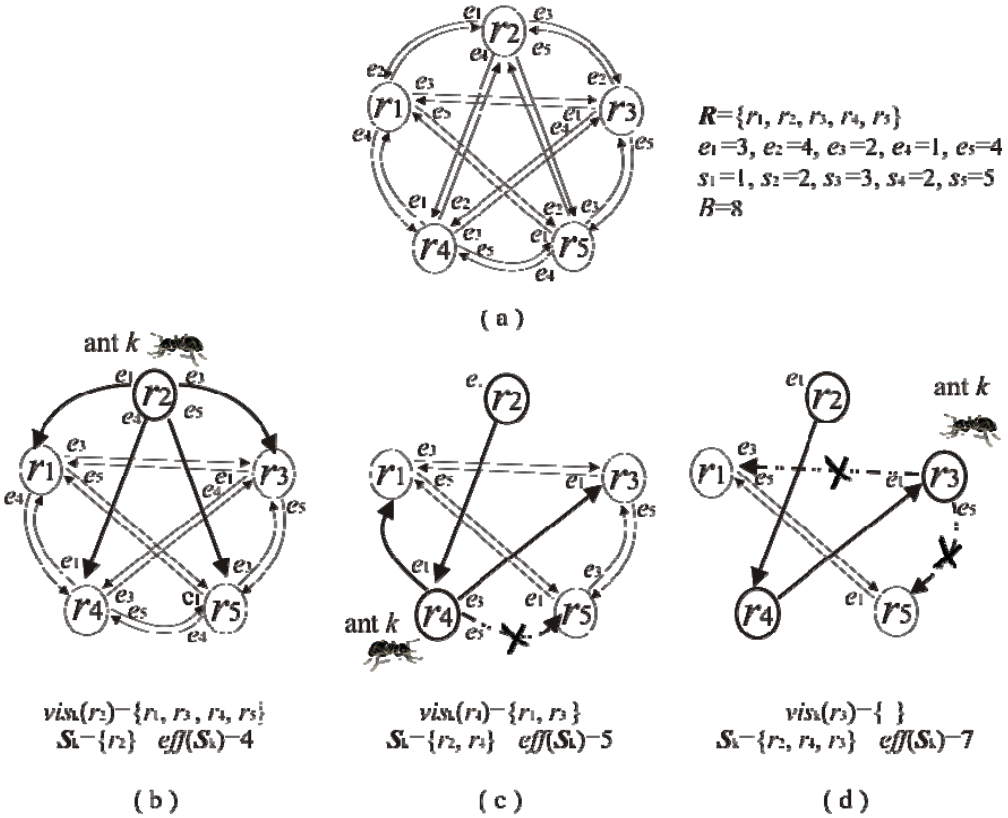


Figure 1. Example of ant k iteration.

where μ is normalization constant; η_{ij} represents a productivity measure for the software development of a requirement. Otherwise, if $q < q_0$, ant k chooses vertex j with a probability given by:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{i \in vis_k(i)} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} & if\ j \in vis_k(i) \\ 0 & otherwise \end{cases} \tag{13}$$

where the parameters α and β control the relative importance of the pheromone and heuristic information. An ant in the colony will have a probability q_0 of exploiting the experience accumulated by the colony (i.e. pheromone values) and a probability (1-q_0) of exploring new paths following the probability distribution p_{ij}^{k}.

The pheromone trail is updated both locally and globally. Global updating tries to highlight arcs belonging to paths that correspond with high satisfaction solutions. Once all ants in the colony have constructed their solutions, the best ant updates the pheromone values of visited arcs (i.e. those arcs used by the ant to create its path). The amount of pheromone left in each arc is

$$\Delta\tau_{ij} = \frac{sat(S_{best})}{sat(R)} \tag{14}$$

where $sat(\boldsymbol{S}_{\text{best}})$ is the satisfaction of the subset of requirements of the best ant and $sat(\boldsymbol{R})$ is the satisfaction of the backlog. In this way, the greater the satisfaction, the greater the amount of pheromone left. The global pheromone updating equation is

$$\tau_{ij} = (1 - \rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij}, \tag{15}$$

where ρ is the pheromone evaporation rate. This equation is only applied to those arcs included in the best solution.

Local pheromone updating emulates the pheromone evaporation process observed in the trails followed by real world ants. During an iteration, each ant k updates pheromone locally as it traverses arcs. If (i, j) is the last arc traversed by ant k, then the pheromone is updated locally as follows

$$\tau_{ij} = (1 - \varphi)\cdot\tau_{ij} + \varphi\cdot\tau_0, \tag{16}$$

where $\varphi \in (0,1]$ is a pheromone decay coefficient and τ_0 is the initial pheromone value. In our problem this value is defined as

$$\tau_0 = 1/sat(\boldsymbol{R}). \quad (17)$$

The goal of local pheromone updating is to prevent ants find the same solution during one iteration. Ants diversify their search because decreases the amount of pheromone of traversed arcs making these arcs less desirable. And thus, the solution returned by the ant colony is the best solution found since the start of the algorithm.

For example, Figure 1(a) shows a fully connected directed graph for a backlog with five requirements $\boldsymbol{R} = \{r_1, r_2, r_3, r_4, r_5\}$, with efforts e_1=3, e_2=4, e_3=2, e_4=1, e_5=4, and satisfactions s_1=1, s_2=2, s_3=3, s_4=2, s_5=5, respectively. The development effort bound B is set to a value of 8. Figures 1(b) to 1(d) depicts the steps follow by an ant during an iteration. Initially, (see Figure 1(b)) the ant chooses randomly requirement r_2 as starting vertex, so $\boldsymbol{S} = \{r_2\}$ and arcs reaching to r_2 have been deleted because they could never been used. From r_2 the set of visible vertices is $vis(r_2) = \{r_1, r_3, r_4, r_5\}$. If the ant only uses heuristic information in order to build its solution $\boldsymbol{S}$, then it will chose r_4 as the vertex to travel to, because it has the highest η_{ij} value. Therefore, ant adds r_4 to its solution, $\boldsymbol{S} = \{r_2, r_4\}$, and searches for a new requirement to add from this vertex as it is depicted in Figure 1(c). In this situation, the ant's visible set is $vis(r_4) = \{r_1, r_3\}$ and using only heuristic information the next vertex to travel to is r_3. Finally, Figure 1(d) shows that once the ant has added r_3 to its solution, $\boldsymbol{S} = \{r_2, r_4, r_3\}$, it has to stop because there are not any other visible vertices.

IV. Case Study

This section presents the benchmark problem we have used to test the three different search algorithms applied for solving the next release problem: simulated annealing, genetic algorithms and ant colony optimization. It also describes the methodology we have followed, the configuration of the different algorithms and the results obtained.

A. Test Problem

The three algorithms were applied to a real software project problem taken from Greer and Ruhe [11]. The problem is a software project that consists of 20 requirements, 5 customers. The software development effort boundary is set to 25. Each requirement has an associated development effort showed in Table I. Also, each customer assigns a priority to each requirement in the 1 to 5 range. The matrix with these importance values is shown in Table II. Finally, the weights, assigned to customers, measure their importance from the point of view of the project manager. These weights were defined based on a pair-wised comparison of customers. For our experiments, we have normalized the original customers' weights in the 1 to 5 range and they are showed in Table III.

TABLE I. Requirement's Development Effort

	Requirements' development effort									
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
Effort	1	4	2	3	4	7	10	2	1	3

	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Effort	2	5	8	2	1	4	10	4	8	4

TABLE II. Customer Assignment of the Priority Level of Each Requirement

	Requirements' priority level									
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
c_1	4	2	1	2	5	5	2	4	4	4
c_2	4	4	2	2	4	5	1	4	4	5
c_3	5	3	3	3	4	5	2	4	4	4
c_4	4	5	2	3	3	4	2	4	2	3
c_5	5	4	2	4	5	4	2	4	5	2

	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
c_1	2	3	4	2	4	4	4	1	3	2
c_2	2	3	2	4	4	2	3	2	3	1
c_3	2	4	1	5	4	1	2	3	3	2
c_4	5	2	3	2	4	3	5	4	3	2
c_5	4	5	3	4	4	1	1	2	4	1

TABLE III. Customers' Weights

	Customers' weights				
	c_1	c_2	c_3	c_4	c_5
Weight	4	4	3	5	5

B. Methodology

We have tested each algorithm, after setting its configuration of parameters, performing 100 independent runs for the test problem. We compute, for the solutions obtained, their mean of satisfaction, effort and time as measures of tendency, and their standard deviation, maximum and minimum as measures of dispersion.

C. Simulated Annealing Results

Initially, following the settings applied by [1], we set the initial temperature to 100 and the cooling control parameter α to 0.9995 in our simulated annealing algorithm. Then, to get a faster cooling scheduling, we decrease the cooling control parameter setting α to 0.995 and 0.95, respectively. The best results (see Table IV) were obtained with the initial configuration of parameters (i.e. T=100 and α=0.9995). We repeat the experiments increasing the initial temperature to 1000 and setting the cooling control parameter α to 0.99995. In this case, we also decrease the cooling control parameter α (i.e. to 0.9995 and 0.995) to make the algorithm faster.

Table IV shows the experimental results obtained by the simulated annealing algorithm. The best results were obtained setting the initial temperature and cooling control parameter, (T, α), to (100, 0.9995) and (1000, 0.99995), respectively. The higher the value of these parameters, the longer execution time of the algorithm, without a very significant improvement in the results. However, the decrease in the value of the cooling control parameter

resulted in a worsening of the results obtained by the algorithm as its spread was increased. Figure 2 depicts a visual comparison of the maximum, average and minimum satisfaction values and of the mean development effort of the solutions found by the different parameters configurations tested for the simulated annealing algorithm.

TABLE IV. SIMULATED ANNEALING RESULTS

Param.	Satisfaction		Effort	Time (ms)
	Min-Max	*Average*		
T=100 α=0.9995	**698–815**	**777.9±29.7**	**24.9±0.3**	**21.6±2.9**
T=100 α=0.995	614–815	743.8±46.0	24.7±0.5	2.09±0.9
T=100 α=0.95	434–815	692.4±61.4	24.8±0.4	0.13±0.6
T=1000 α=0.99995	**705–815**	**797.2±25.9**	**24.9±0.2**	**670.3±20.9**
T=1000 α=0.9995	709–815	782.3±33.4	24.8±0.4	72.5±3.9
T=1000 α=0.995	600–815	738.8±47.9	24.7±0.4	7.55±2.1

D. Genetic Algorithm Results

The genetic algorithm was tested using a population size of 20 (i.e. the number of requirements in the NRP), a crossover probability of 0.8 and 0.9, respectively, for each of the four possible configurations of closed/not closed operators together with/without elitism. Each execution of the algorithm lasts 5000 iterations, so it performs 100000 evaluations of the fitness function.

Table V shows the results obtained by the genetic algorithm. If we do not preserve some of the best individuals from the actual to the new population (i.e. we do not use elitism), the proposed closed operators perform better (obtain better solutions), than the option of using not closed operators and a repair operator to obtain valid individuals (i.e. valid solutions from the point of view of the problem). But in both cases, there is a wide range of customer satisfaction values. Elitism prevents this high variability in the results and reduces it simply by preserving some best individuals between consecutive iterations (e.g. in our test we have retained the 20% or 10% of the best individuals). The best results were obtained for the combination of not closed operators and elitism without significant differences with respect to the value that we used for the probability of crossing (i.e. 0.8 or 0.9). Closed operators do not work as well as the cooperation of not closed and repair operators because they are more likely to get trapped in areas of the search space where local maxima exits. Remember that closed operators only return valid individuals, instead of producing invalid individuals that are then repaired randomly. The behavior of the combination of not closed operators and random repair allows the genetic algorithm to escape from local maxima and exploring other areas of the search space.

TABLE V. GENETIC ALGORITHM RESULTS

Parameters	Satisfaction		Effort	Time (ms)
p_c, *Closed, Elitist*	*Min Max*	*Average*		
0.8, No, No	486-815	636.8±62.5	23.4±1.7	761.2±57.0
0.8, Yes, No	581-815	716.4±50.3	24.3±1.0	626.6±59.8
0.8, No, Yes	**785-815**	**812.6±8.2**	**25.0±0.0**	**898.3±63.0**
0.8, Yes, Yes	742-815	797.3±18.5	25.0±0.1	751.3±70.8
0.9 , No, No	382-778	642.9±68.9	23.3±1.8	763.7±68.9
0.9, Yes, No	549-815	715.1± 48.3	24.2±1.0	623.7± 58.5
0.9, No, Yes	**785-815**	**810.8±10.5**	**25.0±0.0**	**953.6±73.4**
0.9, Yes, Yes	742-815	806.3±15.9	25.0±0.0	782.9±74.4

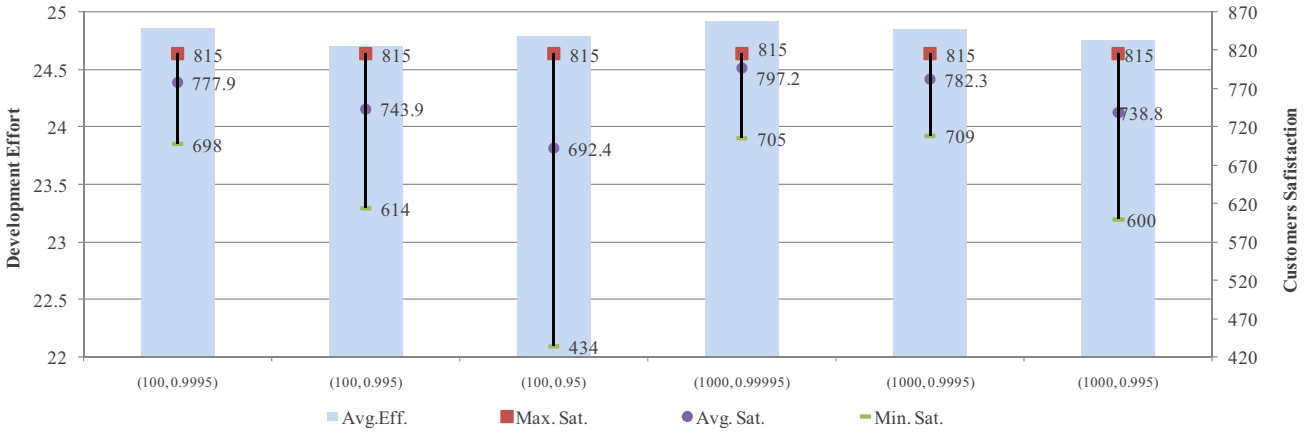


Figure 2. Simulated Anneling Comparison of Results.

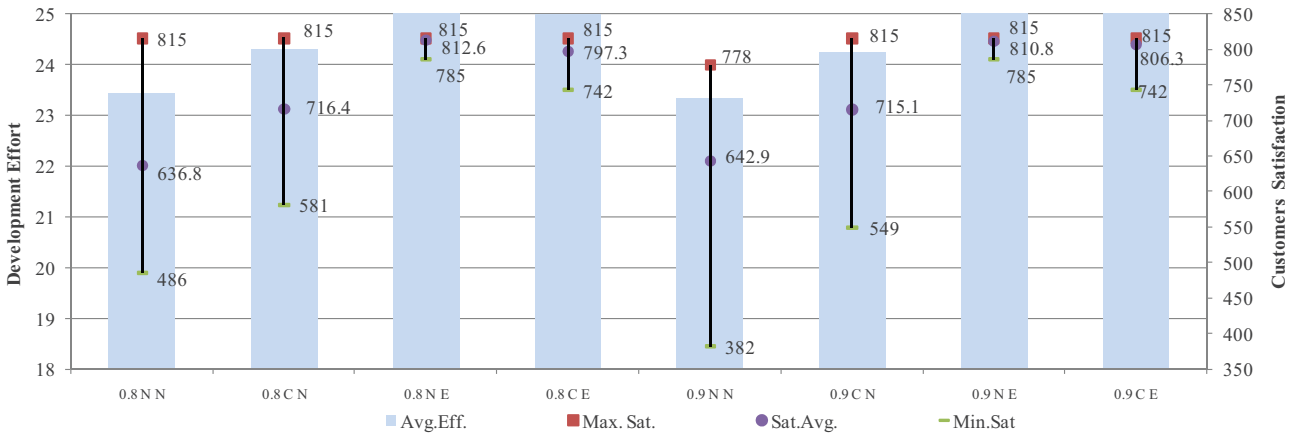


Figure 3. Genetic Algorithm Comparison of Results.

Figure 3 depicts a visual comparison of the maximum, average and minimum satisfaction values and of the mean development effort of the solutions found by the different parameters configurations tested for the genetic algorithm.

E. Ant Colony System Results

In order to test our ant colony system we considered a colony with 10 ants (i.e. the half of the number of requirements in the problem). The colony searched 100 times for a solution and returned the best solution found. We maintain fixed the pheromone evaporation rate ρ=0.1 (we also set the pheromone decay coefficient φ to 0.1) and the parameter q_0 = 0.95 controlling the relative importance of exploitation versus exploration in all executions of the algorithm. Nonetheless, we use different values for the parameters α (values 0 or 1) and β (values 0, 1, 2 or 5) controlling the relative importance of the pheromone and heuristic information. For all the combinations of parameters tested and in all executions, the ant colony system found the same best solution, the set of requirements $\boldsymbol{S}_{best} = \{r_1, r_2, r_3, r_4, r_5, r_8, r_9, r_{10}, r_{11}, r_{14}, r_{15}\}$ with a satisfaction of 815 and development effort of 25. Due to this reason, we have compared the performance of ant colony system for the different parameters combinations in terms of execution time.

Figure 4 shows the ant colony system execution times. It depicts the maximum, minimum and average execution time and it can be seen that the increase in the relative importance of heuristic information (i.e. β parameter) turns into an increasing of the execution time.

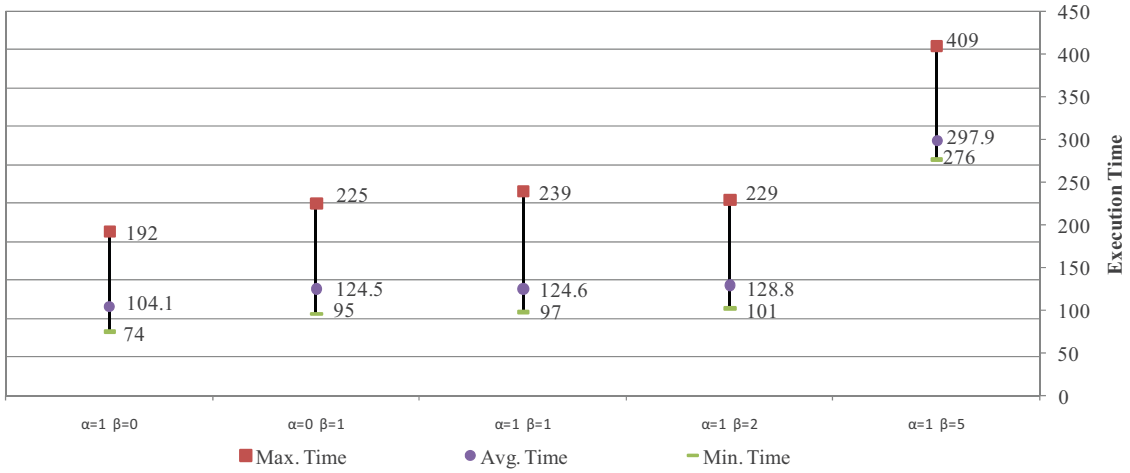


Figure 4. Ant Colony System execution times.

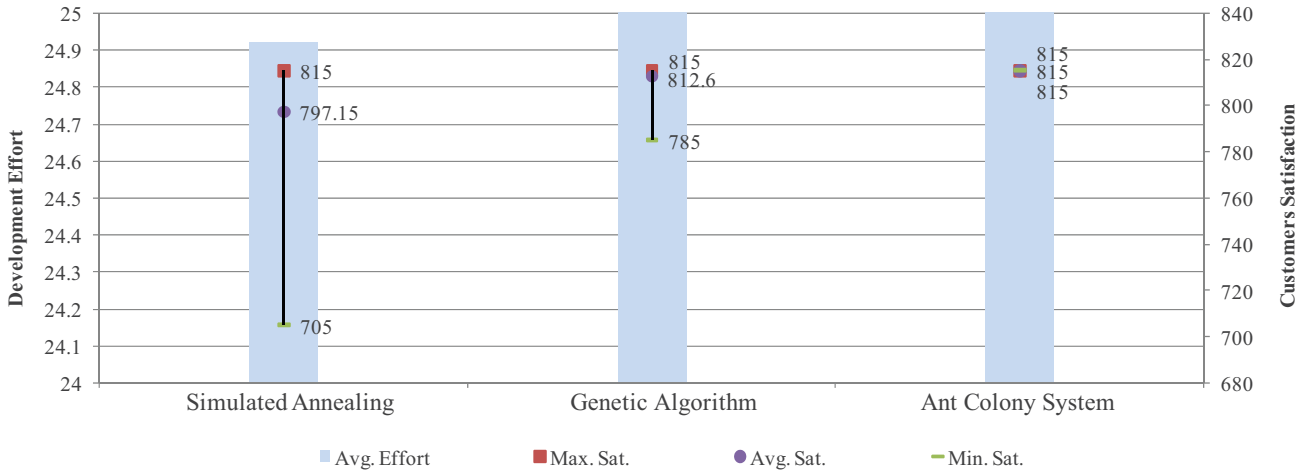


Figure 5. Comparison of the results of the three algorithms.

F. *Comparison of Results*

In this subsection we compare the obtained results by the three evaluated algorithms using the best parameters configurations. Thus, in the case of the simulated annealing algorithm we have selected an initial temperature T=1000 and a cooling control parameter α=0.99995, configuration that has shown to produce the best results in terms of satisfaction and development effort (see Table IV).

The parameters selected for the genetic algorithm are a crossover probability of 0.8, not closed crossover and mutation operators, and elitism. This combination returns the best results as it can be seen on Table V.

Finally, for the ant colony system we have selected a value of 1 for the parameters α and β, because in this way pheromone and heuristic information have the same importance. Also, this configuration provides an intermediate execution time, that it is neither the lowest nor the highest.

Once we have selected the parameters for the different algorithms tested, the results obtained are compared in Figure 5. All of them find the same best solution. Simulated annealing provides the worst results, presenting greater dispersion in terms of the satisfaction of the customers and waste development effort (it does not reach the limit of effort *B* fixed in the problem). The genetic algorithm corrects these deficiencies, using all the available development effort and reducing the dispersion of customer satisfaction. Ant colony system enhances customer satisfaction as it always finds the same solution. This is due to the fact that the ant colony system returns the best solution found since the start of the algorithm, this is the reason why there is no dispersion of customer satisfaction.

V. CONCLUSSIONS

The requirement selection problem is a known issue in search based software engineering. Previous works have successfully afforded it applying hill climbing, simulated annealing and genetic algorithms techniques. In this paper we have studied the genetic algorithm from the point of view of the NRP problem, paying special attention to the characteristics of the crossover and mutation operators used. Thus, we have shown that classical single-point crossover and mutation operators are not closed with respect to NRP (i.e. the individuals returned are solutions that can violate the restrictions of the problem to be solved). This fact lead as to apply a repair operator to the solutions obtained, so that they verify the conditions of the problem. As an alternative to these operators we have defined closed crossover and mutation operators. However, not-closed operators together with the repair operator allow the search to be extended over more areas of the search space. Another important aspect of the genetic algorithm is that the use of elitism prevents from the high variability in the range of customers' satisfaction values of the solutions found. In our tests the combination of not-closed operators and elitism has returned the best results.

We have extended the set of techniques applicable to NRP with ant colony systems. The adaptation of ant colony system to NRP has been made following the next steps: i) find an appropriate representation of the problem (i.e. a fully connected directed graph), ii) define the concept of visibility throughout the process that carries out an ant to construct a solution, iii) use, as heuristic information, a measure of productivity for the software development of a requirement and, iv) adapt to the problem the mechanism for updating pheromone (i.e. the greater the satisfaction, the greater the amount of pheromone left). A characteristic of our ant colony system is that it returns the best solution found from all the iterations performed by the algorithm. This behavior has a direct impact on the variability in the range of customer satisfaction values of the solutions found: in all cases the ant colony system returns the same best solution.

We have evaluated simulated annealing, genetic algorithm and ant colony system and compared them on a case study. The comparison has been done studying the average and standard deviation of development effort and

customers' satisfaction, as well as the minimum and maximum customers satisfaction, of the solutions obtained. Ant colony system obtains the solutions with maximum customers' satisfaction in the limit of effort fixed in the NRP problem. Genetic algorithm shows a similar performance but obtains solutions with a greater variability in customers' satisfaction. Finally, simulated annealing obtains the worst results in terms of the development effort bound and customers' satisfaction.

Concerning to the best solution of the problem that is found by the different algorithms tested, we have observed that it is composed of a high percentage of high-satisfaction and low-effort requirements. A characteristic of this solution is that includes a high percentage of the requirements considered as most important by each individual customer.

The software engineer is often confronted with the problem of selection of requirements within a software development project when applying agile methodologies when he tries to reach an agreement with customers at the start of each iteration. The three techniques studied can help at the time of resolving the conflicts that appear in this process. We believe that the usefulness of these techniques lies in its inclusion in tools to assist software engineering processes. We plan to afford this challenge in the future, including the functionality provided by these techniques in a requirement management tool.

Further work will involve applying meta-heuristic techniques to other reformulations of the next release problem considering a set of objectives, instead of a unique one, and constraints, such as dependency relationships between requirements.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Education and Science under project TIN2007-67418-C03-02 and by the Junta of Andalucía under project P06-TIC-02411-02.

REFERENCES

[1] A. Bagnall, V. Rayward-Smith, and I. Whittley. "The next release problem," Information and Software Technology, 43(14):883–890, Dec. 2001

[2] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis, 2006. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *Proceedings of the 22nd IEEE international Conference on Software Maintenance* (September 24 - 27, 2006). ICSM. IEEE Computer Society, Washington, DC, 176-185.

[3] K. Beck et al. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, "Manifesto for Agile Software Development", 2001, http://www.agilemanifesto.org/.

[4] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin,B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," IEE Proceedings - Software, vol. 150 (3), pp. 161–175, 2003.

[5] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, 2006, pp. 28-39.

[6] M. Dorigo and L.M. Gambardella, "Ant colonies for the traveling salesman problem," *BioSystems*, 43, 1997, pp. 73-81.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, 1996, pp. 29-41.

[8] JJ. Durillo, Y. Zhang, E. Alba, A.J.Nebro. A Study of the Multi-Objective Next Release Problem. *Proceeding of 1st International Symposium on Search Based Software Engineering*, Cumberland Lodge, UK, IEEE Computer Society: Los Alamitos, CA, 2009; 49-58. DOI: 10.1109/SSBSE.2009.21

[9] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, y Y. Zhang, ""Fairness Analysis" in Requirements Assignments," *In Proceedings of the 16th IEEE International Requirements Engineering* Conference. IEEE Computer Society Washintong, DC, pp.115-124, September 2008.

[10] A. Finkelstein, M. Harman, S. Mansouri, J. Ren, y Y. Zhang, "A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making," Requirements Engineering, 14 (4): 231-245. 2009

[11] D. Greer and G. Ruhe. "Software Release Planning: An Evolutionary and Iterative Approach," Information and Software Technology, vol. 46, pp. 243-253, 2004.

[12] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., 1989.

[13] M. Harman, "The Current State and Future of Search Based Software Engineering," International Conference on Software Engineering, 2007.

[14] M. Harman and B. F. Jones, "Search-based software engineering," Information and Software Technology, vol. 43, no. 14, pp. 833–839, December 2001.

[15] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," IEEE Software, pp. 67-74, September/October 1997.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

[17] L. Rising and N. S. Janoff, "The scrum software development process for small teams," Software, IEEE, vol. 17, no. 4, pp. 26–32, August 2002.

[18] J. del Sagrado and I.M. del Águila "Ant Colony Optimization for requirement selection in incremental software development", 1st International Symposioum of Search Based Software Engineering, SSBSE'09, Cumberland Lodge, UK, 2009, fast abstracts, www.ssbse.org/2009/fa/ssbse2009_submission_30.pdf

[19] M.O. Saliu y G. Ruhe, "Bi-objective release planning for evolving software systems," ESEC-FSE '07: In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ACM, New York, NY, USA, pp.105–114. 2007

[20] P. Schuh, *Integrating Agile Development in the Real World (Programming Series)*, Charles River Media, Inc., 2004.

[21] K. Schwaber, "Scrum development process,", Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 1995), Austin, Texas, USA, pp. 117-134, 1995.

[22] K. Schwaber y M. Beedle, *Agile software development with Scrum*, Pearson Education International, 2008.

[23] Y. Zhang, M. Harman, y S.A. Mansouri, "The Multi-Objective Next Release Problem," Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation, ACM, pp. 1129-1137,2007

Search-based prediction of fault-slip-through in large software projects

Wasif Afzal, Richard Torkar and Robert Feldt
Blekinge Institute of Technology, Ronneby, Sweden
Email: waf|rto|rfd@bth.se

Greger Wikstrand
KnowIT YAHM Sweden AB
Email: greger.wikstrand@yahm.se

***Abstract*—A large percentage of the cost of rework can be avoided by finding more faults earlier in a software testing process. Therefore, determination of which software testing phases to focus improvements work on has considerable industrial interest. This paper evaluates the use of five different techniques, namely particle swarm optimization based artificial neural networks (PSO-ANN), artificial immune recognition systems (AIRS), gene expression programming (GEP), genetic programming (GP) and multiple regression (MR), for predicting the number of faults slipping through unit, function, integration and system testing phases. The objective is to quantify improvement potential in different testing phases by striving towards finding the right faults in the right phase. We have conducted an empirical study of two large projects from a telecommunication company developing mobile platforms and wireless semiconductors. The results are compared using simple residuals, goodness of fit and absolute relative error measures. They indicate that the four search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting the fault-slip-through for each of the four testing phases. At the unit and function testing phases, AIRS and PSO-ANN performed better while GP performed better at integration and system testing phases. The study concludes that a variety of search-based techniques are applicable for predicting the improvement potential in different testing phases with GP showing more consistent performance across two of the four test phases.**

I. Introduction

Software quality is most commonly referred to as conformance to both functional and non-functional requirements. Presence of software faults is usually taken to be an important factor in software quality, a factor that shows an absence of quality. There have been different types of software quality evaluation models proposed in software engineering literature, all of them with the objective of accurately quantifying software quality (see [1] for a classification of these models).

We know that faults are cheaper to find and remove earlier in the software development process [2]. Software testing is the major fault-finding activity therefore much research has focused on making the software testing process as efficient as possible. One way to improve testing process efficiency is to avoid unnecessary rework by finding more faults earlier. The Faults-Slip-Through (FST) concept [3], [4] is one way of providing quantified decision support to reduce the effort spent on rework. The Faults-Slip-Through (FST) concept is used for determining whether or not a fault slipped through the phase where it should have been found. The term phase refers to any phase in a typical software development life cycle. However the most interesting and industry-supported applications of FST measurement are during testing. This is because a defined testing strategy within any organization implicitly classifies faults whereby certain types of faults might be targeted by certain strategies. FST is essentially a fault classification approach and focuses on when it is cost-effective to find each fault. Depending on the FST numbers for each testing phase (e.g. unit, function, integration and system), improvement potentials can be determined by calculating the difference between the cost of faults in relation to what the fault cost would have been if none of them would have slipped through the phase where they were supposed to be found. Thus, FST is a way to provide quantified decision support to reduce the effort spent on rework. This reduction in effort is due to finding faults earlier in a cost-effective way.

Found During: / Expected fault identification phase:	Review	Unit Test	Function Test	Integration Test	System Test	Acceptance Test	Customer Identified	Total
Review	15	25	86	25	30	2	1	184
Unit Test		19	56	15	19	1	0	110
Function Test			33	4	4	0	0	41
Integration Test				8	11	0	0	19
System Test					4	0	1	5
Acceptence Test						1	0	1

Figure 1. An example FST matrix.

One way to visualize FST for different software testing phases is using an FST matrix (Figure 1). The columns in Figure 1 represent the phases in which the faults were found (*Found During*), whereas the rows represent the phases where the faults should have been found (*Expected fault identification phase*). For example 56 of the faults that were found in function testing should have been found during unit testing.

Considering the initial successful results of implementing FST across different organizations within Ericsson [3], [4], it is interesting to investigate how to use FST measurement to provide additional decision support for project management. Staron and Meding [5] highlight that the prediction of FST can be a refinement to their proposed approach for predicting the number of defects in the defect database. Similarly

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.19

Damm [3] highlight that FST can potentially be used to support software fault predictions. This additional decision support would be introduced in order to make the software development more *predictable*, i.e. we can better cope with the conflicting demands of doing more with less, doing it faster and doing it with higher quality [6].

Therefore with the goal of providing decision support based on FST, we set out to investigate the following research question:

RQ: How can we predict FST for each testing phase multiple weeks in advance by making use of data about project progress, testing progress and fault inflow from multiple projects?

We are particularly interested in evaluating the accuracy of predictions using search-based techniques [7]. The motivation for applying these techniques is that these are non-linear, data-driven and self-adaptive approaches, having the ability to model potentially complex relationships between input and output data. Traditional statistical methods like the autoregressive moving average have difficulties in modeling non-linearities and unknown explicit relationships among variables. Search-based techniques, on the other hand, are independent of satisfying any data assumptions and do not require the definition of the functional form of the relationship upfront. The above properties make these techniques general and flexible for data-driven modeling tasks. We have added multiple regression as another technique to better compare the prediction ability of search-based techniques.

Secondly, we use metrics related to the status of various work packages, testing progress and fault-inflow at the project level as our independent variables. We present the results of evaluating the use of multiple regression (MR) and four search-based techniques (artificial immune recognition system (AIRS), gene expression programming (GEP), genetic programming (GP) and particle swarm optimization based artificial neural network (PSO-ANN)) to predict FST in each of the four testing phases (unit, function, integration, system) of a large-scale project carried out by a company developing mobile platforms and wireless semiconductors. The predictions are compared for differences in models' residuals, goodness of fit and absolute relative error values. The comparative results indicate that while MR is not able to perform as well as the search-based techniques, GP performs better for two testing phases (integration and system) and AIRS, PSO-ANN perform better for unit and function testing phases respectively.

The remainder of the paper is organized as follows. Section II summarizes related work. Section III describes the research context and Section IV discusses the variables, feature selection and evaluation measures used. The search-based techniques used in this paper are discussed in Section V. Results are presented in Section VI while Section VII contains the discussion. Validity evaluation and conclusions make up Sections VIII and IX, respectively.

II. Related work

There are a number of modeling mechanisms to predict the quality of software. Due to the definition of software quality in many different ways, previous studies have focused on predicting different but related dependent variables of interest; examples include predicting for defect density [8], software defect content estimation [9], fault-proneness [10] and software reliability prediction in terms of time-to-failure [11]. In addition, several independent variables have been used to predict the above dependent variables of interest [1]; examples include prediction using size and complexity metrics [12] and prediction using testing metrics [13]. The actual prediction is performed using a variety of approaches, and can broadly be classified into statistical regression techniques, machine learning approaches and mixed algorithms [14]. Increasingly, evolutionary and bio–inspired approaches are being used for software quality classification [15], [16].

This study is different from the above software quality evaluation studies. First, the dependent variable of interest here is the number of faults slipping through to various testing phases with the aim of triggering corrective actions for avoiding unnecessary rework late in software testing. Second, we make use of several independent variables at the *project* level, i.e. variables depicting work status, testing progress status and fault-inflow. A study by Staron and Meding [5] makes use of project planning and testing status variables for predicting weekly fault inflow. The current study, although taking inspiration from this study, is different in terms of purpose, use of independent and dependent variables and prediction techniques employed.

III. Research context

The data used in this study comes from two large projects at a telecommunication company that develops mobile platforms and wireless semiconductors. The projects are aimed at developing platforms introducing new radio access technologies written using C. The average number of persons involved in these projects is approximately 250. The data from one of the projects is used as a baseline to train the models while the data from the second project is used to evaluate the models' results. We have data from 45 weeks of the baseline project to train the models while we evaluate the results on data from 15 weeks of an on-going project.

The management of these projects is based on the concepts of tollgates, milestones, steering points and checkpoints to manage and control project deliverables. Tollgates represent long-term business decisions while milestones are predefined events at the operating work level. The monitoring of these milestones is an important element of the project management model. Steering points are defined to coordinate multiple parallel platform projects, e.g. handling priorities between different platform projects. The checkpoints are defined in the development process to define

the work status in a process. At the operative work level, software development is structured around work packages (WPs). These work packages are defined during the project planning phase. The work packages are defined to implement change requests or a subset of a use-case, thus the definition of work packages is driven by the functionality to be developed. An essential feature of work packages is that these allow for simultaneous work on different modules of the project at the same time by multiple teams. Since different modules might get affected by developing a single work package, it is difficult to obtain consistent metrics at the module level.

The structure of a project into work packages present an obvious choice for the prediction models since the metrics at work package level have stable values and hold a more direct and intuitive meaning for the company employees. At our subject company the work status of various work packages is grouped using a graphical integration plan (GIP) document. The GIP maps the work packages' status over multiple time-lines that might indicate different phases of software testing or overall project progress. There are different status rankings of the work packages such as number of work packages planned to be delivered for system integration testing.

IV. Research method

Our research method is to apply five different techniques to the task of predicting FST in four testing phases based on input data collected from a telecommunication company developing mobile platforms and wireless semiconductors. The input data is made up of different variables divided into 4 sets (Table I), i.e., fault in-flow, status rankings of work packages, faults-slip-through and test case progress.

During the project life cycle there are certain status rankings related to the work packages (shown under the category of 'status rankings of WPs' in Table I) that influence fault-inflow, i.e. the number of faults found in the consecutive project weeks. The information on these status rankings is also conveniently extracted from the GIP which is a general planning document at the company. Another important set of variables for our prediction models is the actual test case (TC) progress data, shown under the category of 'TC progress' in Table I, which have a more direct influence on the fault in-flow. The information on number of test cases planned and executed at different testing phases is readily available from an automated report generation tool that uses data from an internally developed system for fault logging. These variables along with the status rankings of the work packages influences the fault-inflow; so we monitor the fault-inflow as another variable for our prediction models. Another set of variables representing the output is the number of faults that slipped-through to the unit, function, integration and system testing phases, indicated under the category 'FST' in Table I. We also recorded the accumulated number of faults slipping through to all the testing phases. All of the above measurements were collected at the subject company on a weekly basis.

Table I
Variables of interest for the prediction models.

No.	*Description*	*Abbreviation*	*Category*
1	Fault in-flow	F. in-flow	Fault-inflow
2	No. of work packages planned for system integration	No. WP. PL. SI	Status rankings of WPs
3	No. of work packages delivered to system integration	No. WP. DEL. SI	
4	No. of work packages tested by system integration	No. WP Tested. SI	
5	No. of faults slipping through to all of the testing phases	No. FST	FST
6	No. of faults slipping through to the unit testing	FST-Unit	
7	No. of faults slipping through to the function testing	FST-Func	
8	No. of faults slipping through to the integration testing	FST-Integ	
9	No. of faults slipping through to the system testing	FST-Sys	
10	No. of system test cases planned	No. System. TCs. PL	TC progress
11	No. of system test cases executed	No. System. TCs. Exec.	
12	No. of interoperability test cases planned	No. IOT TCs. PL	
13	No. of interoperability test cases executed	No. IOT TCs. Exec.	
14	No. of network signaling test cases planned	No. NS TCs. PL	
15	No. of network signaling test cases executed	No. NS TCs. Exec.	

We analyzed the dependencies among variables using scatter plots to identify correlated variables, i.e. variables that potentially measure the same attribute and thus using only one of them would be enough. We were especially interested in visualizing the relationship between the measures of status rankings of work packages, test case progress, fault in-flow and rest of the measures related to status rankings of work packages and test case progress. The pair-wise scatter plots of the above attributes showed a tendency of non-linear relationships. Hence, we calculated the Spearman rank-order correlation coefficient to find if there were any strong indications of correlation among the variables. The Spearman rank-order correlation coefficient values varied between 0.2 and 0.6, indicating that the pairs of variables were either weakly or moderately correlated.

We then used the statistical analysis technique of kernel principal component analysis (KPCA) [17] to reduce the number of independent variables to a smaller set that would still capture the original information in terms of explained variance in the data set. The role of original variables in determining the new factors (principal components) is determined by loading factors. Variables with high loadings contribute more in explaining the variance. The results of applying the Gaussian kernel, KPCA (Table II) showed that the first four components explained 97% of the variability in the data set. We did not include the faults-slip-through measures in the KPCA since these are the attributes we are interested in predicting. In each of the four components, all the variables contributed with different loadings, with the exception of two, namely *number of network signaling test cases planned* and *number of network signaling test cases executed*. Hence, we excluded these two variables and used the rest for predicting the fault-slip-through in different testing phases.

The predictive accuracy of different techniques is compared using absolute residuals (i.e. |actual-predicted|) [18],

Table II
THE LOADINGS AND EXPLAINED VARIANCE FROM FOUR PRINCIPAL COMPONENTS.

	Variance explained	Variable loadings. The variable names use abbreviations given in Table I										
		F. inflow	No. WP. PL. SI	No. WP. DEL. SI	No. WP Tested. SI	No. FST	No. System. TCs. PL	No. System. TCs. Exec.	No. IOT TCs. PL	No. IOT TCs. Exec.	No. NS TCs. PL	No. NS TCs. Exec.
Component 1	51.61%	0.60	0.02	0.01	0.09	0.38	0.61	0.34	0.02	0.02	0	0
Component 2	31.07%	0.75	-0.02	0.01	0.13	-0.01	-0.57	-0.32	0.03	0.03	0	0
Component 3	9.88%	-0.29	0.01	0	0.52	0.75	-0.20	-0.11	0.11	0.12	0	0
Component 4	4.64%	0	0	0.02	0.83	-0.50	0.19	0	-0.07	-0.07	0	0

[19] and absolute relative error metric [20]. The goodness of fit of the results from different techniques is assessed using the Kolmogorov-Smirnov (K-S) test. For the K-S test we use $\alpha = 0.05$ and if the K-S statistic J is greater or equal than the critical value J_α, we infer that the two samples did not have the same probability distribution and hence do not represent significant goodness of fit.

V. TECHNIQUES USED

This section discusses the five techniques used in this study for FST prediction.

A. Particle swarm optimization based artificial neural network (PSO-ANN)

The development of artificial neural networks is inspired by the interconnections of biological neurons [21]. Selecting proper ANN architecture parameters is one of the important decisions for designing the ANN model. Different training schemes for ANN have been proposed, the most used being the back-propagation algorithms [22]. However, back-propagation suffers from certain drawbacks [23] e.g. slow convergence, local minima and lack of robustness. Recently, evolutionary optimization methods have been used for neural network training since the search space of a multilayer feed forward neural network with a non-linear activation function is complex and believed to have many local and global minima [24].

Kennedy and Eberhart introduced PSO in 1995 [25] inspired by the coordinated search for food by a swarm of birds, that can be used for parameter estimation of an ANN. The idea behind PSO is that a swarm of particles move through a multidimensional search space for finding a global optimum (e.g. minimum or maximum of a given objective function). Several improvements of the basic PSO algorithm have been proposed, one of them by Trelea [26]. According to [26], the expression for position and velocity update for particle i of the dimension d is given by:

$$v_{id}(t) = a * v_{id}(t-1) + b * (p_{id} - x_{id}(t-1)) + \\ + b * (p_{gd} - x_{id}(t-1))$$

$$x_{id}(t) = c * x_{id}(t-1) + d * v_{id}(t)$$

where $v_{id}(t)$ = velocity of the i^{th} particle of the d^{th} dimension, p_{id} = best position for i^{th} particle of the d^{th} dimension, $x_{id}(t)$ = position of the i^{th} particle of the d^{th} dimension, p_{gd} = global best position of the d^{th} dimension. The parameter a represents the inertia weight that determines the influence of old velocity. The parameter b is the acceleration coefficient that determines the influence of local best position and the global best position. Trelea [26] recommends two sets of parameter values for a and b after simulation experiments, i.e. ($a = 0.6$, $b = 1.7$) and ($a = 0.73$, $b = 1.49$). These two are commonly referred to as TreleaI and TreleaII. The parameters c and d affect a particle's new position and are commonly set to 1.

A three layer feed forward neural network model has been used in this study. The number of layers and the number of nodes at each layer is determined through experimentation. The final ANN structure consisted of one input layer, two hidden layers and one output layer. The two hidden layers consisted of 3 and 5 nodes while the output layer consisted of 1 node. The number of independent variables in the problem determined the number of input nodes. The hyperbolic tangent sigmoid and linear transfer functions have been used for the hidden and output nodes respectively. The PSO variation Trealea II, implemented as part of the MATLAB PSO toolbox [27], has been used in this study with the number of particles in the swarm set to 25 and number of iterations set to 2000. The mean squared error has been used as the fitness function.

B. Artificial immune recognition system (AIRS)

The concepts of artificial immune systems have been used to produce a supervised learning system called artificial immune recognition system (AIRS) [28].

Based on immune network theory, Timmis et al. [29] developed a resource limited artificial immune system and introduced the metaphor of an artificial recognition ball (ARB), a collection of similar B-cells. B-cells are anti-body secreting cells that are a response of the immune system against the attack of a disease. For the resource-limited AIS, a predefined number of resources exists. The ARBs compete based on their stimulation level. Least stimulated cells are removed while an ARB having higher stimulation value could claim more resources. The AIRS is based on the similar idea of a resource-limited system but many of the network principles were abandoned in favor of a simple population-based model [28].

Table III
GEP CONTROL PARAMETERS.

Control Parameter	*Value*
Population size	50
Genes per chromosome	4
Gene head length	8
Termination condition	2000 generations
Functions	{+,−,∗,/,sin,cos,log,sqrt}
Tree initialization	Random
Mutation rate, Inversion rate, IS transposition rate, Root transposition rate, Gene transposition rate, One-point recombination rate, Two-point recombination rate, Gene recombination rate	0.04, 0.1, 0.1, 0.1, 0.1, 0.3, 0.3, 0.1
Selection method	roulette-wheel

The AIRS algorithm consists of five steps [28]: *1) Initialization*, *2) Memory cell identification and ARB generation*, *3) Competition for resources and development of a candidate memory cell*, *4) Memory cell introduction* and *5) Classification*. The details of each of these steps can be found in [28] and are omitted due to space constraints.

The WEKA plug-in for AIRS [30] has been used with the following parameters: Affinity threshold = 0.2, clonal rate = 10, hypermutation rate = 2, knn = 3, mutation rate = 0.1, stimulation value = 0.9 and total resources = 150.

C. Gene expression programming (GEP)

Gene expression programming (GEP) is an evolutionary algorithm introduced by Ferreira [31]. The individuals making up a population in GEP are encoded as linear strings of fixed length that are later expressed as nonlinear expression trees of different sizes and shapes. Thus GEP brings a separation between genotype (the linear chromosomes) and the phenotype (the expression trees). GEP uses karva notation [31] to encode solutions where each gene has a head composed of functions and terminals, and a tail composed only of terminals [31]. The GEP algorithm begins with a random generation of chromosomes to form an initial population. The chromosomes are expressed as trees and evaluated for fitness (Mean squared error, $\sum_{i=1}^{N}(P_i - T_i)^2$ where P_i and T_i are i^{th} predicted and target values). If the termination criterion is not already reached, the selection process selects best-fit individuals for making a new population for the next generation. Genetic diversity is inculcated in the new population using genetic operators of replication, mutation, inversion, insertion sequence (IS) transposition, root transposition, gene transposition, gene recombination, one-and-two point recombination [31], [32]. The process is iterated for a certain number of generations until the termination criterion is reached. The details of genetic operators are omitted due to space constraints and can be found in [31].

The parameter settings for GEP used in this study are shown in Table III.

Table IV
GP CONTROL PARAMETERS.

Control parameter	*Value*
Population size	50
Termination condition	2000 generations
Function set	{+,−,∗,/,sin,cos,log,sqrt}
Tree initialization	Ramped half-and-half method
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	roulette-wheel

D. Genetic programming (GP)

GP, an evolutionary computation technique, is an extension of genetic algorithms [33]. The population structures (individuals) in GP are not fixed length character strings but programs that, when executed, are the candidate solutions to the problem. For the symbolic regression application of GP, programs are expressed as syntax trees, with the nodes indicating the instructions to execute and are called functions (e.g. $\min$, $*$, $+$, $/$), while the tree leaves are called terminals which may consist of independent variables of the problem and random constants (e.g. x, y, 3). The worth of an individual GP program in solving the problem is assessed using a fitness evaluation. The control parameters limit and control how the search is performed like setting the population size and probabilities of performing the genetic operations. The termination criterion specifies the ending condition for the GP run and typically includes a maximum number of generations [7]. GP iteratively transforms a population of computer programs into a new generation of programs using various genetic operators. Typical operators include crossover, mutation and reproduction. The details of genetic operators are omitted due to space constraints but can be found in [34]. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table IV.

E. Multiple regression (MR)

Multiple regression is an extension of simple least-square regression for more than one independent (predictor) variables to estimate the values of the dependent (criterion) variable. The general form of the equation is:

$$y' = a + b_1x_1 + b_2x_2 + \ldots + b_kx_k$$

where y' is the predicted value of the dependent variable, x_1,x_2, …, x_k are independent (predictor) variables and a, b are the coefficients that must be determined from sample data. As in simple regression, least square solution is used to determine the best regression equation.

Table VI
ABSOLUTE RELATIVE ERROR, ARE, VALUES FOR DIFFERENT TECHNIQUES AT UNIT, FUNCTION, INTEGRATION AND SYSTEM TEST PHASES.

	PSO-ANN	AIRS	GEP	GP	MR
Unit testing	2.66	0.81	0.95	1.16	5.34
Function testing	2.01	4.51	2.95	3.15	4.36
Integration testing	0.99	0.56	1.61	0.36	1.73
System testing	0.43	0.7	0.59	0.04	0.06

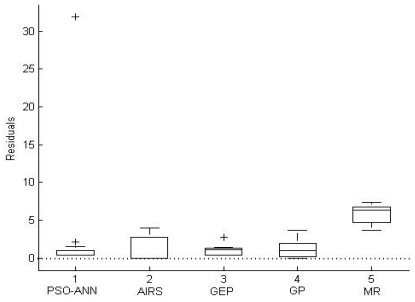
VI. RESULTS AND ANALYSIS

The box plots of the residuals for faults-slip-through at the *unit testing phase* (from the five techniques: PSO-ANN, AIRS, GEP, GP and MR), are shown in Figure 2(a).

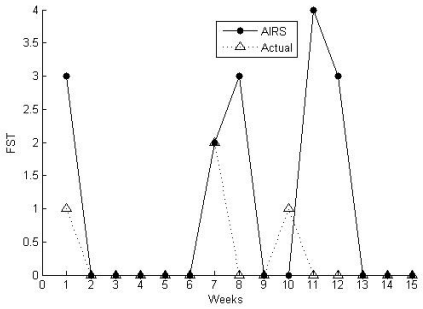
It can be observed that the box plots for PSO-ANN, GEP and GP are narrower than those of AIRS and MR, indicating that the AIRS and MR residuals are more spread out as compared to the other techniques. The box plots for PSO-ANN and GEP are smaller in comparison with that of GP. Since the residual box plots were skewed, we resorted to the non-parametric Kruskal-Wallis test to examine any statistical differences between the residuals and to confirm the trend observed from the box plots (Figure 2(a)). The result of the Kruskal-Wallis test ($p = 0.0000000025$) suggested that it is possible to reject the null hypothesis of all samples being drawn from the same population at significance level, $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. We used Wilcoxon rank sum test to investigate which of the techniques' residuals differ significantly. The null hypothesis of samples being drawn from identical distributions was rejected at $\alpha = 0.05$ for MR:PSO-ANN, MR:GEP, MR:GP, MR:AIRS. The corresponding p-values for these pairs appear in Table VII. The p-values of rest of the pair-wise comparisons (PSO-ANN:AIRS, PSO-ANN:GEP, PSO-ANN:GP, AIRS:GEP, AIRS:GP, GEP:GP) suggested that there were no significant differences between the model residuals. This suggests that while the group of search-based techniques do not differ significantly pair-wise for residuals, MR residuals are significantly different. We also measured the goodness of fit for predictions from each technique relative to the actual FST at the unit testing phase. Table V shows the K-S statistic J for each of the five techniques. The AIRS technique showed statistically significant goodness of fit which has to do with an exact match of actual FST data on 9 out of 15 instances (Figure 2(b)). Table VI shows the ARE measure for each of the five techniques at the unit test phase. AIRS gives the least ARE value, followed by GEP and GP.

The box plots of the residuals from the five techniques for FST at the *function testing phase* are shown in Figure 2(c). We can observe that there is a greater spread of box plots for each of the techniques as compared with those at the unit testing phase. The plots for each of the techniques are also farther away from the 0 mark on the y-axis, with PSO-ANN being the closest of all but having a much larger spread as compared with other box plots. To test for any significant differences between the residuals, the results of applying Kruskal-Wallis test ($p = 0.00022$) suggested that the null hypothesis of all the samples being drawn from the same population can be rejected at $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. To further investigate which of the techniques' residuals differ, we used the Wilcoxon rank sum test for pair-wise comparisons. Table VII shows the significant p-values for pair-wise comparisons where the null hypothesis of samples being drawn from identical distributions was rejected. Except for the pairs GEP:GP and MR:AIRS, all pair-wise model residuals differ significantly. This confirms that there is much variability in the residuals at the function testing phase for every pair of the techniques except for GEP:GP and MR:AIRS. The pairs GEP:GP and MR:AIRS do not differ significantly but as is evident from their box plots (Figure 2(c)) they both are not in close proximity of mark 0 on the y-axis. The goodness of fit of predictions from each of the techniques in relation to the actual FST during function testing is measured using the K-S test. The results appear in Table V. PSO-ANN has statistically significant goodness of fit which is also evident from Figure 2(d) where the PSO-ANN curve is closer to the actual FST curve at the function testing phase. The box plot of PSO-ANN in Figure 2(c) also suggests that the median is much nearer to the 0 mark than other techniques. PSO-ANN is also able to give best ARE values in comparison with other techniques (Table VI).

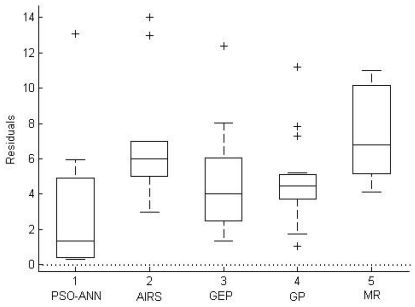
The residual box plots from the five techniques for FST at the *integration testing phase* are shown in Figure 2(e). The box plots in this case are narrow for all the techniques with GP having a median closest to the 0 mark on the y-axis while MR box plot appears farthest away. This indicates that while there does not seem to be much difference in absolute residuals of the four search-based techniques, the residuals of MR appear to be differently placed. In order to confirm this, the Kruskal-Wallis test was performed and the result ($p = 0.000077$) showed that at least one sample median is significantly different from others. As with previous two test phases the Wilcoxon rank sum test for pair-wise comparisons was performed. The p-values in Table VII confirmed that for all pairs involving MR (i.e. MR:PSO-ANN, MR:AIRS, MR:GEP, MR:GP) we can reject the null hypothesis of samples being drawn from identical distributions. For all the pair-wise comparisons of residuals of four search-based techniques, there were no significant differences between samples. The results of Kolmogorov-Smirnov test for measuring the goodness of fit of predictions in relation to the actual FST at the integration testing phase are given in Table V. Table V shows that AIRS and GP have statistically significant goodness of fit, with their K-S



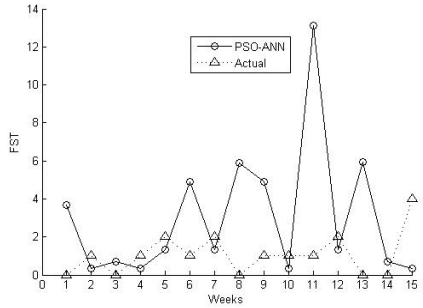
(a) Box plots of the residuals for each technique at the unit testing phase.



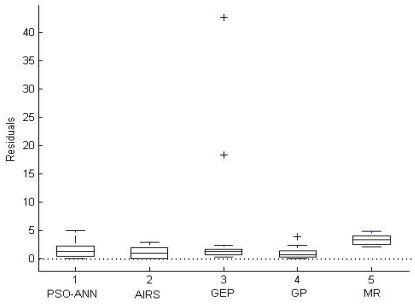
(b) Plot of predicted vs. the actual FST values at the unit testing phase for techniques having significant goodness of fit.



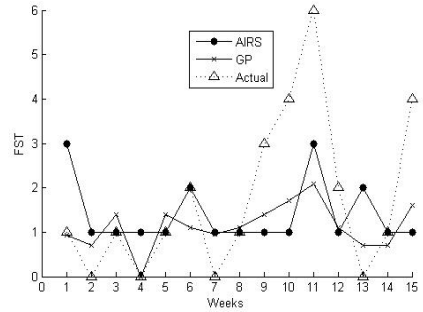
(c) Box plots of the residuals for each technique at the function testing phase.



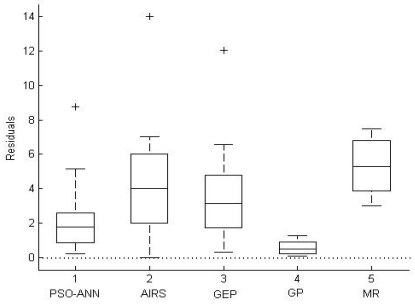
(d) Plot of predicted vs. the actual FST values at the function testing phase for techniques having significant goodness of fit.



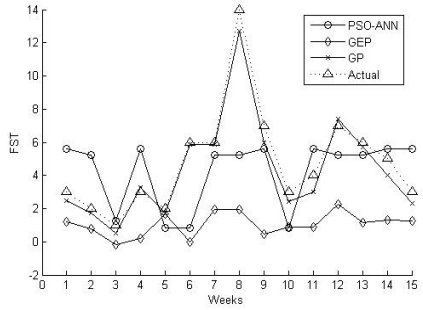
(e) Box plots of the residuals for each technique at the integration testing phase.



(f) Plot of predicted vs. the actual FST values at the integration testing phase for techniques having significant goodness of fit.



(g) Box plots of the residuals for each technique at the system testing phase.



(h) Plot of predicted vs. the actual FST values at the system testing phase for techniques having significant goodness of fit.

Figure 2. Box plots of the residuals and the plot of predictions for unit, function, integration and system testing phase.

Table V
TWO-SAMPLE TWO SIDED K-S TEST RESULTS FOR PREDICTIONS AT DIFFERENT TESTING PHASES WITH CRITICAL VALUE J_α = 0.54.

K-S test statistic, J																				
Unit testing					Function testing					Integration testing					System testing				
PSO-ANN	AIRS	GEP	GP	MR	PSO-ANN	AIRS	GEP	GP	MR	PSO-ANN	AIRS	GEP	GP	MR	PSO-ANN	AIRS	GEP	GP	MR
0.8	0.27	0.8	0.6	1	0.40	0.90	0.90	0.90	1	0.60	0.27	0.60	0.33	0.73	0.40	0.73	0.27	0.20	0.67

Table VII
P-VALUES AFTER APPLYING THE WILCOXON RANK SUM TEST ON RESIDUALS AT UNIT, FUNCTION, INTEGRATION AND SYSTEM TESTING PHASES WITH α = 0.05.

Unit testing				*Function testing*							*Integration testing*				*System testing*						
MR:PSO-ANN	MR:AIRS	MR:GEP	MR:GP	PSO-ANN:AIRS	PSO-ANN:GEP	PSO-ANN:GP	AIRS:GEP	AIRS:GP	MR:PSO-ANN	MR:GEP	MR:GP	MR:PSO-ANN	MR:AIRS	MR:GEP	MR:GP	PSO-ANN:AIRS	PSO-ANN:GP	AIRS:GP	GEP:GP	MR:PSO-ANN	MR:GEP	MR:GP
0.00004	0.000003	0.000003	0.000002	0.00	0.04	0.04	0.03	0.03	0.0014	0.008	0.006	0.0004	0.00009	0.0008	0.00006	0.04	0.00	0.00	0.00	0.0002	0.02	0.000003

test statistic being less than the critical value of $J_\alpha = 0.54$ (Figure 2(f)). As for ARE, GP has the lowest ARE value in comparison with other techniques (Table VI).

The residual box plots from the five techniques for predicting FST at the *system testing phase* are shown in Figure 2(g). As with the box plots for the FST at the function testing phase (Figure 2(c)), there is a greater variance for different techniques at the system testing phase (Figure 2(g)). The sizes of the box plots vary, with AIRS having a larger box plot in comparison with the others. The box plot for GP is smallest with the median closer to 0. To test for any significant differences between the residuals, the results of applying the Kruskal-Wallis test ($p = 0.0000001$) suggested that the null hypothesis of all the samples being drawn from the same population can be rejected at $\alpha = 0.05$. We subsequently used the Wilcoxon rank sum test as a post-hoc test to determine which particular comparisons differ significantly. The p-values obtained are shown in Table VII. The results indicate that there are significant differences in residuals between the pairwise combinations of PSO-ANN:GP, AIRS:GP, GEP:GP, PSO-ANN:AIRS, MR:PSO-ANN, MR:GEP, MR:GP (having a p-value of less than 0.05). A common trend from this result show that the predictions made by GP are significantly different from the others, a trend that is also confirmed from the box plots in Figure 2(g). The goodness of fit of predictions from each of the techniques with actual FST data during system testing phase is measured using the K-S test. The results appear in Table V. The results in Table V show that PSO-ANN, GEP and GP have statistically significant goodness of fit (Figure 2(h)). As is evident from the box plots in Figure 2(g), AIRS has a wider box plot and consequently has a non-significant goodness of fit in relation to the actual FST at the system testing phase. ARE values from different techniques (Table VI) indicate that GP gives the lowest ARE values.

In summary, a general trend from the results show that the search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting FST at unit, function, integration and system testing phases. The results show that MR model residuals are different and inferior for majority of the pair-wise comparisons with search-based techniques' residuals for all the testing phases. The goodness of fit of MR is not significant and ARE values not the lowest for any of the testing phases in comparison with search-based techniques. At the *unit* testing phase the model residuals for four search-based techniques do not differ significantly but AIRS performance is better in terms of having both statistically significant goodness of fit and lowest ARE value. At the *function* testing phase the box plots of model residuals for the search-based techniques show a certain degree of variability but PSO-ANN box plot is promising with its median nearer to the 0 mark on y-axis. PSO-ANN is also better in having both statistically significant goodness of fit and lowest ARE value. At the *integration* testing phase, the performance of GP is better with having the median residuals closer to the 0 mark on y-axis, statistically significant goodness of fit and lowest ARE value; while at the *system* testing phase GP is again better in terms of having significantly different and better residuals, goodness of fit and lowest ARE.

So while MR is not able to perform as good as the search-based techniques, GP performs better for two testing phases (integration and system) and AIRS, PSO-ANN perform better for unit and function testing phases respectively.

VII. DISCUSSION

One of the basic objectives of doing measurements is monitoring of activities so that action can be taken as early as possible to control the final outcome. With this objective in focus, FST metrics work towards the goal of minimization of avoidable rework by finding faults where they are most cost-effective to find. Early prediction of FST at different testing phases is an important decision support to the development team whereby advance notification of improvement potential can be made.

An overall interesting outcome of the study is that a

variety of search-based techniques perform better than multiple regression for FST prediction, especially GP performs better for two of the four testing phases. The search-based techniques are also likely to be robust to changes in the process since they are assumption-free and do not require prior definition of the functional structure to evolve.

While the focus of this paper has been on a quantitative evaluation of techniques, one has to be mindful that there are additional qualitative characteristics of the empirical models that are important for real-world use, e.g. lower cost of ownership and robustness to withstand minor process changes. While these qualitative issues are not the focus in this study, they are still worth investigating e.g. by asking industrial professionals to fill structured questionnaires to assess the usefulness and usability aspects. Additionally, building automated tool support for different search-based techniques would ease parameter tuning and would allow the practitioners to evaluate the results of applying multiple techniques more easily. Moreover, selection of predictor variables that are easy to gather (e.g. the project level metrics at the subject company in this study) and that do not conflict with the development life cycle have better chances of industry acceptance. There is evidence to support that general process level metrics are more accurate than code/ structural metrics [35], however this subject requires further research.

Another interesting outcome of this study is the performance of search-based techniques outside their respective training ranges, i.e., the predictions are evaluated for 15 weeks of an on-going project after being trained on another baseline project data. This is to say that the over-fitting is within acceptable limits that can be judged from the ARE values in Table VI where the values for GP are between 0.04 and 3.15 and for PSO-ANN between 0.43 and 2.66; they are acceptable not being only smaller but also considering the fact that we are dealing with large scale projects where the degree of variability in fault occurrences can be large. This issue is also related to the amount of data available for training the different techniques which in case of large projects is typically available enough for the prediction techniques. We used the historical data from the past 45 weeks to train the models.

In short the implication of the results in this study is that they have the potential to provide *early* indications of quantified improvement potential within the software testing life cycle.

VIII. Validity evaluation

Conclusion validity refers to a statistically significant relationship between the treatment and the outcome. We have used non-parametric statistics in this study due to the violation of the normality assumptions required for parametric statistical tests. Since the power of parametric statistical tests are generally higher than non-parametric ones, there is a chance that the use of non-parametric statistics might have missed potential differences in outcomes. However this threat is minimized to an extent by additionally looking at the box-plots where general trends are visible. *Internal validity* refers to a causal relationship between treatment and outcome. A possible threat to internal validity is that we cannot publicize our industrial data sets due to proprietary concerns. However, the transformed representation of the data can be made available if requested. Secondly, each of the search-based techniques are executed 10 times to minimize the effect of randomness inherent in these techniques. *Construct validity* is concerned with the relationship between theory and application. Our choice of selecting project level metrics (Table I) instead of structural code metrics was influenced by multiple factors. First, metrics relevant to work packages (Section IV) have an intuitive appeal for the employees at the subject company where they can relate FST to the proportion of effort invested. Secondly, the existence of a module in multiple work packages made it difficult to obtain consistent metrics at the component level. Thirdly, the intent of this study is to use project level metrics that are readily available and hence reduces the cost of doing such predictions. *External validity* is concerned with generalization of results outside the scope of the study. This study is designed for projects structured according to work packages (Section III), therefore a different set of variables are required for projects structured differently. However the use of project level metrics as predictors of fault-slippages require more empirical studies to increase generalizability, e.g., by using more projects from different domains.

IX. Conclusions

This paper has evaluated the application of five techniques for predicting the fault-slippage in various testing phases using two projects from the telecommunication industry. One of the projects is used as a baseline project to train the techniques while the other on-going project is used to evaluate the prediction performance of each of the techniques. Various project level metrics are used for building the techniques. The results from the empirical study show that the search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting FST at different testing phases based on model residuals, goodness of fit and absolute relative error values. At the unit testing phase, though there are no significant differences between the model residuals of the search-based techniques, AIRS show statistically significant goodness of fit and lowest ARE value. PSO-ANN perform better at function testing phase – having residuals closer to 0, statistically significant goodness of fit and lowest ARE value. At integration and system testing phases, GP perform better with respect to the three measures of model residuals, goodness of fit and absolute relative error values. We conclude that a variety of search-based techniques are applicable for predicting fault-

slippage between different testing phases, especially the use of GP as a prediction technique is promising as it shows better prediction performance for two of the four testing phases.

REFERENCES

[1] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Tran. on SW Eng.*, vol. 25, no. 5, 1999.

[2] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, 2001.

[3] L.-O. Damm, L. Lundberg, and C. Wohlin, "Faults-slip-through–A concept for measuring the efficiency of the test process," *SW Process: Improv. & Prac.*, vol. 11, no. 1, 2006.

[4] L.-O. Damm, "Early and cost-effective software fault detection – measurement and implementation in an industrial setting," Ph.D. dissertation, Blekinge Inst. of Tech., 2007.

[5] M. Staron and W. Meding, "Predicting weekly defect inflow in large software projects based on project planning and test status," *IST*, vol. 50, no. 7–8, 2008.

[6] S. Rakitin, *Software verification and validation for practitioners and managers*, 2nd ed. Artech House., Inc., 2001.

[7] E. K. Burke and G. Kendall, Eds., *Search methodologies – Introductory tutorials in optimization and decision support techniques*. Springer Science and Business Media, Inc., 2005.

[8] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *ICSE'05*, 2005.

[9] L. Briand, K. Emam, B. Freimut, and O. Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content," *IEEE Tran. on SW Eng.*, vol. 26, no. 6, 2000.

[10] J. Munson and T. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Tran. on SW Eng.*, vol. 18, no. 5, 1992.

[11] M. R. Lyu, *Handbook of software reliability engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.

[12] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Tran. on SW Eng.*, vol. 31, no. 10, 2005.

[13] A. Veevers and A. C. Marshall, "A relationship between software coverage metrics and reliability," *Software Testing, Verification and Reliability*, vol. 4, no. 1, 1994.

[14] V. Challagulla, F. Bastani, I. Yen, and R. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Proc. of the 10th IEEE Workshop on OO Real-Time Dependable Systems*, 2005.

[15] W. Afzal and R. Torkar, "A comparative evaluation of using genetic programming for predicting fault count data," in *Proc. of the 3rd int. conf. on SW eng. advances*. IEEE, 2008.

[16] C. Catal and B. Diri, "Software fault prediction with object-oriented metrics based AIRS," in *PROFES'07*. LNCS, 2007.

[17] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy, "SVM and kernel methods toolbox," Perception Systémes et Information, INSA de Rouen, Rouen, France, 2005.

[18] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd, "What accuracy statistics really measure?" *IEE Proceedings Software*, vol. 148, no. 3, 2001.

[19] M. Shepperd, M. Cartwright, and G. Kadoda, "On building prediction systems for software engineers," *Empirical Software Engineering*, vol. 5, no. 3, 2000.

[20] T. Khoshgoftaar, N. Seliya, and N. Sundaresh, "An empirical study of predicting software faults with CBR," *Software Quality Control*, vol. 14, no. 2, 2006.

[21] S. Russell and P. Norvig, *Artificial intelligence—A modern approach*. USA: Prentice Hall Series in AI, 2003.

[22] G. Zhang, "Avoiding pitfalls in neural net. research," *IEEE Tran. on Systems Man and Cybernetics*, vol. 37, no. 1, 2007.

[23] B. Curry and P. Morgan, "Neural networks: A need for caution," *Omega*, vol. 25, no. 1, 1997.

[24] G. K. Jha, P. Thulasiraman, and R. K. Thulasiram, "PSO based neural network for time series forecasting," in *International Joint Conference on Neural Networks*, 2009.

[25] J. Kennedy and R. Eberhart, "PSO," in *Proc. of the IEEE Int. Conf. on Neural Networks*, 1995.

[26] I. C. Trelea, "The PSO algorithm: convergence analysis and parameter selection," *IP Letters*, vol. 85, no. 6, 2003.

[27] B. Birge, "PSO–MATLAB toolbox," 2005, http://www.mathworks.com/matlabcentral/fileexchange/7506-particle-swarm-optimization-toolbox.

[28] A. Watkins, J. Timmis, and L. Boggess, "Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm," *GPEM*, vol. 5, no. 3, 2004.

[29] J. Timmis, M. Neal, and J. Hunt, "An artificial immune system for data analysis," *Biosystems*, vol. 55, no. 1-3, 2000.

[30] J. Brownlee, "WEKA plug-in for AIRS." http://wekaclassalgos.sourceforge.net/, 2010.

[31] C. Ferreira, "GEP: A new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, 2001.

[32] P. H. Sherrod, *DTREG*, Predictive modeling and forecasting, 2010, http://www.dtreg.com/DownloadManual.htm.

[33] J. Koza, *GP: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[34] S. Silva, "GPLAB—A genetic programming toolbox for MATLAB," http://gplab.sourceforge.net.

[35] E. Arisholm, L. Briand, and E. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *JSS*, vol. 83, no. 1, 2010.

Genetic Programming for Effort Estimation: an Analysis of the Impact of Different Fitness Functions

Filomena Ferrucci, Carmine Gravino, Rocco Oliveto, Federica Sarro
DMI, University of Salerno, via Ponte don Melillo, 84084 Fisciano (SA), Italy
{*fferrucci, gravino, roliveto, fsarro*}*@unisa.it*

***Abstract*—*Context*: The use of search-based methods has been recently proposed for software development effort estimation and some case studies have been carried out to assess the effectiveness of Genetic Programming (GP). The results reported in the literature showed that GP can provide an estimation accuracy comparable or slightly better than some widely used techniques and encouraged further research to investigate whether varying the fitness function the estimation accuracy can be improved. *Aim*: Starting from these considerations, in this paper we report on a case study aiming to analyse the role played by some fitness functions for the accuracy of the estimates. *Method*: We performed a case study based on a publicly available dataset, i.e., Desharnais, by applying a 3-fold cross validation and employing summary measures and statistical tests for the analysis of the results. Moreover, we compared the accuracy of the obtained estimates with those achieved using some widely used estimation methods, namely Case-Based Reasoning (CBR) and Manual StepWise Regression (MSWR). *Results*: The obtained results highlight that the fitness function choice significantly affected the estimation accuracy. The results also revealed that GP provided significantly better estimates than CBR and comparable with those of MSWR for the considered dataset.**

***Keywords*-Software Development Effort Estimation; Genetic Programming; Empirical Studies.**

I. Introduction

Effort estimation is a critical activity for planning and monitoring software project development and for delivering the product on time and within budget. Significant over or under-estimates can be very expensive for a company and the competitiveness of a software company heavily depends on the ability of its project managers to accurately predict in advance the effort required to develop software systems. Several approaches have been proposed to estimate software development effort. Among them, data-driven approaches exploit data from past projects to estimate the effort for a new project under development [1] [2] [3] [4]. These data consist of information about some relevant factors (named cost drivers) and the effort actually spent to develop the projects. Usually a data-driven method tries to explain the relation between effort and cost drivers building an estimation model (equation) that is used to estimate the effort for a new project. Widely used and studied data-driven approaches are Linear and StepWise Regression (LR and SWR), and Case Based-Reasoning (CBR) [5].

Recently the use of search-based methods has been suggested to address the software development effort estimation problem [6] [7]. Such a problem can be formulated as an optimisation problem where we have to identify the estimation model which provides the best predictions. In the literature some attempts have been reported on the use of Genetic Programming (GP) for building software development effort estimation models [8] [9] [10] [11]. All the studies showed that GP provided models with estimation accuracy comparable with the ones obtained employing some widely used estimation techniques. However, several crucial design choices need to be made when using GP, such as population size, maximum number of generations, genetic operator rates, and fitness function [6] [8]. Special relevance has the choice of the fitness function to guide the search towards a solution able to provide accurate estimates. The common fitness function analysed in the previous studies was based on the Mean Magnitude of Relative Error (MMRE) [12] that represents the most widely used evaluation criterion for assessing the accuracy of a software prediction model. However, it was observed that employing MMRE as fitness function had the effect to degrade a lot of other measures that are usually employed to complement the analysis of the estimation accuracy [8]. Moreover, it has also been argued that the choice of the criterion for establishing the best model can be a managerial issue. In particular, a project manager could prefer to use MMRE as the criterion for judging the quality of the prediction, while another might prefer to use another criterion, just for example Pred(25) [12]. Thus, further research was solicited to analyse which measures is the most appropriate as fitness function.

Based on these considerations we have carried out an empirical analysis to provide an insight on the use of GP for effort estimation and in particular to analyse how the estimation accuracy of GP is affected by the use of different fitness functions. To this end we experimented with different fitness functions based on widely recognised indicators used to evaluate the accuracy of the estimates (i.e., MMRE, MdMRE, Pred(25), MEMRE, and MdEMRE [12] [13]) and combinations of them (i.e., Pred(25) and MMRE, Pred(25) and MdMRE). The empirical study was based on a publicly available dataset, i.e., Desharnais [14], which has been widely and recently used to evaluate and

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.20

compare estimation methods, see e.g., [4] [8] [15] [16] [17]. We also performed a comparison of the effectiveness of GP with widely used estimation methods, namely Manual SWR (MSWR) and CBR. The analysis of the results is based on widely used summary measures (i.e., MMRE, MdMRE, Pred(25), MEMRE, and MdEMRE [12] [13]) and on statistical test. To the best of our knowledge, only summary measures, including MMRE and Pred(25), have been used to compare estimations in the case studies carried out so far with the use of GP [8] [9] [10].

The rest of the paper is organised as follows. Section II introduces GP. The experimental method is described in Section III, while the results are reported in Section IV. Case study validity is discussed in Section V. Section VI reports on related work, while final remarks and directions for future work are presented in Section VII.

II. Genetic Programming

Genetic Programming (GP) [18] belongs to the family of evolutionary algorithms that, inspired by the theory of natural evolution, simulates the evolution of species emphasising the law of survival of the strongest to solve, or approximately solve, optimisation problems. Thus, these algorithms create consecutive populations of individuals, considered as feasible solutions for a given problem, to search for a solution which gives the best approximation of the optimum for the problem under investigation. To this end, a fitness function is used to evaluate the goodness (i.e., fitness) of the solutions represented by the individuals and genetic operators based on selection and reproduction are employed to create new populations (i.e., generations). The elementary evolutionary process of these algorithms is composed by the following steps:

- $\mathbf{S}_1$: a random initial population is generated and a fitness function is used to assign a fitness value to each individual;
- $\mathbf{S}_2$: according to their fitness value some individuals are selected to form the parents and new individuals are created by applying genetic operators (i.e., crossover and mutation). In particular, the crossover operator combines two individuals (i.e., parents) to form one or two new individuals (i.e., offspring), while the mutation operator is employed to randomly modify an individual. Then, to determine who will survive among the offspring and their parents a survivor selection is applied according to the individuals' fitness values;
- $\mathbf{S}_3$: step $\mathbf{S}_2$ is repeated until stopping criteria hold.

With respect to other evolutionary methods, GP is characterised by the fact that individuals are computer programs (e.g., mathematical expressions) usually encoded as a tree where the leaves are terminals (e.g., operands) and the internal nodes are functions (e.g., mathematical operators). The initial population is usually generated building random trees of fixed or variable depth or a combination of them. The crossover and mutation operators are defined exchanging parent subtrees and making random changes in trees, respectively. Note that at each generation these operators are applied with a certain probability, named crossover rate and mutation rate. The stopping criterion for the evolutionary process is usually based on a maximum number of generations. This stopping criterion can be combined with other criteria to reduce the computation time. For example, the search process can be stopped after a certain number of generations or after some number of generations that do not provide an improvement in the fitness value.

III. Case study planning

This section presents the design of the case study we carried out to get an insight in the use of GP for effort estimation. The research goals of our study can be outlined as follows:

- $\mathbf{RG}_1$: Analysing the impact of different fitness functions on the accuracy of the estimation models built with GP.
- $\mathbf{RG}_2$: Comparing the estimates achieved by applying GP with the estimates obtained using widely and successfully employed estimation methods.

To address research goals $\mathbf{RG}_1$ we experimented with several fitness functions as reported and discussed in Section III-B. The second research goal ($\mathbf{RG}_2$) aims to get an insight on the estimation accuracy of GP and understand the actual effectiveness of the technique with respect to other effort estimation methods. For this reason, we first verified whether the estimates obtained with GP were characterised by significantly better accuracy than the simply mean and median of effort of past projects. Indeed, if the investigated estimation method does not outperform the results achieved by using the mean or median effort it cannot be transferred to industry [19] [20]. For a software company it could be more useful to simply use the mean or the median effort of past projects rather than dealing with complex computations of estimation methods. Moreover, we compared the estimations achieved using GP with those obtained by using MSWR [20] and CBR [4] to have other benchmarks to assess the effectiveness of GP.

A. Dataset Selection

To carry out the empirical study we exploited an industrial dataset comprising 81 software projects. This dataset was derived from a Canadian software house by Jean-Marc Desharnais [14]. It is one of the largest, publicly available, datasets and it has been widely and recently used to evaluate estimation methods, see e.g., [4] [8] [15] [16] [17].

Table I reports the description of the eleven variables (nine independent and two dependent) included in the Desharnais dataset. In our analysis we considered as dependent variable the total effort while we did not consider the length of the code [8]. Moreover, we excluded from the analysis four

Table I
PROJECT FEATURES OF THE DESHARNAIS DATASET

Feature	Description	Type
TeamExp	The team experience measured in years	Discrete
ManagerExp	The manager experience measured in years	Discrete
Entities	The number of the entities in the system data model	Discrete
Transactions	The number of basic logical transactions in the system	Discrete
AdjustedFPs	The adjusted Function Points	Continuous
RawFPs	The raw Function Points	Continuous
Envergure	A complex measure derived from other factors defining the environment	Discrete
Language	The language used to develop the system	Categorical
YearEnd	The project year finished	Discrete
Effort	The actual effort measured in person hours (dependent variable)	Discrete
Length	The length of the code (dependent variable)	Discrete

Table II
DESCRIPTIVE STATISTICS OF THE DESHARNAIS DATASET FACTORS

Variable	Min	Max	Mean	Std. Dev.
TeamExp	0	4	2.30	1.33
ManagerExp	0	4	2.65	1.52
Entities	7	386	121.540	86.11
Transactions	9	661	162.940	146.08
AdjustedFPs	73	1127	284.480	182.26
RawFPs	62	1116	282.39	186.36
Envergure	5	52	27.24	8.600
Effort	546	2349	4903.95	4188.19

projects that had missing values. The same choice has been done in other studies (e.g., [4] [17] [15]). Categorical (or nominal) variables (i.e., Language and YearEnd) were also excluded from the analysis, as done in [15].

B. Setting of the Experimented GP-based Method

In this section we present the design choices we made in defining the GP-based estimation method experimented in our case study.

1) Solution Representation: In the context of effort estimation a solution consists of an estimation model described by an equation of this type:

$$\textit{Effort} = c_1\ op_1\ f_1\ op_2\ ...\ op_{2n-2}\ c_n\ op_{2n-1}\ f_n\ op_{2n}\ C \quad (1)$$

where f_i represents the value of the i^{th} project feature and c_i its coefficient, C represents a constant, while op_i represents the i^{th} mathematical operator of the model. It is worth noting that the equations feasible for the effort estimation problem are those providing positive value for *Effort*.

To encode such a solution we used a binary tree containing features and coefficients as leafs and mathematical operators as internal nodes. In particular, we took into account the following mathematical operators $\{+, -, \cdot, exp, ln\}$.

According to [21] the initial population is generated by building $10V$ random trees of fixed depth, where V is the number of features, aiming at achieving a good compromise between the running time of GP and the accuracy of the estimates.

2) Fitness Function: The fitness function guides the search for the best estimation model. To this end, a suitable fitness function should be able to determine whether an estimation model leads to better predictions than another. In the literature, a large number of different prediction accuracy measures have been proposed. The most widely used are MMRE and Pred(25) [12]. The former is the mean of Magnitude of Relative Error (MRE), where MRE [12] is defined as:

$$\textit{MRE} = \frac{|\textit{Effort}_{real} - \textit{Effort}_{estimated}|}{\textit{Effort}_{real}} \quad (2)$$

where $\textit{Effort}_{real}$ and $\textit{Effort}_{estimated}$ are the actual and the estimated efforts, respectively. MRE is calculated for each project whose effort has to be estimated and MMRE is used to have a cumulative measure of the error. Another cumulative measure widely employed is the Median of MRE (MdMRE) which is less sensitive to extreme values [22].

The *Prediction at level l* – Pred(l) – [12] is another useful indicator that measures the percentage of the estimates whose error is less than *l*% and *l* is usually set at 25. It can be defined as:

$$Pred(25) = \frac{k}{N} \quad (3)$$

where N is the total number of projects and k is the number of observations whose MRE is less than or equal to 0.25.

Kitchenham *et al.* [13] suggest also the use of the Magnitude of Relative Error relative to the Estimate (EMRE). The EMRE has the same form of MRE, but the denominator is the estimate, giving thus a stronger penalty to underestimates:

$$\textit{EMRE} = \frac{|\textit{Effort}_{real} - \textit{Effort}_{estimated}|}{\textit{Effort}_{estimated}} \quad (4)$$

As well as for MRE, we can also calculate the mean EMRE (MEMRE) and median EMRE (MdEMRE).

To address research goal $\mathbf{RG}_1$ we experimented with each of the above accuracy measures as fitness function to analyse the impact on the estimation accuracy of the constructed models. Moreover, the observation that different accuracy measures take into account different aspects of predictions accuracy [13] suggested us to investigate also the effectiveness of some combinations of those accuracy measures. In particular, we also experimented with $\frac{Pred(25)}{MMRE}$ and $\frac{Pred(25)}{MdMRE}$ as fitness functions[1].

3) Evolutionary Process: The evolutionary process we experimented with employed two widely used selection operators, i.e., roulette wheel selector and tournament selector [18], whereas the crossover and mutation operators are specific for our solution encoding.

In particular, we used the roulette wheel selector [18] to choose the individuals for reproduction, while we employed the tournament selector [18] to determine the individuals that are included in the next generation (i.e., survivals). The former assigns a roulette slice to each chromosome according to its fitness value. In this way, even if candidate

[1]Thus, if MMRE (MdMRE, MEMRE, or MdEMRE) was used as fitness function the GP goal was to find the solution having the lowest MMRE (MdMRE, MEMRE, or MdEMRE) value. Otherwise, if Pred(25) ($\frac{Pred(25)}{MMRE}$ or $\frac{Pred(25)}{MdMRE}$) was used as fitness function the GP goal was to find the solution having the highest Pred(25) ($\frac{Pred(25)}{MMRE}$ or $\frac{Pred(25)}{MdMRE}$) value.

solutions with a higher fitness have more chance to be selected, there is still a chance that they may be not. On the contrary, using the tournament selector only the best n solutions (usually $n \in [1, 10]$) are copied straight into the next generation.

Crossover and mutation operators were defined to preserve well-formed equations in all offspring. To this end, we used a single point crossover which randomly selects the same point in each tree and swaps the subtrees corresponding to the selected node. Since the two trees are cut at the same point, the trees resulting after the swapping have the same depth as compared to those of parent trees. Concerning the mutation, we employed an operator that selects a node of the tree and randomly changes the associated value. The mutation can affect internal node (i.e., operators) or leaves (i.e., coefficients) of the tree. In particular, when the mutation involves internal node, a new operator $op'_i \in \{\{+, -, \cdot, exp, ln\} \backslash op_i\}$ is randomly generated and assigned to the node, while if the mutation involves a leaf a new coefficient $c'_i \in R$ is assigned to the node. It is worth noting that also the employed mutation preserves the syntactic structure of the equation. Crossover and mutation rate were fixed to 0.5 and 0.1, respectively, since in previous works recommended crossover rate ranged from 0.45 to 0.95 [21] and mutation rate ranged from 0.06 to 0.1 [23].

According to [21] the evolutionary process is stopped after $1000V$ trials, where V is the number of features or if the fitness value of the best solution does not change after $100V$ trials[2].

Since GP does not give the same solution each time it is executed, we performed 10 runs and among the 10 solutions we retained as final prediction model the one that had the fitness value closest to the average value achieved on the training sets in the 10 runs.

C. Validation Method and Evaluation Criteria

In order to verify whether or not a method gives useful estimations of the actual development effort a validation process is required. To this end, we performed a multiple-fold cross validation, partitioning the whole dataset into training sets, for model building, and test sets, for model evaluation. Indeed, when the accuracy of the model is computed using the same dataset employed to build the prediction model, the accuracy evaluation is considered optimistic [5]. Cross validation is widely used in the literature to validate effort estimation models when dealing with medium/small datasets (see, e.g. [1] [2]). In particular, to apply the multiple-fold cross validation, we partitioned the dataset in 3 randomly test sets (one containing 25 observations and two containing 26 observations), and then for each test set we considered the remaining observations as training set to build the estimation model. The three folds are given in Table III to allow for replications of our study.

[2]Since we focused on seven features (see section III-A) we executed GP using a population of 70 (i.e., 10*# features) individuals and the generation process was stopped after 7000 (1000*# features) generations or when the best results did not change after 700 (100*# features) generations.

Table III
THE 3 FOLDS EMPLOYED IN OUR STUDY

	Project Id
Fold 1	11, 19, 24, 77, 01, 26, 78, 02, 05, 12, 22, 23, 35,
(25 observations)	40, 58, 61, 68, 69, 29, 50, 13, 81, 49, 70, 65.
Fold 2	10, 15, 30, 41, 42, 43, 21, 03, 47, 63, 56, 62, 74,
(26 observations)	31, 52, 37, 57, 73, 76, 34, 27, 33, 72, 79, 54, 80.
Fold 3	04, 08, 09, 14, 16, 17, 18, 25, 32, 36, 39, 45, 51,
(26 observations)	53, 55, 59, 60, 67, 71, 06, 07, 20, 28, 46, 48, 64.

Concerning the evaluation of the estimates obtained with the analysed estimation methods, we used several summary measures, namely MMRE, MdMRE, Pred(25), MEMRE and MdEMRE [12][13]. According to [12], a good effort estimation model should have an MMRE less than 0.25, to denote that the mean estimation error should be less than 25%, and a Pred(25) greater than 0.75, meaning that at least 75% of the predicted values should fall within 25% of their actual values. Moreover, we complemented these indicators with the analysis of the boxplots of the absolute residuals, as suggested in [13] [24] [25]. The use of boxplots is widely used in exploratory data analysis since they summarise the data (using five values, i.e., median, upper and lower quartiles, minimum and maximum values, and outliers) through a visual representation [13]. In the context of effort estimation, boxplots are generally used to represent in a visual fashion the amount of the error for a given estimation method. To this end, the spread of the absolute residuals, calculated as $|\mathit{Effort}_{real} - \mathit{Effort}_{estimated}|$, can be graphically rendered.

The analysis of summary measures and boxplots gives only an indication on which is the estimation method that globally gives best effort estimations. In order to establish if one of the estimation methods provides better results than the others it is necessary to test the statistical significance of the obtained results. For this reason we tested the statistical significance of the absolute residuals achieved with different estimation methods [13] [22] [26]. Such an analysis aims at verifying that the estimations of one method are significantly better than the estimations provided by another method. Since (i) the absolute residuals for all the analysed estimation methods were not normally distributed (as confirmed by the Shapiro test [27] for non-normality), and (ii) the data was naturally paired, we used the Wilcoxon Test [28] setting the confidence limit at $\alpha = 0.05$.

IV. ANALYSIS AND INTERPRETATION OF THE RESULTS

The following subsections present and discuss on the results achieved in the empirical study. In subsection IV-C we also compare the results with the ones obtained in the literature exploiting GP on the same dataset.

A. Influence of the Fitness Function

In this section we report the results related to the first research goal and obtained employing different fitness functions, i.e., MMRE, Pred(25), MdMRE, MEMRE, MdEMRE, $\frac{Pred(25)}{MMRE}$, and $\frac{Pred(25)}{MdMRE}$.

To get an insight on the use of these fitness functions, we first analysed the ability of the obtained estimation models to fit data considering the results obtained on the training sets and then we analysed their predictive capability considering the results obtained on the test sets.

Table IV reports on the average summary measures obtained on the training sets. We can observe that the use of MMRE and MdEMRE as fitness functions provided the worst results, whereas the best results were achieved by using Pred(25), MdMRE and $\frac{Pred(25)}{MMRE}$ as fitness function. However, there is no clear winner among them. Furthermore, the use of MEMRE and $\frac{Pred(25)}{MdMRE}$ provided an MMRE value worse than the ones obtained using other fitness functions (e.g., Pred(25)).

Interesting considerations can be made by observing the relationship between the fitness function and the summary measures used to evaluate the estimation model. In particular, the results achieved on the training set (see Table IV) suggest that almost all the fitness functions are able to guide the search to get the best value for the considered summary measure (as highlighted in bold face in Table IV). Indeed, GP with MMRE obtained the best value for MMRE (0.51). This happens also for GP with Pred(25) that gets 0.50 as Pred(25) that is the best value obtained with the considered fitness functions. This does not hold for GP based on MdEMRE and $\frac{Pred(25)}{MdMRE}$ since other fitness functions are able to get better results for MdEMRE and $\frac{Pred(25)}{MdMRE}$ values, respectively. Moreover, the use of some summary measures as fitness functions decreased the value of the other summary measure values. In particular, using MMRE as fitness function, the estimation accuracy in terms of the other summary measures was poor, since MEMRE and MdEMRE values were very high and Pred(25) was very low. This confirms the observation of Burgess of Lefley [8] for MMRE. However, such a phenomenon can be also observed for MEMRE and MdEMRE, whose use determines an increasing of the MMRE value. This does not hold for the use of the other fitness functions (i.e., Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$) that were able to provide good fitness value without decreasing so much the other measures. This observation is also confirmed by graphically comparing the trend of the values of summary measures MMRE, Pred(25), MdMRE, MEMRE, and MdEMRE achieved by GP during the evolution process. As an example, in Figure 1 we can observe that using MMRE as fitness function the values of the Pred(25) and MEMRE became worst during the evolution process, while as we can see in Figure 2 this did not happen using the MdMRE as fitness function.

Table IV
RESULTS ON TRAINING SET USING DIFFERENT FITNESS FUNCTIONS

Fitness Function	MMRE	Pred(25)	MdMRE	MEMRE	MdEMRE
MMRE	**0.51**	0.26	0.44	0.82	0.63
Pred(25)	0.68	**0.50**	0.28	0.42	0.31
MdMRE	0.68	0.44	**0.28**	0.39	0.33
MEMRE	0.85	0.42	0.35	**0.36**	0.32
MdEMRE	1.14	0.32	0.57	0.46	**0.34**
$\frac{Pred(25)}{MMRE}$	0.59	0.48	0.31	0.43	0.32
$\frac{Pred(25)}{MdMRE}$	0.75	0.48	0.29	0.43	0.32

Table V
RESULTS ON TEST SET USING DIFFERENT FITNESS FUNCTIONS

Fitness Function	MMRE	Pred(25)	MdMRE	MEMRE	MdEMRE
MMRE	**0.58**	0.23	0.44	0.84	0.63
Pred(25)	0.68	**0.43**	0.33	**0.38**	**0.31**
MdMRE	0.67	**0.43**	**0.32**	**0.38**	0.32
MEMRE	0.91	0.38	0.36	0.39	0.35
MdEMRE	1.32	0.26	0.62	0.47	0.44
$\frac{Pred(25)}{MMRE}$	0.64	0.39	0.33	0.44	0.34
$\frac{Pred(25)}{MdMRE}$	0.87	0.36	0.40	0.47	0.39

Another interesting observation can be made related to the number of iterations (i.e., generations) performed by GP during the evolution process. In particular, we observed that GP was able to find a solution in a relative low number of generations with all the fitness functions. We also observed that the maximum number of generations was rarely achieved and the evolutionary process was generally stopped because the best solution found did not change after a fixed number of generations. In particular, we compared the trend of the fitness value of the best solution obtained with the trend of the average fitness value of the whole population for all the considered fitness functions. The analysis suggested that less than 1,000 iterations are needed to GP to converge. As an example, when MMRE is used as fitness function the analysis highlighted that after about 700-800 generations the two curves were identical indicating that the best solution found cannot be improved. Thus, we can state that the stopping criteria we used is sufficient for GP to converge and the convergence is not influenced by the fitness function employed.

Table V reports the summary measures related to the accuracy achieved by the models constructed by GP with the analysed fitness functions on the test sets. First of all, we can observe that the summary measures were not much more worse than the ones achieved on the training sets. However, we can observe that none of the exploited fitness functions was able to provide summary measure values that satisfy thresholds provided in [12]. Indeed, Pred(25) value was always less than 0.75 and MMRE and MdMRE values were always greater than 0.25. Moreover, we can observe that the predictive capacity of the estimation models is very similar to the ones achieved on the training sets. Indeed, the use of Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$ provided the best results, while the worst summary measure were achieved by using MMRE and MdEMRE as fitness functions. Finally, despite the use of MEMRE and $\frac{Pred(25)}{MdMRE}$ provided results similar to Pred(25) except for the MMRE value that was

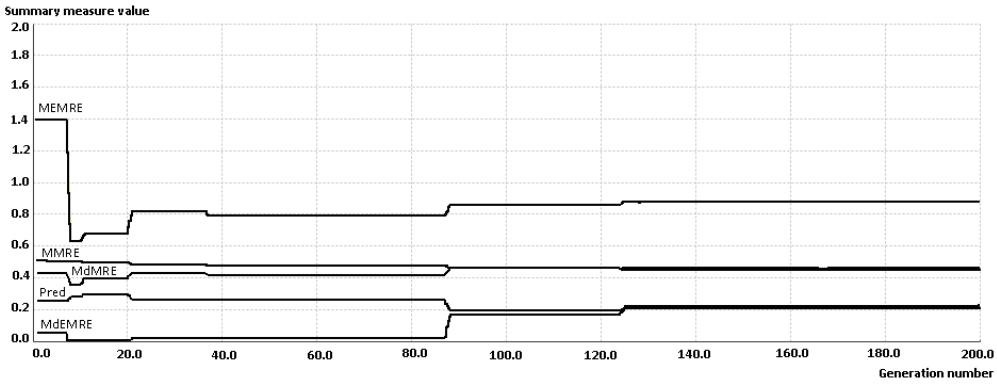


Figure 1. An excerpt of the trend of summary measures when MMRE is used as fitness function.

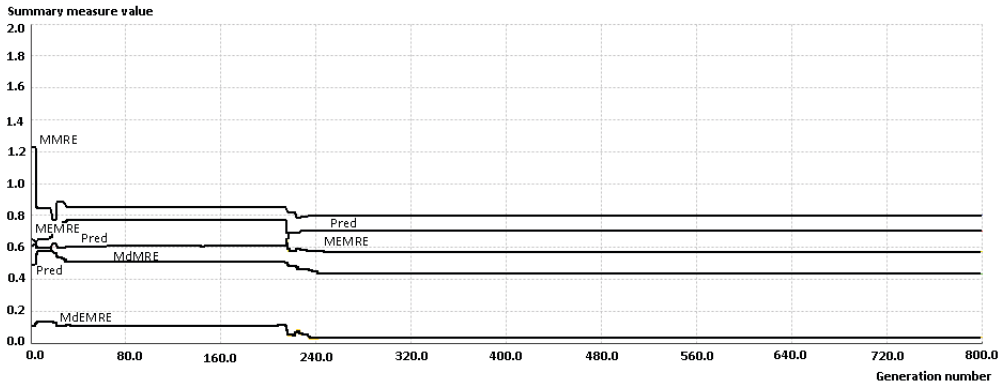


Figure 2. An excerpt of the trend of summary measures when MdMRE is used as fitness function.

worse. As in the case of training sets we also noted that the model employing MMRE (MdEMRE) as fitness function improved the estimation accuracy in terms of MMRE (MdEMRE) but decreased the accuracy in terms of the others summary measures. This did not happen using the other fitness functions (i.e., Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$).

The boxplots in Figure 3 confirm the results obtained in terms of summary measures. Indeed, the median of $\frac{Pred(25)}{MMRE}$ is more close to zero than those of MMRE, MEMRE, MdEMRE, and $\frac{Pred(25)}{MdMRE}$. On the other hand, the median of MdMRE and Pred(25) is very close to the one of $\frac{Pred(25)}{MMRE}$. Moreover, even if the box length and tails of MMRE and $\frac{Pred(25)}{MdMRE}$ are close to the ones of Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$ they have more outliers that are more far from the box than the ones of Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$.

The indications given by summary measures and boxplots are confirmed also by using statistical tests. In particular, we performed the Wilcoxon test to verify the following null hypothesis: "*the use of f_i as fitness function does not provide better results than using f_j*", where f_i and f_j are two experimented fitness functions. The results shown in Table VI reveal that the use of MdEMRE provided the worst

Table VI
RESULTS OF THE WILCOXON TESTS COMPARING FITNESS FUNCTIONS

$<$	MMRE	Pred(25)	MdMRE	MEMRE	MdEMRE	$\frac{Pred(25)}{MMRE}$	$\frac{Pred(25)}{MdMRE}$
MMRE	-	0.989	0.986	0.486	**0.032**	0.995	0.735
Pred(25)	**0.011**	-	0.286	**0.001**	**7.6e-5**	0.190	**0.000**
MdMRE	**0.014**	0.715	-	**0.001**	**9.9e-5**	0.201	**0.000**
MEMRE	0.516	0.999	0.999	-	**0.000**	0.984	0.708
MdEMRE	0.969	1	1	0.999	-	0.999	0.999
$\frac{Pred(25)}{MMRE}$	**0.005**	0.811	0.800	**0.016**	**0.001**	-	**0.010**
$\frac{Pred(25)}{MdMRE}$	0.996	0.990	0.999	0.293	**0.003**	0.990	-

accuracy, i.e., all the other fitness functions provided better results than it. Moreover, the use of Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$ provided statistically significant better estimates than the use of MMRE, MEMRE, and $\frac{Pred(25)}{MdMRE}$. This is especially interesting taking into account that previous studies on the use of Genetic Programming employed only MMRE as fitness function [8] [10] [16].

B. Comparison with other Effort Estimation Methods

In this section we report the results related to the second research goal and obtained by comparing the estimates provided by GP with the ones provided by Mean, Median, MSWR, and CBR. In particular, since the previous analysis revealed that GP performs better when Pred(25), MdMRE,

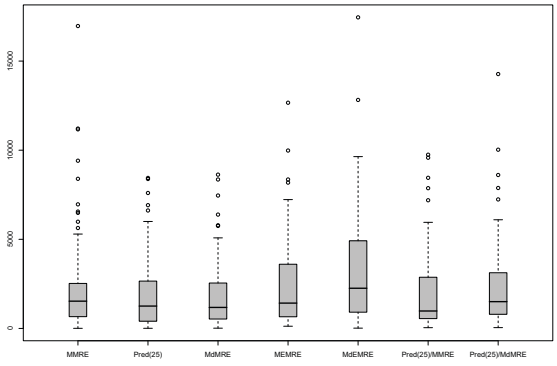


Figure 3. Boxplots of absolute residuals related to the application of the models constructed by GP with the analysed fitness functions on the test sets.

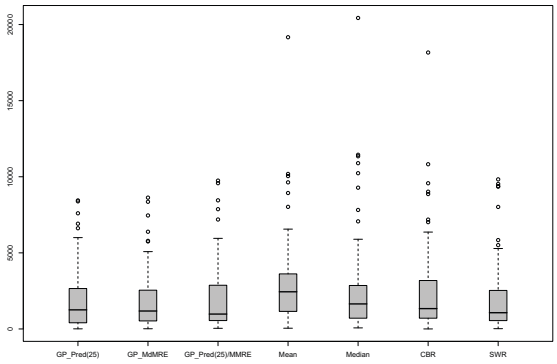


Figure 4. Boxplots of absolute residuals obtained with the analysed estimation methods.

and $\frac{Pred(25)}{MMRE}$ are used as fitness function, we exploited them (reported in the following as $GP_{Pred(25)}$, GP_{MdMRE} and, $GP_{\frac{Pred(25)}{MMRE}}$, respectively) for the comparison.

The analysis of summary measures (see Table VII) suggests that the estimations obtained with GP are better than those achieved by using Mean and Median of effort. The boxplots of absolute residuals shown in Figure 4 confirm these results. Indeed, the median of GP boxplots are more close to zero than the one of the boxplots of Median and Mean of effort. Furthermore, GP boxplots have less ouliers (and less far from their boxes) and their box length and tails are less skewed than those of the boxplots of Mean and Median of effort. Moreover, Table VIII shows the results of the Wilcoxon test to statistically compare the accuracy provided by GP with the accuracy provided by Mean and Median. This test revealed that the absolute residuals obtained with GP are significantly better than those obtained by using the Mean and Median of effort.

Concerning the comparison with MSWR and CBR,

Table VII
COMPARISON BASED ON SUMMARY MEASURES

Method	MMRE	Pred (25)	MdMRE	MEMRE	MdEMRE
$GP_{Pred(25)}$	0.68	0.43	0.33	0.38	0.31
GP_{MdMRE}	0.67	0.43	0.32	0.38	0.32
$GP_{\frac{Pred(25)}{MMRE}}$	0.64	0.39	0.33	0.44	0.34
Mean	1.21	0.22	0.54	0.62	0.51
Median	0.83	0.35	0.41	0.74	0.42
CBR	0.72	0.35	0.42	0.55	0.41
MSWR	0.62	0.40	0.36	0.47	0.38

Table VIII
COMPARISON BASED ON WILCOXON TESTS

$<$	$GP_{Pred(25)}$	GP_{MdMRE}	$GP_{\frac{Pred(25)}{MMRE}}$	Mean	Median	CBR	MSWR
$GP_{Pred(25)}$	-	0.286	0.190	**9.3e-6**	**0.001**	**0.007**	0.100
GP_{MdMRE}	0.715	-	0.201	**1.1e-5**	**0.002**	**0.008**	0.077
$GP_{\frac{Pred(25)}{MMRE}}$	0.811	0.800	-	**1.1e-5**	**0.001**	**0.021**	0.133
Mean	1	1	1	-	0.952	0.999	1
Median	0.998	0.998	0.999	0.050	-	0.976	0.998
CBR	0.993	0.992	0.979	**0.001**	**0.025**	-	0.963
MSWR	0.900	0.923	0.868	**2.3e-5**	**0.002**	**0.037**	-

the analysis of the summary measures (see Table VII) demonstrates that GP (i.e., $GP_{Pred(25)}$, GP_{MdMRE} and, $GP_{\frac{Pred(25)}{MMRE}}$) achieved better results than CBR and comparable results with MSWR. This result is also confirmed by the analysis of boxplots (see Figure 4). In particular, box length and tails of GP boxplots have a median more close to zero than CBR boxplot and very close to the median of MSWR boxplot, while the box length and tails of the GP boxplots are similar to the ones of MSWR and CBR boxplots. Furthermore, we can observe that outliers of the CBR boxplot are more far from its box than those of the other boxplots. As designed we also tested whether there was statistically significant difference between estimates obtained with GP and those obtained with MSWR and CBR. The results suggest that the absolute residuals obtained with GP are significantly less than those achieved with CBR (see Table VIII), while no statistically significant difference was found between the estimates provided by GP and MSWR.

C. Comparison with Burgess and Lefley's Case Study

Burgess and Lefley [8] also assessed the use of GP for estimating software development effort in a case study that exploited the Desharnais dataset [14]. The GP parameters they used are: a population of 1000 individuals, 500 generations, and 10 executions (see Table IX). They exploited only one fitness function designed to minimise MMRE and employed canonical genetic operators. They also used a tree-based representation for the solutions. However, differently from our proposal, during the evolutionary process trees with different depths can be produced. As for cross validation, they employed the hold-out which is the simplest kind of cross validation (i.e., one-fold), where the dataset is split into a training set used to build the estimation model and a test used to validate it. In particular a training set containing 63 observations and a test set containing 18 observations were considered[3]. Thus, they employed also the 4 observations we excluded from the analysis due to the presence of missing

[3]Note that this split is not publicly available.

Table IX
COMPARISON OF THE SETTING OF GP-BASED APPROACHES

	Burgess and Lefley [8]	Our approach
Population size	1000	70
Number of generations	500	¡=7000
Number of executions	10	10
Tree depth	Variable	Fixed
Evolutionary approach	Canonical	Canonical
Fitness function	MMRE	MMRE, Pred(25), MdMRE, MEMRE, MdEMRE $\frac{Pred(25)}{MMRE}$ $\frac{Pred(25)}{MdMRE}$

Table X
RESULTS ACHIEVED BY BURGESS AND LEFLEY [8]

	Method	MMRE	#MRE<0.25	Pred(25)
Burgess and Lefley [8]	Genetic Programming (GP)	0.45	4	0.23
	Linear LSR (LR)	0.46	10	0.56
	2 nearest neighbours (CBR_2)	1.62	8	0.44
	5 nearest neighbours (CBR_5)	1.68	8	0.44

values. However, hold-out procedure can be biased since the prediction performance may heavily depend on how the dataset is split (what are the data points in training set and in test set, respectively). Therefore, in our case study we used a 3-fold cross validation which less suffers of such bias. Moreover, differently from our work, they did not perform statistical tests to verify differences in the distribution of the absolute residuals or MRE values.

Table X shows the average results Burgess and Lefley achieved executing 10 runs of their GP based estimation method. We can observe that their approach obtained an MMRE equals to 0.45 and a Pred(25) equals to 0.23. Even if GP did not outperform LR (Linear Regression) the results encouraged other research in this field. They observed that the use of MMRE degraded the other accuracy measures and suggested that the use of different functions could improve the accuracy of estimations. By using the same dataset but with a different validation method (3-fold vs hold-out) and validation criteria our study has confirmed the observation and the intuition of Burgess and Lefley. Indeed, first of all we have confirmed that MMRE as fitness function is not the best choice, since it allows us to get better MMRE values but not an overall good prediction accuracy as we have shown analysing other summary measures (i.e. Pred, MdMRE, MEMRE, MdEMRE), boxplot of absolute residuals, and statistical significance tests. We have also shown that this behaviour is common to other summary measures investigated in our paper as fitness function (MdEMRE, MEMRE, $\frac{Pred(25)}{MdMRE}$). Moreover, we have identified some accuracy measures (i.e. Pred, MdMRE, and $\frac{Pred(25)}{MMRE}$) that can be more promising as fitness functions since they do not exhibit this problem. Furthermore, we have confirmed the intuition of Burgess and Lefley, since we have identified GP settings with estimation accuracy very close to the one achieved with MSWR and significantly better than those obtained with CBR. Note that we used a different application of linear regression (i.e., MSWR) that is considered more solid [20].

V. VALIDITY EVALUATION

It is widely recognised that several factors can bias the validity of empirical studies. In this section we discuss the validity of the empirical study based on three types of threats, namely *construct*, *conclusion*, and *external* validity.

As highlighted by Kitchenham *et al.* [29], to satisfy construct validity a study has "to establish correct operational measures for the concepts being studied". This means that the study should represent to what extent the predictor and response variables precisely measure the concepts they claim to measure [22]. Thus, the choice of the features and how to collect them represents the crucial aspects. We tried to mitigate such a threat by evaluating the employed estimation methods on a publicly available dataset [14]. Moreover, since the dataset is publicly available it has been previously used in many other empirical studies carried out to evaluate effort estimation methods, e.g., [4] [8] [15] [17].

Concerning the conclusion validity we carefully applied the statistical tests, verifying all the required assumptions. Moreover, we used a medium size dataset to mitigate the threats related to the number of observations composing the dataset. However, the employed dataset contains projects related to one context that might be characterised by some specific project and human factors, such as development process, developer experience, tools, technologies used, time, and budget constraints [30]. This represents an important external validity threat that can be mitigated only replicating the study taking into account data from other companies, thus getting a generalisation of the results.

VI. RELATED WORK

Besides the work of Burgess and Lefley [8] whose results have been reported and discussed in Section IV-C, some empirical investigations have been performed to assess the effectiveness of GP in estimating software development effort. In particular, GP was employed by Dolado [9] in order to automatically derive equations alternative to multiple linear regression. The aim was to compare the linear equations with those obtained automatically. GP was run a minimum of 15 times and each run had an initial population of 25 equations. Even if in each run the number of generations varied, the best results were obtained with three to five generations (as reported in the literature, usually more generations are used) and by using the Mean Square Error (MSE) [12] as fitness function. As dataset, 46 projects developed by academic students were exploited. It is worth noting that the main goal of Dolado work was not the assessment of GP but the validation of the component-based method for software sizing. However, he observed that GP provided similar or better values than regression equations.

Successively, Shepperd and Lefley [10] also assessed the effectiveness of GP and compared it with several estimation techniques such as LR, ANN (Artificial Neural Networks), and CBR. As for GP setting they applied the same choice

of Burgess and Lefley [8] while a different dataset was exploited. This dataset is refereed as "Finnish Dataset" and included 407 observations and 90 features, obtained from many organizations. After a data analysis, a training set of 149 observations and a test set of 15 observations were obtained and used in the empirical analysis. Even if the results revealed that there was not a method that provided better estimations than the others, GP performed consistently well. However, the authors observed that GP and ANN were harder to configure than LR and companies have to weight the complexity of these methods against the small increases in accuracy to decide whether to use it to estimate development effort [10].

An evolutionary computation method, named Grammar Guided Genetic Programming (*GGGP*), was proposed in [11] to fit models, with the aim of improving the estimation of the software development effort. Data of 423 software projects from *ISBSG* (http://www.isbsg.org.au) database were employed to build the estimation models using GGGP and LR. The fitness function was designed to minimize MSE, an initial population of 1000 was chosen, the maximum number of generations was 200 and the number of executions was 5. The results revealed that GPPP performed better than LR in terms of MMRE and Pred(25).

VII. Conclusions and Discussion

The choice of the fitness function represents one of the main critical design choices in the use of GP. This is especially true in the context of effort estimation since it should guide the search to get a model with good estimation accuracy. Unfortunately, not a unique criterion exists to measure such an accuracy and then to define the corresponding fitness function. Indeed, although the identification of "the" software estimation accuracy measure is still an open issue, the research community has agreed that the prediction model accuracy assessment should be based on the comparison of different summary measures (e.g., MMRE, MdMRE, Pred(25), MEMRE, and MdEMRE) as well as on the use of more sophisticated techniques (e.g., boxplots of absolute residuals, statistical tests).

Starting from the observation by Burgess and Lefley [8] that the use of MMRE (the most widely used evaluation criterion) as fitness function could degrade a lot of the other accuracy measures, we carried out an empirical study whose main goal was to analyse how the use of different fitness functions affects the accuracy of GP for effort estimation. We also compared the estimations achieved using GP with those achieved by widely used estimation techniques, such as MSWR and CBR. The experimentation was performed by the Desharnais dataset [14].

The main result achieved in the case study is that the choice of the accuracy measures as fitness function significantly influenced the accuracy of estimations obtained with GP. In particular, the results of Wilcoxon test revealed that the use of Pred(25), MdMRE, and $\frac{Pred(25)}{MMRE}$ as fitness function provided statistically significant better estimates than the use of MMRE, MEMRE, MdEMRE, and $\frac{Pred(25)}{MdMRE}$. This also confirms and extends the observation made by Burgess and Lefley [8] on the use of MMRE as objective function. Nevertheless, the analysis also demonstrated that there are some measures, e.g., Pred(25), that when used as objective functions are able to guide towards estimation model with better accuracy since do not degrade a lot of all the other summary measures.

The idea that several factors should be taken into account by the fitness function suggested us to investigate the use of two combinations of summary measures as fitness function (namely $\frac{Pred(25)}{MMRE}$ and $\frac{Pred(25)}{MdMRE}$). In our case study $\frac{Pred(25)}{MMRE}$ is one of the fitness functions that gave the best results. Nevertheless, the combination only performs better than MMRE but it is not better than Pred(25). On the contrary $\frac{Pred(25)}{MdMRE}$ did not provide good results, maybe it was not able to capture complementary accuracy aspects. In any case, the use of more than one summary measure in the definition of the fitness function deserves to be further investigated by employing more sophisticated combinations and multi-objective optimisation approaches (e.g., the ones based on Pareto optimality) [31]. This will be part of our agenda of future work. Moreover, an empirical analysis could be carried out to verify whether acting on others GP design choice (e.g., population size, generation number, crossover and mutation rates) influences the GP performance aiming at choosing the most appropriate GP setting, as done in [32].

The case study also highlighted that GP performed better than widely used estimation methods. In particular, the results showed that GP provided better results than CBR and MSWR in terms of summary measures. Moreover, GP significantly outperformed CBR.

It is clear that the results presented in the paper should be verified with other datasets. Moreover, the behaviour of other search-based algorithms could be investigated [31], such as Simulated Annealing and Tabu Search, only employed in a preliminary case study [16].

References

[1] L. Briand, K. El. Emam, D. Surmann, I. Wiekzorek, and K. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *Proceedings of International Conference on Software Engineering*. IEEE press, 1999, pp. 313–322.

[2] L. Briand, T. Langley, and I. Wiekzorek, "A replicated assessment and comparison of common software cost modeling techniques," in *Proceedings of International Conference on Software Engineering*. IEEE press, 2000, pp. 377–386.

[3] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais, "A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models," *Journal of Systems and Software*, vol. 39, no. 3, pp. 281–289, 1997.

[4] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transaction on Software Engineering*, vol. 23, no. 11, pp. 736–743, 2000.

[5] L. C. Briand and I. Wieczorek, "Software resource estimation," *Encyclopedia of Software Engineering*, pp. 1160–1196, 2002.

[6] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[7] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, K. Rees, and M. Roper, "Reformulating software engineering as a search problem," *IEE ProceedingsSoftware*, vol. 150, p. 2003, 2003.

[8] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? a comparative evaluation," *Information and Software Technology*, vol. 43, no. 14, pp. 863–873, 2001.

[9] J. J. Dolado, "A validation of the component-based method for software size estimation," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 1006–1021, 2000.

[10] M. Lefley and M. J. Shepperd, "Using genetic programming to improve software effort estimation based on general data sets," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2003, pp. 2477–2487.

[11] Y. Shan, R. I. Mckay, C. J. Lokan, and D. L. Essam, "Software project effort estimation using genetic programming," in *Proceedings of International Conference on Communications Circuits and Systems*. IEEE press, 2002, pp. 1108–1112.

[12] D. Conte, H. Dunsmore, and V. Shen, *Software engineering metrics and models*. The Benjamin/Cummings Publishing Company, Inc., 1986.

[13] B. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, "What accuracy statistics really measure," *IEE Proceedings Software*, vol. 148, no. 3, pp. 81–85, 2001.

[14] J. M. Desharnais, "Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction," Ph.D. dissertation, Unpublished Masters Thesis, University of Montreal, 1989.

[15] G. Kadoda and M. Shepperd, "Using simulation to evaluate predictions techniques," in *Proceedings of International Software Metrics Symposium*. IEEE press, 2001, pp. 349–358.

[16] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Using tabu search to estimate software development effort," in *Proceedings of Mensura 2009*. Lecture Notes in Computer Science 5891 Springer, 2009, pp. 307–320.

[17] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of International Conference on Software Engineering*. IEEE press, 1996, pp. 170–178.

[18] J. R. Koza, *Genetic Programming*. MIT Press, 1992.

[19] E. Mendes and B. Kitchenham, "A comparison of cross-company and within-company effort estimation models for web applications," in *Proceedings of Conference on Evaluation and Assessment in Software Engineering*, 2004, pp. 47–55.

[20] E. Mendes and B.Kitchenham, "Further comparison of cross-company and within-company effort estimation models for web applications," in *Proceedings of International Software Metrics Symposium*. IEEE press, 2004, pp. 348–357.

[21] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Journal of Systems and Software*, vol. 48, no. 11, pp. 1034–1045, 2006.

[22] E. Mendes, S. Counsell, N. Mosley, C. Triggs, and I. Watson, "A comparative study of cost estimation models for web hypermedia applications," *Empirical Software Engineering*, vol. 8, no. 23, pp. 163–196, 2003.

[23] H. G. Cobb and J. J. Grefenstette, "Proceedings of the 5th international conference on genetic algorithms, icga, 1993," in *ICGA*. Morgan Kaufmann, 1993, pp. 523–530.

[24] B. Kitchenham, T. Foss, E. Stensrud, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transaction on Software Engineering*, vol. 29, no. 11, pp. 985–995, 2003.

[25] I. Myrtveit, M. Shepperd, and E. Stensrud, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380–39, 2005.

[26] E. Stensrud and I. Myrtveit, "Human performance estimating with analogy and regression models: an empirical validation," in *Proceedings of International Software Metrics Symposium*. IEEE press, 1996, pp. 205–.

[27] P. Royston, "An extension of Shapiro and Wilk's W test for normality to large samples," *Applied Statistics*, vol. 31, no. 2, pp. 115–124, 1982.

[28] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Lawrence Earlbaum Associates, 1988.

[29] B. Kitchenham, L. Pickard, and S. Pfleeger, "Case studies for method and tool evaluation," *IEEE Software*, vol. 12, no. 4, pp. 52–62, 1995.

[30] L. C. Briand and J. Wüst, "Modeling development effort in object-oriented systems using design properties," *IEEE Transaction on Software Engineering*, vol. 27, no. 11, pp. 963–986, 2001.

[31] M. Harman, "The current state and future of search based software engineering," in *Proceedings of Future of Software Engineering - International Conference on Software Engineering*, 2007, pp. 342–357.

[32] V. Garousi, "Empirical analysis of a genetic algorithm-based stress test technique," in *Proceedings of the 10th Conference on Genetic and Evolutionary Computation*. ACM press, 2008, pp. 1743–1750.

AUSTIN: A tool for Search Based Software Testing for the C Language and its Evaluation on Deployed Automotive Systems

Kiran Lakhotia
King's College London, CREST,
Strand, London, WC2R 2LS, U.K.
kiran.lakhotia@kcl.ac.uk

Mark Harman
King's College London, CREST,
Strand, London, WC2R 2LS, U.K.
mark.harman@kcl.ac.uk

Hamilton Gross
Berner & Mattner Systemtechnik GmbH,
Gutenbergstr. 15, D–10587 Berlin, Germany
hamilton.gross@berner-mattner.com

***Abstract*—Despite the large number of publications on Search–Based Software Testing (SBST), there remain few publicly available tools. This paper introduces AUSTIN, a publicly available SBST tool for the C language. The paper validates the tool with an empirical study of its effectiveness and efficiency in achieving branch coverage compared to random testing and the Evolutionary Testing Framework (ETF), which is used in-house by Daimler and others for Evolutionary Testing. The programs used in the study consist of eight non–trivial, real-world C functions drawn from three embedded automotive software modules. For the majority of the functions, AUSTIN is at least as effective (in terms of achieved branch coverage) as the ETF, and is considerably more efficient.**

***Keywords*-Software Testing; SBSE; SBST**

I. INTRODUCTION:

Search–Based Software Testing (SBST) was the first Software Engineering problem to be attacked using optimization [1] and also the first to which a search–based optimization technique was applied [2]. Recent years have witnessed a dramatic rise in the growth of work on SBST and in particular on techniques for generating test data that meets structural coverage criteria. Yet, despite an increasing interest in SBST and, in particular, in structural coverage using SBST, there remains a lack of publicly available tools that provide researchers with facilities to perform search–based structural testing.

This paper introduces such a tool, AUSTIN, and reports our experience with it. AUSTIN uses a variant of Korel's [3] 'Alternating Variable Method' (AVM) and augments it with techniques adapted from recent work on directed adaptive random testing and dynamic symbolic execution [4], [5], [6], [7]. It can handle a large subset of C, though there are some limitations. Most notably AUSTIN cannot generate meaningful inputs for strings, void and function pointers, as well as union constructs. Despite these limitations, AUSTIN has been applied 'out of the box' to real industrial code from the automotive industry (see Section V) as well as a number of open source programs [8].

This paper presents an empirical study in which AUSTIN has been compared to an Evolutionary Testing Framework (ETF), which was developed as part of the EvoTest project [9]. The Framework represents a state–of–the–art evolutionary testing system and has been applied to case studies from the automotive and communications industry. Three case studies from the automotive industry, provided by Berner & Mattner Systemtechnik GmbH, formed the benchmark which AUSTIN was compared against for effectiveness and efficiency when generating branch adequate test data.

Automotive code was chosen as the benchmark because the automotive industry is subject to testing standards that mandate structural coverage criteria [10] and so the developers of production code for automotive systems are a natural target for automated test data generation techniques, such as those provided by AUSTIN.

The rest of the paper is organised as follows: Section II provides background information on the field of search–based testing and gives an overview of related work. Section III introduces AUSTIN and describes the different techniques implemented, whilst Section IV provides details about the Evolutionary Testing Framework against which AUSTIN has been compared. The empirical study used to evaluate AUSTIN alongside the hypotheses tested, evaluation and threats to validity are presented in Section V. Section VI concludes.

II. BACKGROUND

Test data generation is a natural choice for Search–Based Software Engineering (SBSE) researchers because the search space is clearly defined (it is the space of inputs to the program) and tools often provide existing infrastructures for representing candidate inputs and for instrumenting and recording their effect. Similarly, the test adequacy criterion is usually well defined and is also widely accepted as a metric worthy of study by the testing community, making it an excellent candidate for a fitness function [11]. The role of the fitness function is to return a value that indicates how 'good' a point in a search space (*i.e.* an input vector) is compared to the best point (*i.e.* the required test data): the global optimum. For example, if a condition `a == b` must be executed as true, a possible objective function is $|a - b|$. When this function is 0, the desired input values have been

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.21

found. Different branch functions exist for various relational operators in predicates [3].

McMinn [12] provides a detailed survey of work on SBST up to approximately 2004. It shows that the most popular search technique applied to structural testing problems has been the Genetic Algorithm. However, other search–based algorithms have also been applied, including parallel Evolutionary Algorithms [13], Evolution Strategies [14], Estimation of Distribution Algorithms [15], Scatter Search [16], [17], Particle Swarm Optimization [18], [19] and Tabu Search [20].

Due to the large body of work on SBST, Ali et al. [21] performed a systematic review of the literature in order to asses the quality and adequacy of empirical studies used in evaluating SBST techniques. One of their key findings is that empirical studies in SBST need to include more statistical analysis, in the form of hypothesis testing, in order to account for the randomness in any meta–heuristic algorithm.

Outside the search–based testing community there has been a growing number of publicly available tools for structural testing problems, most notably from the field of Dynamic Symbolic Execution (DSE) [4], [5], [6], [7]. DSE combines symbolic [22] and concrete execution. Concrete execution drives the symbolic exploration of a program, and runtime values can be used to simplify path constraints produced by symbolic execution to make them more amenable to constraint solving. For example, assume two inputs `a` and `b` have the values `38` and `100` respectively, and that a path condition of interest is of the form `(int)log(a) == b`. Further assume that a particular constraint solver cannot handle the call to the `log` function. Now suppose during concrete execution, the expression `(int)log(38)` evaluated to 3. DSE can then simplify the path condition to `3 == b`. The constraint solver can now be used to provide a value for `b` which satisfies the constraint. A more detailed treatment of this approach can be found in the work of Godefroid et al. [4].

AUSTIN draws together strands of research on search–based testing for structural coverage and DSE so that it can generate branch adequate test data for integers, floating point and pointer type inputs. Currently AUSTIN only addresses one small, but important part of testing: it generates input values that reach different parts of a program. Whether these inputs reveal any faults is still left for the user to decide.

III. AUSTIN

AUgmented Search–based TestINg (AUSTIN) is a structural test data generation tool which combines a simple hill climber for integer and floating point type inputs with a set of constraint solving rules for pointer type inputs. It has been designed as a unit testing tool for C programs. AUSTIN considers a unit to be a function under test and all the functions reachable from within that function.

Algorithm 1 High level description of the AUSTIN-AVM

```
currentSolution := random
bestSolution := currentSolution
doLocalRestart := true
while not reached stopping criterion do
  if solve pointer constraint then
    if solvePC(currentSolution) = NULL then
      currentSolution := random
    end if
  else if trapped at local optimum then
    if doLocalRestart then
      for i := 0 to currentSolution.length do
        localRestart(currentSolution[i])
      end for
      doLocalRestart := false
    else
      currentSolution := random
      doLocalRestart := true
    end if
  else
    improvement := exploratoryMove(currentSolution)
    restartExploration := false
    while improvement do
      bestSolution := currentSolution
      if reached stopping criterion then
        return bestSolution
      end if
      improvement := patternMove(currentSolution)
      restartExploration := true
    end while
    if restartExploration then
      reset search parameters
    end if
  end if
end while
return bestSolution
```

AUSTIN can be used to generate a set of input data for a given function which achieve (some level of) branch coverage for that function. During the test data generation process, AUSTIN does not attempt to execute specific paths through a function in order to cover a target branch; the path taken up to a branch is an emergent property of the search process. The search is guided by an objective function that was introduced by Wegener *et al.* [23] for the Daimler Evolutionary Testing System. It evaluates an input against a target branch using two metrics: the *approach level* and the *branch distance*. The approach level records how many of the target branch's control dependent nodes were not executed by a particular input. The fewer control dependent nodes executed, the 'further away' the input is from executing the branch in control flow terms. The branch

Algorithm 2 High level description of solvePC
Inputs: Equivalence graph of symbolic variables EG and candidate solution $currentSolution$

Compute path condition pc for currentSolution
Compute the approximate path condition pc' from pc by dropping all constraints over arithmetic types from pc
Trim pc' by removing all non critical branching nodes
Invert the binary operator ($\in \{=, \neq\}$) of the last constraint in pc'
for all constraints c_i in pc' **do**
  $left :=$ get lhs of c_i
  $right :=$ get rhs of c_i
  Get node n_{left} from EG which contains $left$ or create a new node n_{left} if no such node exists
  Get node n_{right} from EG which contains $right$ or create a new node n_{right} if no such node exists
  if operator op_i in c_i is $=$ **then**
    if n_{left} has an edge with n_{right} in EG **then**
      return NULL {infeasible}
    else
      Merge nodes n_{left} and n_{right}
      Update EG
    end if
  else if operator op_i in c_i is $\neq$ **then**
    if $n_{left} = n_{right}$ **then**
      return NULL {infeasible}
    else
      Add edge between n_{left} and n_{right}
      if $n_{left} \neq$ null node **then**
        Add edge between n_{left} and null node
      end if
      if $n_{right} \neq$ null node **then**
        Add edge between n_{right} and null node
      end if
      Update EG
    end if
  end if
end for
for all nodes n_i in EG **do**
  if n_i has no edge to null node **then**
    $m :=$ NULL
  else
    if n_i represents the address A of a variable **then**
      $m := A$
    else
      $m := malloc$
    end if
  end if
  for all symbolic variables s_i in n_i **do**
    Update corresponding element for s_i in currentSolution with m
  end for
end for
return currentSolution

distance is computed using the condition of the decision statement at which the flow of control diverted away from the current 'target' branch. Taking the true branch from node (2) in Figure 1 as an example, if the false branch is taken at node (2), the branch distance is computed using `|one->key - 10|`. The branch distance is normalised and added to the approach level.

Similar to DSE [4], [6], [7], AUSTIN also instruments the program under test to symbolically execute the program along the concrete path of execution for a particular input vector. Collecting constraints over input variables via symbolic execution serves to aid AUSTIN in solving constraints over memory locations, denoted by pointer inputs to a function. Consider the example given in Figure 1 and suppose execution follows the false branch at node (1). AUSTIN will use a custom procedure to solve the constraint over the pointer input `one`. On the other hand, if the false branch is taken at node (2), the AVM is used to satisfy the condition at node (2).

A. AVM

The AVM used in AUSTIN works by continuously changing each arithmetic type input in isolation. First, a vector is constructed containing the arithmetic type inputs (*e.g.*, integers, floats) to the function under test. All variables in this vector are initialised with random values. Then, so called *exploratory moves* are made for each element in turn. These consist of adding or subtracting a delta from the value of an element. For integral types the delta starts off at 1, *i.e.*, the smallest increment (decrement). When a change leads to an improved fitness value, the search tries to accelerate towards an optimum by increasing the size of the neighbourhood move with every step. These are known as *pattern moves*. The formula used to calculate the delta added or subtracted from an element is: $\delta = 2^{it} * dir * 10^{-prec_i}$, where it is the repeat iteration of the current move, dir either -1 or 1, and $prec_i$ the precision of the i^{th} input variable. The precision only applies to floating point variables, as will be described in Section III-B (*i.e.*, it is 0 for integral types).

Whenever a delta is assigned to a variable, AUSTIN checks for a possible over- or underflow. For integral types this is done with a set of custom macros that use `gcc`'s `typeof` operator [24]. For floating point operations on the other hand, AUSTIN does not check for over- or underflow errors *per se*. Instead, it watches for $\pm$ INF or $\pm$ NaN values. Whenever a potential move leads to an overflow (underflow) of integral types or results in an input taking on the value $\pm$ INF or $\pm$ NaN, AUSTIN discards the move as invalid and explores the next neighbour. As a consequence, code which explicitly checks for $\pm$ INF or $\pm$ NaN cannot be covered by AUSTIN. So far this has not been observed in practice. To handle possible overflow (underflow) in bit fields, AUSTIN sets a lower bound for signed bit fields at $-(2^l/2)$ (0 for unsigned bit fields), and an upper bound at $(2^l/2)-1$ (2^l-1

Node id	Example Function

	`typedef struct item {`
	`  int key;`
	`};`
	`void testme(item* one) {`

(1)	`  if ( one != null ) {`
(2)	`  if ( one->key == 10 )` `  // target`
	`}`

Figure 1. Example C code used for demonstrating how AUSTIN combines custom constraint solving rules for pointer inputs with an AVM for other inputs to a function under test. The goal is to find an input which satisfies the condition at node (2).

for unsigned bit fields), where l is the length of the bit field. A user can also specify custom bounds for every variable. Again, updates to an input which violate these bounds are discarded as infeasible, forcing the search to move on. The main motivation for including such 'bounds checking' in AUSTIN is to save wasteful moves.

Once no further improvements can be found for an input, the search continues exploring the next element in the vector, recommencing with the first element if necessary, until no changes to an input lead to further improvements. At this point the search restarts at another randomly chosen location in the search space. This is known as a *random restart* strategy and is designed to overcome local optima and enable the search to explore a wider region of the input domain for the function under test.

B. Floating Point Variables

As mentioned in the previous section, each floating point variable has an associated precision. For example, setting the precision $prec_i$ of the i^{th} input variable to 1 limits the smallest possible move for that variable to ± 0.1. Increasing the precision to 2 limits the smallest possible move to ± 0.01, and so forth. In practice, the precision of floating point numbers is limited. Double precision types have about 16 decimal digits, and single precision types about 7 decimal digits of precision.

Initially, the precision variable $prec_i$ of each input is set to 0, and whenever the AVM is stuck at a local optimum it tries to increase this value. To avoid redundant changes to $prec_i$, the AVM checks if adding the delta 10^{-prec_i} to a variable changes its value, or, $prec_i$ is less than 7, or 16 for single or double precision type inputs. If the precision of floating point variables cannot be increased any further, or, no exploratory move with an increased precision results in an improved fitness value, the AVM resorts to a random restart.

C. Random Restarts

The AVM performs two types of random restarts in order to escape local optima. The first type is a global restart while the second is a local restart (see Algorithm 1). In a global restart, all input variables are assigned new random values. This is likely to place the starting point for the next hill climb far away from the local optimum where the search got stuck. While this is desirable in many cases, it is not an ideal strategy when a global optimum is surrounded by many local optima. The chances are the search will just get stuck at the same local optimum again. Therefore the AVM also uses a local restart, which is designed to stay in the vicinity of the current search space while still being able to escape from a local optimum.

In a local restart, a random number r (between 0 and 1) is created for each input variable. This number is then 'scaled' by the formula $10^{-prec_i} * r$, where $prec_i$ is the current precision of the i^{th} input variable. This 'scaled' random number is then added to the existing value of the input variable. If such a local restart does not enable the search to make further progress, a global restart is performed. Thus, the AVM alternates between local and global restarts.

D. Pointer Inputs

We will now describe how the AVM has been extended to add automatic support for pointers and dynamic data structures. The constraint solving procedure for pointer inputs has been adapted from the DSE tool CUTE [6].

A high level description of the algorithm for solving pointer constraints is shown in Algorithm 2. It is an improved version of the approach introduced in our earlier work [25], which did not include feasibility checks or use an equivalence graph. Algorithm 2 first constructs a symbolic path condition pc, describing the path taken by the concrete execution. At this stage pc may contain constraints over both arithmetic type, as well as pointer type inputs. AUSTIN thus constructs a sub–path condition, pc', in which all constraints over arithmetic types are dropped. This includes constraints which contain a pointer dereference to a primitive type. pc' is further simplified by removing all constraints which originated from non–critical branching nodes with respect to the current target branch. Furthermore, AUSTIN uses the CIL [26] framework to ensure that the remaining constraints over memory locations are of the form $x = y$ and $x \neq y$, where both x or y may be the constant $null$ or a symbolic variable denoting a pointer input.

The path conditions pc and pc' describe the flow of execution taken by a concrete input vector, which took an infeasible path with respect to the current target branch. To

generate input values which take the execution 'closer' to the target branch in terms of the control flow graph, the binary comparison operator (*i.e.* $=, \neq$) of the last constraint in pc' must be inverted.

From this updated pc' AUSTIN generates an equivalence graph of symbolic variables, which is used to solve pointer constraints. The equivalence relationship between symbolic variables is defined by the '=' operators in pc'. The nodes of the graph represent abstract pointer locations, with node labels representing the set of symbolic variables which point to those locations. Edges between nodes represent inequalities.

The graph is built up incrementally as the search proceeds (*i.e.* with every invocation of the `solvePC` procedure), and always contains a special node to represent the constant $null$. For each constraint c_i in pc', the symbolic variables involved in c_i are extracted. Note that because every symbolic variable is also mapped to a concrete variable, runtime information from concrete executions can be used to resolve aliases between symbolic variables. AUSTIN then checks if the symbolic variables are already contained within the nodes of the equivalence graph. If they are not, a new node for each 'missing' symbolic variable is added to the graph.

Given the node(s) representing the symbolic variables in c_i, AUSTIN checks for satisfiability of c_i. If the symbolic variables in c_i all belong to the same node, and the binary operator in c_i denotes an inequality, the constraint is infeasible. Similarly, if the symbolic variables belong to different nodes connected by an edge, and the binary operator in c_i denotes an equality, the constraint is also infeasible. Given an infeasible constraint, AUSTIN is forced to perform a global random restart, with the hope of traversing a different path through the program. The expectation is that a new path will result in a solvable pc'.

For each feasible constraint in pc', AUSTIN updates the equivalence graph by either adding nodes, adding edges, or merging (unconnected) nodes. Whenever an edge is added between two nodes where neither node labels contain the constant $null$, for example to capture the constraint $x \neq y$, an edge is added from each of the nodes to the node for $null$.

The final step of the algorithm is to derive concrete pointer inputs from the equivalence graph. For every node n in the graph AUSTIN checks if it has an edge to the node for the constant $null$. If no edge exists, all concrete inputs represented by the symbolic variables in n are assigned $null$. Otherwise AUSTIN does the following: if a node n represents the address of another symbolic variable s, all concrete pointer inputs represented by the labels of n are assigned the address of the concrete variable represented by s. Otherwise, a new memory location is created via `malloc`, and each concrete pointer input represented by the labels of n are assigned that memory location.

IV. ETF

The ETF was developed as part of the multidisciplinary European Union–funded research project EvoTest [9] (IST-33472), applying evolutionary algorithms to the problem of testing software systems. It supports both black–box and white–box testing and represents the state–of–the–art for automated evolutionary structural testing. The framework is specifically targeted for use within industry, with much effort spent on scalability, usability and interface design. It is provided as an Eclipse plug-in, and its white–box testing component is capable of generating test cases for single ANSI C functions. A full description of the system is beyond the scope of this document and the interested reader is directed towards the EvoTest web page located at `www.evotest.eu`.

At its core, the ETF contains a user configurable evolutionary engine, which has been integrated from the GUIDE [27] project. The framework also implements a subset of the approach introduced by Prutkina and Windisch [28] to handle pointers and data structures. It maintains different pools of variables, which are used as the target of pointers and whose values are optimized by the evolutionary search. Each pool contains a subset of global variables and formal parameters of the function under test, and all variables in a given pool are of the same type. In addition, for parameters denoting a pointer to a primitive type or data structure, the ETF creates a *temporary* variable whose type matches the target type of the pointer. These temporary variables are also added to the pools.

Each variable in a pool is assigned an index in the range $0, ..., n-1$, where n is the size of its pool (*i.e.*, the number of variables in that pool). The individual (chromosome) describing a pointer input to the function under test contains two fields; one denoting an index and the other a value. The index is used to select a variable from the (correct) pool whose address is used as the target for the pointer input. Note that an index may be negative to denote the constant $null$. If the index corresponds to one of the temporary variables, the value field is used to instantiate that variable. In this way the ETF is able to generate pointers initialised to $null$, pointers to primitive types and pointers to simple data structures (*i.e.* not containing any pointer members). However the approach does not extend to pointers to recursive types, such as lists, trees and graphs.

V. EMPIRICAL STUDY

The objective of the empirical study was to investigate the effectiveness and efficiency of AUSTIN when compared to a state–of–the art evolutionary testing system (the ETF). The study consisted of 8 C functions that are summarised in Table I. They were taken from three embedded software modules and had been selected by Berner & Mattner Systemtechnik GmbH to form part of the evaluation of the ETF within the EvoTest [9] project. The functions had been

Table I
CASE STUDIES. LOC REFERS TO THE TOTAL PREPROCESSED LINES OF C SOURCE CODE CONTAINED WITHIN THE CASE STUDIES. THE LOC HAVE BEEN CALCULATED USING THE CCCC TOOL [29] IN ITS DEFAULT SETTING.

Case Study	LOC	Functions Tested	Software Module
B	18,200	02, 03, 06	Adaptive headlight control
C	7,449	07, 08, 11	Door lock control
D	8,811	12, 15	Electric window control

Table II
TEST SUBJECTS. THE LOC HAVE BEEN CALCULATED USING THE CCCC TOOL [29] IN ITS DEFAULT SETTING. THE NUMBER OF INPUT VARIABLES COUNTS THE NUMBER OF INDEPENDENT INPUT VARIABLES TO THE FUNCTION, *i.e.*, THE MEMBER VARIABLES OF DATA STRUCTURES ARE ALL COUNTED INDIVIDUALLY.

Obfuscated Function Name	LOC	Branches	Nesting Level	# Inputs	Pointer Inputs
02	919	420	14	80	no
03	259	142	12	38	no
06	58	36	6	14	no
07	85	110	11	27	yes
08	99	76	7	29	yes
11	199	129	4	15	yes
12	67	32	9	3	no
15	272	216	4	28	yes

chosen to provide a representative sample of real world automotive code, with particular attention paid to the number of branches and nesting level. Table II gives a breakdown of relevant metrics for the selected functions.

Effectiveness of AUSTIN

In order to investigate the effectiveness of AUSTIN compared to the ETF we formulated the following null and alternate hypotheses:

H_0 : *AUSTIN is as effective as the ETF in achieving branch coverage.*
H_A : *AUSTIN is more effective than the ETF in achieving branch coverage.*

Efficiency of AUSTIN

Alongside coverage, efficiency is also of paramount importance especially in an industrial setting. To compare the efficiency of AUSTIN (in terms of fitness evaluations) to the ETF, we formulated these null and alternative hypotheses:

H_0 : *AUSTIN is equally as efficient as the ETF in achieving branch coverage of a function.*
H_A : *AUSTIN is more efficient than the ETF in achieving branch coverage of a function.*

A. Experimental Setup

The data for the experiments with the ETF on the functions listed in Table II had already been collected for the evaluation phase of the EvoTest project [9]. This section serves to describe how the ETF had been configured and how AUSTIN was adapted to ensure as fair a comparison as possible between AUSTIN and the ETF.

Every branch in the function under test was treated as a goal for both the ETF and AUSTIN. The order in which branches are attempted differs between the two tools. AUSTIN attempts to cover branches in level-order of the Control Flow Graph (CFG), starting from the exit node, while the ETF attempts branches in level-order of the CFG starting from the entry node. In both tools, branches that are covered serendipitously while attempting a goal are removed from the list of goals. The fitness budget for each tool was set to $10,000$ evaluations per branch.

The ETF supports a variety of search algorithms. For the purpose of this study, the ETF was configured to use a Genetic Algorithm (GA) whose parameters were manually tuned to provide a good set which was used for all eight functions. The GA was set up to use a population size of 200, deploy strong elitism as its selection strategy, use a mutation rate of 1% and a crossover rate of 100%.

The ETF also provides the option to reduce the size of the search space for the GA by restricting the bounds of each input variable, or even completely excluding variables from the search. Reducing the size of the input domain will improve the efficiency of search–based testing [30]. The input domain reduction in the ETF is a manual process and was carried out by members of the EvoTest project for each of the eight functions. AUSTIN was configured to apply the same input domain reduction in order to ensure a fair comparison. Finally, due to the stochastic nature of the algorithms used in both tools, each tool was applied 30 times to each function.

B. Evaluation

Effectiveness of AUSTIN. Figure 2 shows the level of coverage achieved by both the ETF and AUSTIN with error bars in each column indicating the standard error of the mean. The results provide evidence to support the claim that AUSTIN can be equally effective in achieving branch coverage than the more complex search algorithm used as part of the ETF. In order to test the first hypothesis, a test for statistical significance was performed to compare the coverage achieved by each tool for each function. A two–tailed test was chosen such that *reductions* in the branch coverage achieved by AUSTIN compared with the ETF could also be tested. Since the samples were often distinctly un–normally distributed and possessed heterogeneous variances and skew, the samples were first rank–transformed as recommended by Ruxton [31]. Then a two–tailed t–test for unequal variances with ($p \leq 0.05$) was carried out on the ranked samples. The two–tailed t-test was used instead of the Wilcoxon–Mann–Whitney test because the latter is sensitive to differences in the shape and variance of the distributions.

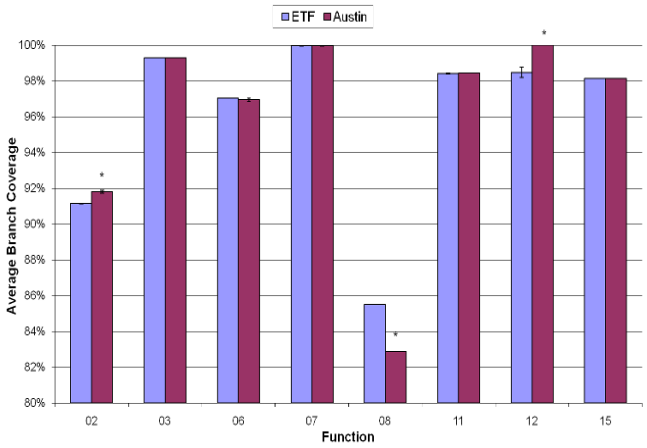


Figure 2. Average branch coverage of the ETF versus AUSTIN. The $y - axis$ shows the coverage achieved by each tool in percent, for each of the functions shown on the $x - axis$. The error bars show the standard error of the mean. Bars with a $*$ on top denote a statistically significant difference in the mean coverage ($p \leq 0.05$).

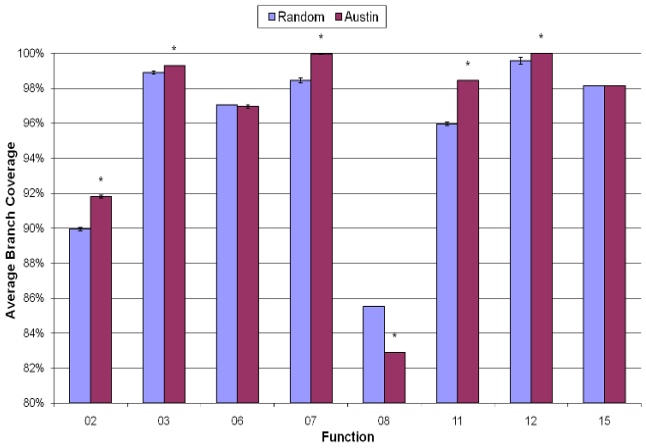


Figure 3. Average branch coverage of random search versus AUSTIN. The $y - axis$ shows the coverage achieved by each tool in percent, for each of the functions shown on the $x - axis$. The error bars show the standard error of the mean. Bars with a $*$ on top denote a statistically significant difference in the mean coverage ($p \leq 0.05$).

As shown in Figure 2 and detailed in Table III, AUSTIN delivered a statistically significant increase in coverage for functions 02 and 12. For function 08 AUSTIN achieved a statistically significant lower branch coverage than ETF (82.9% vs. 85.5%, $t(58) = \text{Inf}$). For all other functions, the null–hypothesis that the coverage achieved by the two tools comes from the same population was not rejected.

Function 08 is interesting because it is the only function for which AUSTIN performs significantly worse than the ETF. Therefore the results were analysed in more detail. The first point of interest was the constant number of fitness evaluations AUSTIN used during the 30 runs of this function. This can only occur in one of two cases: 1) AUSTIN is able to find a solution for each target branch from its initial starting point, and the starting points are all equidistant from the global optima; 2) for all targets which require AUSTIN to perform a random restart, it fails to find a solution, *i.e.*, the random restart has no effect on the success of AUSTIN. In this case it will continue until its fixed limit of fitness evaluations has been reached. For function 08 the latter case was true.

Analysing AUSTIN's coverage for function 08 revealed that it was unable to cover thirteen branches. These branches were guarded by a 'hard to cover' condition. Manual analysis showed that the difficult condition becomes feasible when traversing only two out of 63 branches prior to it. The other 61 branches lead to a 'killing' assignment to the input variable, whose value is checked in the difficult guarding condition. The paths which contain one of the two branches, which make the difficult condition feasible, are themselves hard to cover. To check if AUSTIN's failure was due to the inherent difficulty of the problem or, because not enough resources had been allocated, we repeated 30 runs for AUSTIN for function 08, this time without any input domain reduction, and a fitness budget of $100,000$ evaluations per branch. The results show that, given this larger fitness budget, AUSTIN is on average able to cover 97.60% of the branches. This is a marked increase from the average coverage of 82.89% shown in Figure 2. We could not repeat the experiments for the ETF with the extended fitness budget of $100,000$ evaluations per branch, because the fitness budget of $10,000$ evaluations per branch is currently hard coded in the ETF, and we do not have access to its source code. Therefore it is not possible to say how the ETF would have performed given a larger fitness budget.

Efficiency of AUSTIN. Figure 4 shows the average number of fitness evaluations used by both ETF and AUSTIN when trying to achieve coverage of each function. In order to test the second hypothesis, a two–tailed test for statistical significance was performed to compare the mean number of fitness evaluations used by each tool to cover each function. Since the difference in achieved coverage between the two tools was generally very small, it was neglected when comparing their efficiency. A two–tailed test was carried out to also test for functions in which AUSTIN's efficiency was worse than that of the ETF. The distributions of the samples were sufficiently normal (as determined by bootstrap re–sampling each sample–pair over 1000 iterations and visual inspection of the resulting distribution of mean values) to proceed with a two–tailed t–test for unequal variances on the raw un–ranked samples ($p \leq 0.05$).

As shown in Figure 4, AUSTIN delivered a statistically significant increase in efficiency compared with the ETF for functions 03, 06, 07, 11, 12, 15. For functions 02 and 08, AUSTIN used a statistically significant larger number of

Table III
SUMMARY OF FUNCTIONS WITH A STATISTICALLY SIGNIFICANT DIFFERENCE IN THE BRANCH COVERAGE ACHIEVED BY AUSTIN AND THE ETF. THE COLUMNS STDDEV INDICATE THE STANDARD DEVIATION FROM THE MEAN FOR ETF AND AUSTIN. THE T VALUE COLUMN SHOWS THE DEGREES OF FREEDOM (VALUE IN BRACKETS) AND THE RESULT OF THE T–TEST. A P VALUE OF LESS THAN 0.05 MEANS THERE IS A STATISTICALLY SIGNIFICANT DIFFERENCE IN THE MEAN COVERAGE BETWEEN ETF AND AUSTIN.

Function	Coverage ETF (%)	StdDev (%)	Coverage AUSTIN (%)	StdDev (%)	t value	p
02	91.15	0.09	91.81	0.42	$t(58) = 7.15$	$1.6 \cdot 10^{-9}$
08	85.53	0.00	82.89	0.00	$t(58) = Inf$	0
12	98.48	1.76	100.0	0.00	$t(63) = 4.93$	$6.3 \cdot 10^{-6}$

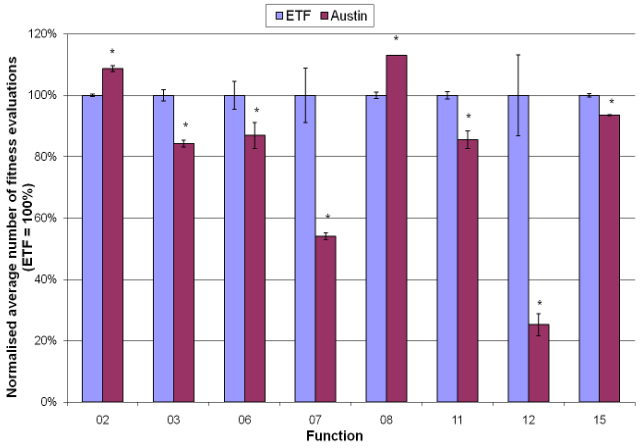


Figure 4. Average number of fitness evaluations (normalised) for ETF versus AUSTIN. The $y - axis$ shows the normalised average number of fitness evaluations for each tool relative to the ETF (shown as 100%) for each of the functions shown on the $x - axis$. The error bars show the standard error of the mean. Bars with a $*$ on top denote a statistically significant difference in the mean number of fitness evaluations ($p \leq 0.05$).

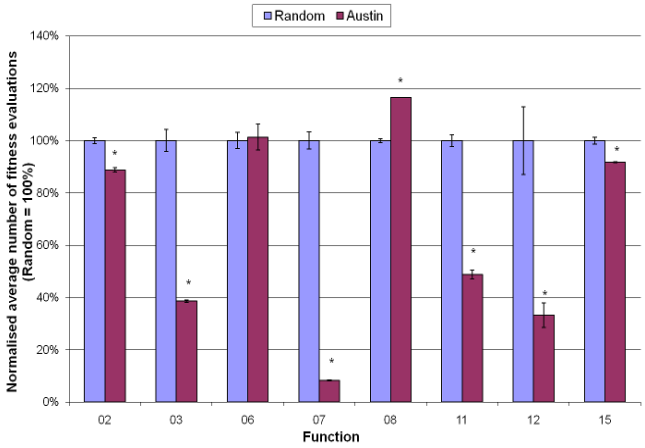


Figure 5. Average number of fitness evaluations (normalised) for random versus AUSTIN. The $y - axis$ shows the normalised average number of fitness evaluations for each tool relative to the random search (shown as 100%) for each of the functions shown on the $x-axis$. The error bars show the standard error of the mean. Bars with a $*$ on top denote a statistically significant difference in the mean number of fitness evaluations ($p \leq 0.05$).

evaluations to achieve its respective level of branch coverage. The results are summarised in Table IV.

Comparison with random search. As a sanity check, the efficiency and effectiveness of AUSTIN was also compared with a random search. Since the random search was performed using the ETF, the same pointer handling technique and input domain reduction were applied as described in Sections IV and V-A respectively. For each branch the random search was allowed at most $10,000$ evaluations. Any branches covered serendipitously by random during the test testing process were counted as covered and removed from the pool of target branches.

The coverage data is presented in Figure 3 and efficiency in Figure 5. Results show that, using the same tests for statistical significance as described in the previous paragraphs, AUSTIN covers statistically significantly more branches than random for functions 02, 03, 07, 11 and 12. For functions 06 and 15 there is no statistically significant difference in coverage, while for function 08, AUSTIN performs statistically significantly worse than random. Recall though, that given a larger fitness budget as mentioned above, AUSTIN is able to achieve a higher coverage for function 08 than the one shown in Figures 2 and 3.

Comparing AUSTIN's efficiency with that of a random search, AUSTIN is statistically significantly more efficient than random for functions 02, 03, 07, 11, 12 and 15. For function 06 we cannot say that either random or AUSTIN is more efficient, while for function 08 random is statistically significantly more efficient than AUSTIN.

C. Threats to Validity

Naturally there are threats to validity in any empirical study such as this. This section provides a brief overview of the threats to validity and how they have been addressed. The paper studied two hypotheses; 1) that AUSTIN is more effective than the ETF in achieving branch coverage of the functions under test and 2) that AUSTIN is more efficient than the ETF. Whenever comparing two different techniques, it is important to ensure that the comparison is as reliable as possible. Any bias in the experimental design that could affect the obtained results poses a threat to the *internal validity* of the experiments. One potential source of bias comes from the settings used for each tool in the

Table IV
SUMMARY OF FUNCTIONS WITH STATISTICALLY SIGNIFICANT DIFFERENCES IN THE NUMBER OF FITNESS EVALUATIONS USED. THE COLUMNS STDDEV INDICATE THE STANDARD DEVIATION FROM THE MEAN FOR ETF AND AUSTIN. THE T VALUE COLUMN SHOWS THE DEGREES OF FREEDOM (VALUE IN BRACKETS) AND THE RESULT OF THE T–TEST. A P VALUE OF LESS THAN 0.05 MEANS THERE IS A STATISTICALLY SIGNIFICANT DIFFERENCE IN THE MEAN NUMBER OF FITNESS EVALUATIONS BETWEEN ETF AND AUSTIN.

Function	Evals ETF (%)	StdDev (%)	Evals AUSTIN (%)	StdDev (%)	t value	p
02	100	1.90	108.78	0.08	$t(58) = 7.67$	$2.20 \cdot 10^{-10}$
03	100	11.14	84.23	6.18	$t(58) = 7.00$	$2.00 \cdot 10^{-9}$
06	100	24.76	86.94	23.42	$t(58) = 2.10$	0.04
07	100	52.40	54.09	6.04	$t(63) = 4.79$	$1.00 \cdot 10^{-5}$
08	100	5.97	113.10	0.00	$t(58) = 9.11$	$8.70 \cdot 10^{-13}$
11	100	6.90	85.58	15.79	$t(58) = 4.41$	$4.49 \cdot 10^{-5}$
12	100	78.33	25.25	19.48	$t(63) = 5.13$	$3.02 \cdot 10^{-6}$
15	100	3.03	93.55	0.90	$t(58) = 9.22$	$5.80 \cdot 10^{-13}$

experiments, and the possibility that the setup could have favoured or harmed the performance of one or both tools.

The experiments with the ETF had already been completed as part of the EvoTest project, thus it was not possible to influence the ETF's setup. It had been manually tuned to provide the best consistent performance across the eight functions. Therefore, care was taken to ensure AUSTIN was adjusted as best as possible to use the same settings as the ETF.

Another potential source of bias comes from the inherent stochastic behaviour of the meta–heuristic search algorithms used in AUSTIN and the ETF. The most reliable (and widely used) technique for overcoming this source of variability is to perform tests using a sufficiently large sample of result data. In order to ensure a large sample size, experiments were repeated at least 30 times. To check if one technique is superior to the other a test for a statistically significant difference in the mean of the samples was performed. Care was taken to examine the distribution of the data first, in order to ensure the most robust statistical test was chosen to analyse the data.

A further source of bias includes the selection of the functions used in the empirical study, which could potentially affect its *external validity*, *i.e.* the extent to which it is possible to generalise from the results obtained. The functions used in the study had been selected by Berner & Mattner Systemtechnik GmbH on the basis that they provided interesting and worthwhile candidates for automated test data generation. Particular attention was paid to the number of branches each function contained, as well as the maximum nesting level of `if` statements within a function. Finally, all the functions from the study contain machine generated code only. While the overall number of branches provides a large pool of results from which to make observations, the number of functions itself is relatively small. Therefore, caution is required before making any claims as to whether these results would be observed on other functions, in particular hand written code.

VI. CONCLUSION

This paper has introduced and evaluated the AUSTIN tool for search–based software testing. AUSTIN is a free, publicly available tool for search–based test data generation. It uses an alternating variable method, augmented with a set of simple constraint solving rules for pointer inputs to a function. Test data is generated by AUSTIN to achieve branch coverage for large C functions. In a comparison with the ETF, a state–of–the–art evolutionary testing framework, AUSTIN performed as effectively and considerably more efficiently than the ETF for 7 out of 8 non–trivial C functions, which were implemented using code–generation tools.

ACKNOWLEDGMENT

We would like to thank Bill Langdon for his helpful comments and Arthur Baars for his advice on the Evolutionary Testing Framework. Kiran Lakhotia is funded by EPSRC grant EP/G060525/1. Mark Harman is supported by EPSRC Grants EP/G060525/1, EP/D050863, GR/S93684 & GR/T22872 and also by the kind support of Daimler Berlin, BMS and Vizuri Ltd., London.

REFERENCES

[1] W. Miller and D. L. Spooner, "Automatic Generation of Floating-Point Test Data," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 223–226, September 1976.

[2] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios, "Application of Genetic Algorithms to Software Testing," in *Proceedings of the 5th International Conference on Software Engineering and Applications*, Toulouse, France, 7-11 December 1992, pp. 625–636.

[3] B. Korel, "Automated software test data generation," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 870–879, 1990.

[4] P. Godefroid, N. Klarlund, and K. Sen, "DART: directed automated random testing," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 213–223, Jun. 2005.

[5] C. Cadar and D. R. Engler, "Execution generated test cases: How to make systems code crash itself," in *Model Checking Software, 12th International SPIN Workshop, San Francisco, CA, USA, August 22-24, 2005, Proceedings*, vol. 3639. Springer, 2005, pp. 2–23.

[6] K. Sen, D. Marinov, and G. Agha, "CUTE: a concolic unit testing engine for C," in *ESEC/SIGSOFT FSE*. ACM, 2005.

[7] N. Tillmann and J. de Halleux, "Pex-white box test generation for.NET," in *TAP*, B. Beckert and R. Hähnle, Eds., vol. 4966. Springer, 2008, pp. 134–153.

[8] K. Lakhotia, P. McMinn, and M. Harman, "Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?" in *4th Testing Academia and Industry Conference - Practice and Research Techniques*, 2009, pp. 95–104.

[9] H. Gross, P. M. Kruse, J. Wegener, and T. Vos, "Evolutionary white-box software test with the evotest framework: A progress report," in *ICSTW '09*, Washington, DC, USA, 2009, pp. 111–120.

[10] Radio Technical Commission for Aeronautics, "RTCA DO178-B Software considerations in airborne systems and equipment certification," 1992.

[11] M. Harman and J. Clark, "Metrics are fitness functions too," in 10^{th} *International Software Metrics Symposium (METRICS 2004)*. Los Alamitos, California, USA: IEEE Computer Society Press, Sep. 2004, pp. 58–69.

[12] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, Jun. 2004.

[13] E. Alba and F. Chicano, "Observations in using Parallel and Sequential Evolutionary Algorithms for Automatic Software Testing," *Computers & Operations Research*, vol. 35, no. 10, pp. 3161–3183, October 2008.

[14] ——, "Software Testing with Evolutionary Strategies," in *Proceedings of the 2nd Workshop on Rapid Integration of Software Engineering Techniques (RISE '05)*, vol. 3943. Heraklion, Crete, Greece: Springer, September 2005, pp. 50–65.

[15] R. Sagarna, A. Arcuri, and X. Yao, "Estimation of Distribution Algorithms for Testing Object Oriented Software," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07)*. Singapore: IEEE, 25-28 September 2007, pp. 438–444.

[16] R. Blanco, J. Tuya, E. Daz, and B. A. Daz, "A Scatter Search Approach for Automated Branch Coverage in Software Testing," *International Journal of Engineering Intelligent Systems (EIS)*, vol. 15, no. 3, pp. 135–142, September 2007.

[17] R. Sagarna, "An Optimization Approach for Software Test Data Generation: Applications of Estimation of Distribution Algorithms and Scatter Search," Ph.D. dissertation, University of the Basque Country, San Sebastian, Spain, January 2007.

[18] R. Lefticaru and F. Ipate, "Functional Search-based Testing from State Machines," in *Proceedings of the First International Conference on Software Testing, Verfication and Validation (ICST 2008)*. Lillehammer, Norway: IEEE Computer Society, 9-11 April 2008, pp. 525–528.

[19] A. Windisch, S. Wappler, and J. Wegener, "Applying Particle Swarm Optimization to Software Testing," in *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*. London, England: ACM, 7-11 July 2007, pp. 1121–1128.

[20] E. Díaz, J. Tuya, R. Blanco, and J. J. Dolado, "A Tabu Search Algorithm for Structural Software Testing," *Computers & Operations Research*, vol. 35, no. 10, pp. 3052–3072, October 2008.

[21] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test-case generation," *IEEE Transactions on Software Engineering*, To appear.

[22] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, Jul. 1976.

[23] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[24] Free Software Foundation, "Gcc, the gnu compiler collection," 2009. [Online]. Available: http://gcc.gnu.org/

[25] K. Lakhotia, M. Harman, and P. McMinn, "Handling dynamic data structures in search based testing," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. Atlanta, GA, USA: ACM, 12-16 Jul. 2008, pp. 1759–1766.

[26] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer, "CIL: Intermediate language and tools for analysis and transformation of C programs," *Lecture Notes in Computer Science*, vol. 2304, pp. 213–228, 2002.

[27] L. D. Costa and M. Schoenauer, "Bringing evolutionary computation to industrial applications with GUIDE," 2009.

[28] M. Prutkina and A. Windisch, "Evolutionary Structural Testing of Software with Pointers," in *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*. Lillehammer, Norway: IEEE, 9-11 April 2008, pp. 231–231.

[29] T. Littlefair, "An Investigation Into The Use Of Software Code Metrics In The Industrial Software Development Environment," Ph.D. dissertation, Faculty of computing, health and science, Edith Cowan University, Australia, 2001.

[30] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener, "The impact of input domain reduction on search-based test data generation," in *ESEC/SIGSOFT FSE*, I. Crnkovic and A. Bertolino, Eds. ACM, 2007, pp. 155–164.

[31] G. D. Ruxton, "The unequal variance t-test is an underused alternative to Student's t-test and the Mann-Whitney U test," *Behavioral Ecology*, vol. 17, no. 4, pp. 1045–2249;1465–7279, Jul 2006.

A Search-Based Approach to Functional Hardware-in-the-Loop Testing

Felix Lindlar, Andreas Windisch
Daimler Center for Automotive IT Innovations
Berlin Institute of Technology
Berlin, Germany
{*felix.lindlar, andreas.windisch*}*@dcaiti.com*

***Abstract*—The potential of applying search-based testing principles to functional testing has been demonstrated in various cases. The focus was mainly on simulating the system under test using a model or compiled source code in order to evaluate test cases. However, in many cases only the final hardware unit is available for testing. This research presents an approach in which evolutionary functional testing is performed using an actual electronic control unit for test case evaluation. A test environment designed to be used for large-scale industrial systems is introduced. An extensive case study has been carried out to assess its capabilities. Results indicate that the approach proposed in this work is suitable for automated functional testing of embedded control systems within a Hardware-in-the-Loop test environment.**

***Keywords*-software testing; automatic testing; road vehicle electronics; optimization methods**

I. Introduction

The complexity of embedded software systems is ever increasing while high software quality is being demanded at the same time. Consequences of poor software quality range from customer dissatisfaction to damage of physical property or the environment. Even considering the lesser of these effects, defective software gives rise to high maintenance costs for the developing company. Therefore, the aim is to find as many errors as possible through testing prior to a system's is released [1]. At present, testing can already take up to more than 50% of development cost and time [2], [3], [4]. However, reasonable empirical evidence shows that one of the major sources of software and systems errors lies in deficient testing of both functional and non-functional properties.

This paper aims at improving functional testing for complex embedded systems. Functional testing serves the purpose of proving the correct implementation of functional requirements and is one of the most important approaches in order to gain confidence in the correct functional behavior of a system. When it comes to large and complex systems, analytical and classical test generation methods often fail because of the combinatorial explosion. Therefore, dynamic testing approaches need to be applied that require executing the system under test (SUT) and assessing its behavior. However, the weakness of dynamic functional testing lies in the fact that the correct behavior of the system is only proven for the sets of test data the system was executed with. In order to prove the correctness of a system it must be fed with all possible inputs, which is obviously not possible for complex systems. Consequently, it is unavoidable to limit the search for test data to critical regions where an erroneous behavior of the system is assumed. The likelihood of finding errors depends on the quality of the test case design. In cases of manual test case design the test quality directly depends on the skills of the tester. If relevant test cases are forgotten, the probability of finding errors is reduced significantly. In order to increase the efficiency of the test and to lower the costs, test case design has to be systematized and automated.

One automation approach is to apply evolutionary testing techniques, thus transforming the testing problem into an optimization problem to be solved using evolutionary algorithms (EA) [5]. Research over recent years has already demonstrated the successful application of EA for functional testing (e.g. [6], [7]). However, the need for executing the SUT using continuous test data has not sufficiently been addressed therein. This drawback has been addressed by the authors in previous work [8], [9], [10], which this paper is based on. Case studies presented in the aforementioned work were performed as Model-in-the-Loop (MiL) tests, in which a model of the SUT is used for test evaluation. However, it is common in the automotive industry that an external company supplies an electronic control unit (ECU) which the software has already been uploaded to. In that case, only the external company has access to the model and the car manufacturer has to perform black-box tests using the ECU to assure that all requirements are fulfilled. Wegener and Kruse already discussed the potential of performing search-based functional testing on various test platforms [11]. They demonstrated the practicality of evolutionary functional testing without extensively carrying out a real case study which, in contrast, is the aim of this paper.

This paper presents an approach to functional Hardware-in-the-Loop (HiL) testing incorporating an actual ECU ready to be deployed in a passenger car. Interface ports of the ECU are interconnected with a HiL testing system in order to enable communication. An advantage of this approach is the incorporation of the actual embedded system hardware into an automated testing process. Performing a large amount of effective HiL tests and thus potentially detecting faults

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.22

early in the development cycle can help reduce costs (e.g. by reducing the number of costly vehicle tests).

II. Evolutionary Functional Testing

Functional testing aims at validating the functional behavior of a software system. Therefore, a proof by contradiction is carried out. The tester is supposed to find test data that violates the functional requirements of the SUT. This task is very cost-intensive, time-consuming and error-prone when done manually, thus the automation of this process is highly aspired.

Evolutionary functional testing allows for the automation of functional testing by transforming the task of generating relevant test data into an optimization problem and tries to solve it using a meta-heuristic search technique [12]. Stochastic search techniques such as EA are used to find approximations to the optimal solution in multidimensional search spaces. These general-purpose search techniques make very few assumptions about the underlying problem they are attempting to solve. As a consequence, they are useful during the automated generation of effective test data.

A working test environment provided, three domain-specific components have to be supplied for each test goal in order to perform an evolutionary functional test. First, a suitable fitness function has to be derived from the functional specification of the SUT. The fitness function evaluates every test run with respect to its criticality and assigns a fitness value to it. This fitness value is used to compare multiple test data sets with each other and to optimize the data sets towards a failure of the system behavior. The second problem that has to be tackled is defining the search space. When testing complex embedded systems the execution time for a single test run can take up a considerable amount of time. Effort has to be put into limiting the search space to the important data ranges to make the search feasible. Finally, a test driver has to be implemented. The task of the test driver is to map the generated test data to the input ports of the SUT and to execute it.

III. Test Environment

The test environment or Evolutionary Testing Framework (ETF) has been developed in the context of the EU-funded project EvoTest [13], [14]. In the first section of this chapter our approach to generating realistic continuous input signals is described. Section III-B describes the basic functionality of the ETF, whereas section III-C introduces the experimental testing environment necessary for facilitating HiL testing.

A. Test Data Generation

The creation of effective test scenarios for dynamic systems in general, i.e. the creation of system input signals by arrays of numbers, is hard to realize for real-world models as these often require the signals to have a specific minimum length (number of included time steps). In combination with a small sample rate, the resulting size of arrays representing the signals prevent its application to optimization engines. Accordingly, a distinction between optimization and simulation sequences needs to be accomplished as shown in figure 1. The optimization sequence (a set of variables $\{v_n, n \in \mathbb{N}\}$) is used by an optimization engine and can be transformed into a simulation sequence which in turn is used as input for the execution of the model under test.

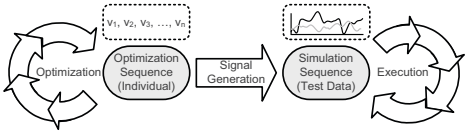


Figure 1. Signal representations split into optimization and simulation sequences

In previous work the authors proposed approaches for generating and optimizing signals by stringing together a certain number of parameterized base signals [10]. These approaches make use of further information about the input signals for the SUT that need to be provided by the tester. This signal specification, on the one hand, includes general attributes, e.g. the length of the signals to be generated and their designated resolution. On the other hand, it also contains separate specifications of the signals for each input of the SUT, such as minimum and maximum amplitude values for each signal.

B. Evolutionary Testing Framework

The Evolutionary Testing Framework has been used to perform the case study. It provides general components and interfaces to facilitate the automatic generation, execution, monitoring and evaluation of test scenarios. It is designed to be used with large and complex systems which are common in industrial settings. The underlying evolutionary computation techniques are hidden through a user interface. This is an important practical component because it allows practicing software engineers to use the ETF without requiring any knowledge of evolutionary computation. Optimization within the framework is carried out by an optimization engine which is generated using the Evolving Objects library [15]; together, they implement a highly-configurable genetic algorithm (GA).

Figure 2 shows the general workflow of the ETF. The framework has to be capable of generating realistic input signals due to the dependence on realistic input signals of the SUT. This task is done by the signal generator, which in turn expects a signal representation from the tester. This signal representation describes the nature of each input signal to be generated. Based on the signal representation the signal generator is able to create and provide the optimization engine with a description of the problem or the search space, respectively. According to this, an optimization engine is generated and compiled in order to optimize the problem

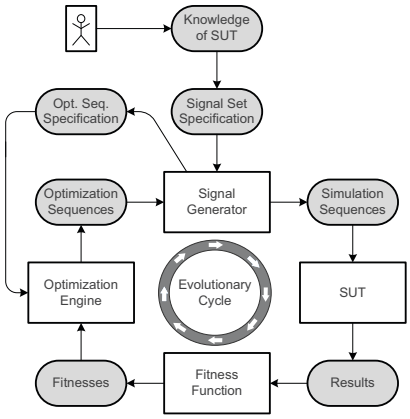


Figure 2. System Architecture of the Test Environment

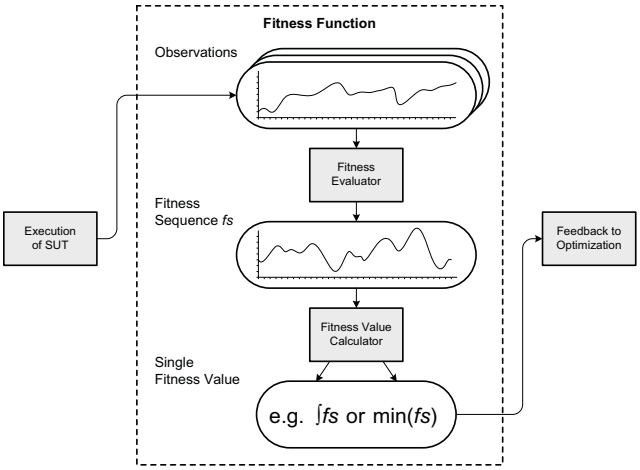


Figure 3. Handling signals within fitness calculation.

at hand. The optimization engine will create and initialize the first population of individuals which are sent to the signal generator. The signal generator uses the provided set of individuals in combination with the specified signal representation to generate a number of signal sets. These signal sets are passed to the test driver. The test driver then executes the SUT with the provided signal sets and records the resulting output signals.

The results which represent the system behavior for the respective input scenarios are then evaluated using the Fitness Function as shown in figure 3. The monitored set of observation signals is evaluated for each enclosed time step. The *Fitness Evaluator* calculates one fitness value for each time step using distance metrics, based on the observations measured during experimentation. This component is responsible for calculating the distance of the currently evaluated test data to violating the safety requirement under investigation (see section IV-D). However, meta-heuristic search techniques expect one single evaluated fitness value instead of a sequence of fitness values. For this reason the integral of this fitness sequence or the minimum value are possible options.

Subsequently, the fitness values for each input signal set are passed to the Optimization Engine, which creates a new population of individuals based on this assessment. This closes the evolutionary cycle and the optimization continues until the demanded solution has been found or another stopping criterion applied.

C. Hardware-in-the-Loop Test Platform

The optimization runs were performed within a HiL setup using an ECU. While this ECU will be later installed inside the actual vehicle, the environment around it is still simulated. The environment has to support real-time behavior to ensure that the test is as realistic as possible. As a result the execution times are usually longer compared to a simulated MiL test. However, these disadvantages are compensated by a realistic test setup comparable to a driving scenario using real vehicles. The approach of using EA to test ECUs is especially useful when the code running on the ECU is unknown. This is quite common within the automotive industry because control units and control software are often developed by a supplier. Obviously, the car manufacturer has to reassure that all functional and non-functional requirements are fulfilled. The goal of the case study introduced in this paper is to prove that the *Radar Decision Unit* (RDU) reacts correctly in critical situations. The RDU is an ECU which contains software for an Adaptive Cruise Control system (ACC) which will be described in detail in section IV-A.

Figure 4 shows the complete setup of the test environment, whereas figure 5 shows pictures of the setup the tester is working with. The test environment has been carefully designed with respect to simulating driving scenarios which are as realistic as possible at the interface of the RDU. The purpose of *PC 1* is to generate the actual test data and to evaluate the outcome of each test run. The *EvoTest Framework* component (ETF, also see section III-B) runs on *PC 1* and provides capabilities to perform an evolutionary test. Since the ETF cannot communicate with the HiL system directly, it sends the generated test data to the *ProoveTech:TA* component which then forwards the data to the *real-time computer*. The *real-time computer* uses the test data and an environmental model to simulate a realistic driving scenario. Further tasks of the *real-time computer* are transmitting the current velocity to the *Display* and writing simulation data on the Controller Area Network (CAN). Two different CAN networks are involved in this case study: the *Sensor-CAN* and the *Dynamics-, Chassis-, and Body-CAN*. The *Radar-Simulator* imitates an actual radar and writes this information on the *Sensor-CAN*. With this data the RDU determines the relative position and speed of vehicles driving in front of it. Additional required data, such as the current velocity and the interaction of the driver with the speed

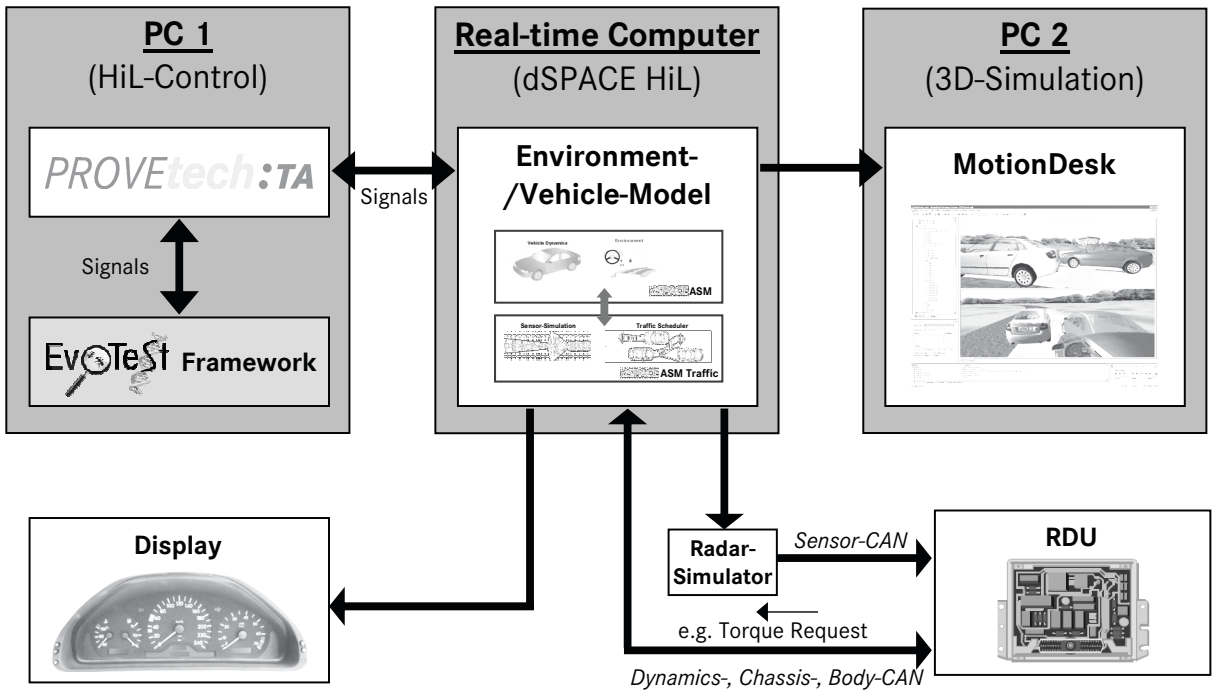


Figure 4. Hardware-in-the-Loop test platform

control lever, are transmitted via the *Dynamics-,Chassis-, and Body-CAN*. Finally, *PC 2* is running a 3D simulation of the driving scenarios. *PC 2* receives information about the current speed and position of all vehicles taking part in the test run from the *real-time computer* and updates the 3D simulation accordingly.

IV. Evolutionary Testing of the ACC System

This chapter provides an overview of the prerequisites required for testing the adaptive cruise control system using the test environment introduced in section III. A brief description of the ACC system is given in the first section of this chapter. In section IV-B the driving scenario used in the optimization runs is introduced. Section IV-D deals with the design of the fitness function.

A. Adaptive Cruise Control System (ACC)

Many accidents are rear-end collisions on highways. When activated, the purpose of ACC is not only to maintain a given speed but also to control the distance to preceding vehicles and thus prevent accidents. The first generation Adaptive Cruise Control system was introduced in 1998 by Daimler. Currently work is being performed on the third generation. The system has two main functionalities: distance control and cruise control. The cruise control maintains the setpoint speed intended by the driver independently of the engine load. Gear shifts are also performed automatically. In case of a violation of the minimum distance to a preceding vehicle the distance control adjusts the speed by braking with up to 20 percent of the maximum braking power. If the maximum braking power of the system is insufficient to maintain a safe distance, the driver is alerted by visual and acoustical signals. The driver then has to take back control over the car to resolve the situation.

The distance control function is mainly used in the long-distance haulage sector, on freeways, highways or expressways as well as routes similar to freeways with curvature radiuses larger than 250m. The sensor system must therefore be capable of reliably detecting the traffic situations relevant to this area. These also include construction site areas or areas with changed traffic routing.

B. Test Scenario

Due to the fact that a higher number of input parameters not only increases the runtime of the EA but also complicates the design of a suitable fitness function a basic scenario is chosen. As shown in figure 6 only two vehicles are taking part in the simulation: a vehicle that is equipped with the Adaptive Cruise Control system and a preceding vehicle. Both vehicles are driving in the same lane following a straight road. Turns are not provided in the scenario. In doing so, cases are avoided in which the simulation would get into physically critical situations that do not influence the functionality of the distance control (e.g. if the velocity of the vehicle is too high for a given curve radius so that the vehicle can't stay inside the lane).

(a) (b) (c)

Figure 5. Pictures of the Evolutionary Hardware-in-the-Loop Testing Setup: a) shows the testing rack on the whole, b) presents the user interface: on top the car dashboard display can be seen, the upper screen shows the current simulated driving scenario from different points of view and the lower screen is used to control the entire setup. Finally, c) shows the actual embedded ECU under test (black-box).

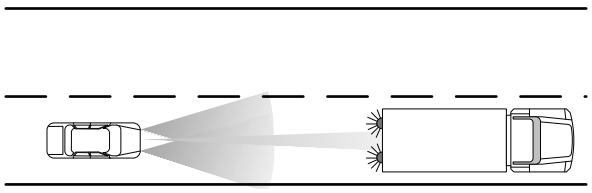


Figure 6. Adaptive Cruise Control Scenario

C. Search Space

For this case study two input signals had to be generated in order to stimulate the SUT: the speed of the preceding vehicle and the interaction of the driver with the cruise control lever. For each of the signals a number of parameters (e.g. amplitude bounds and type of signal) needed to be specified. These parameters make up the search space or individual specification for the evolutionary functional test.

An actual test case is always preluded by a short initialization sequence. During this initialization sequence the vehicle, which is equipped with the ACC system, is accelerated from standstill until it reaches a defined velocity. The system then behaves according to what has been described in IV-A. The preceding vehicle on the other hand drives the speed that is provided by an input signal and varies during the progress of the simulation sequence. The goal of the test is to find a scenario where the minimum distance criterion is severely violated, i.e. an accident is inevitable, without a warning signal being raised.

D. Fitness Function

In order to guide the evolutionary search towards desired scenarios as described in the previous section, the objective function (fitness function) must be designed accordingly. Therefore, the time to collision as well as the time of the arrival of the driver warning must be taken into account. Section IV-D1 describes the calculation of the time to collision whereas section IV-D2 describes how the final objective function, including the warning arrival time, is designed.

1) Time-to-Collision: We need to distinguish two time-to-collision calculations: one that calculates the time to collision if both our own car and the preceding car do not brake (TTC_d); this time to collision can be calculated as follows:

$$\begin{aligned} TTC_d &= \frac{d(t_0)}{max(v_{ego}(t_0) - v_{target}(t_0), 0.002\frac{m}{s})} \\ &= \frac{d(t_0)}{max(v_{rel}(t_0), 0.002\frac{m}{s})} \end{aligned} \quad (1)$$

Additionally, we want to calculate the time to collision if the driver of our car is actually braking because of the

ACC red warning light flashing up (TTC_b). Therefore, we need to account for the reaction time of the driver and the possible deceleration.

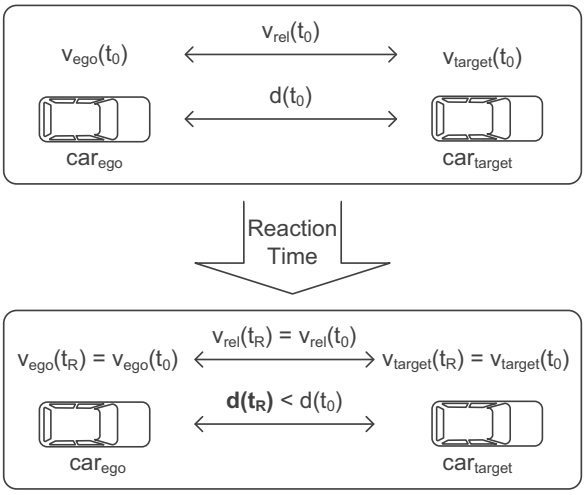


Figure 7. Changing Distance after Reaction Time

At first, the distance to the preceding car after reaction time T_R of the driver needs to be calculated as shown in figure 7; $t_R = t_0 + T_R$ is the very point of time after reacting.

$$\begin{aligned} d(t_R) &= d(t_0) - v_{ego}(t_0) \cdot T_R + v_{target}(t_0) \cdot T_R \\ &= d(t_0) - v_{rel}(t_0) \cdot T_R \end{aligned} \quad (2)$$

If the distance to the preceding car after reaction time is already smaller than zero, a crash will inescapably occur. Thus, in this case the non-braked time to collision TTC_d can be used. If the distance after reacting is greater than zero, we have time for the car to be decelerated. Our approach for calculating the braked time to collision is as follows. The distance to the preceding car at the critical collision time t_c can be calculated as:

$$d(t_c) = d(t_R) - s_{ego}(t_c) + s_{target}(t_c) \quad (3)$$

Since we propose that the target distance is zero in case of a collision, we can simply set $d(t_c) \stackrel{!}{=} 0$ and determine the associated time t_c.

$$t_c = \frac{v_{rel}(t_0) + \sqrt{v_{rel}(t_0)^2 + 2 \cdot a_{ego} \cdot d(t_R)}}{a_{ego}} \quad (4)$$

The calculated time t_c is the elapsing time from beginning of braking, i.e. after reaction time, until the distance to the preceding car equals zero, i.e. until a collision occurs. In order to calculate the entire time to collision from the maneuver starting time t_0, i.e. including reaction time, we simply need to add the reaction time to t_c. Remember that we can use the time to collision for constant speeds TTC_d, if the vehicles collide during the driver's reaction time.

$$TTC_b = t_c + T_R \quad (5)$$

TTC_b is undefined if the term inside of the square root is lower than zero, i.e. in the case if $v_{rek}(t_0)^2 + 2 \cdot a_{ego} \cdot d(t_R) < 0$. This appears if the distance to the preceding car is large enough, such that a collision will be avoided by braking. In this case the time to collision without braking can be consulted, since we still want to ensure the search guidance. Hence, the overall time to collision can be formulated as:

$$TTC = \begin{cases} TTC_d, & d(T_R) \leq 0, \\ TTC_d, & (v_{rel}(t_0)^2 + 2 \cdot a_{ego} \cdot d(T_R) < 0), \\ TTC_b, & else. \end{cases} \quad (6)$$

2) Final Objective Function: A first approach to finding interesting scenarios would be setting the minimum value of the TTC signal as an objective function.

$$objValue = min(TTC(t)) \quad (7)$$

This equation does not incorporate the arrival time of the warning light. The warning signal is either 0 if a warning is not to be displayed, or it is 1 if the driver must be alerted. If the warning light is showing up, the time to collision is not of importance to us anymore, since the driver must take over control. Thus, we can simply multiply the warning signal with a big value δ and add it to the TTC signal, such that these values are assuredly not the minimum values chosen by the min function.

$$objValue = min(TTC(t) + Warn(t) \cdot \delta) \quad (8)$$

This function returns the minimum time to collision for each input scenario. In case the warning signal arrives and the vehicles still collide, this is a matter of the virtual driver since there is no braking during the simulations. Adding big values whenever the warning signal is active ensures that these cases will not cause good, i.e. low objective values.

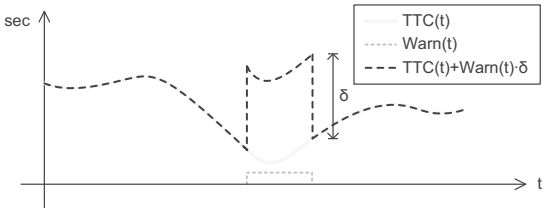


Figure 8. Signal Characteristics of Objective Function

Additionally, to distinguish individuals in more detail, i.e. to reward more critical scenarios with a lower fitness and punish less critical scenarios with a bigger fitness, the

minimum TTC value can be divided by the gradient of the TTC signal $\frac{\partial TTC}{\partial t}(t_{min})$ at time step t_{min} to incorporate the amount of change of the signal into the evaluation.

$$objValue = \frac{min(TTC(t) + Warn(t) \cdot \delta)}{\left|\frac{\partial TTC}{\partial t}(t_{min})\right|} \quad (9)$$

V. Experiments

This section describes the experimental case study intending to reveal the ability of the proposed evolutionary functional testing approach to generating real-world signals for testing an ECU that enables adaptive cruise control functionality. Applied to evolutionary testing, the underlying search technique searches for signal sequences to be used as test stimuli for the SUT. This search is based on and guided by the applied objective function described in section IV-D. The experiments were carried out in real-time as HiL tests.

A. Configuration of the Evolutionary Testing Framework

In order to prepare the ETF for performing the optimization run, the characteristics of the signals to be generated have to be specified in order to be applied as inputs for the SUT. As described in section IV-C, both the utilization of the ACC control lever and the velocity of the preceding vehicle are signals that are optimized by the ETF. Thus they need to be specified accordingly. The signal for the control lever is specified to be composed of ten signal segments that can only be rendered using impulse transition types. The amplitudes of these signal segments can only take integer values between 0 and 2. An amplitude value of 0 is thus to be interpreted as a state of rest whereas the values 1 and 2 represent the acceleration and deceleration commands respectively. The velocity of the preceding vehicle is specified to be composed of eight signal segments that are connected by spline transitions. The amplitude values are supposed to be bounded to real values in between 15 and 35 meters per second. This signal specification leads to an optimization sequence consisting of 36 real-valued parameters.

Moreover, the underlying search engine is another configuration point within the framework. Table I shows the settings of the GA used for the experiments. The parameters have been determined using mathematical benchmarking functions (e.g. Rastrigin and Ackley); i.e. several optimization runs have been performed with different parameter settings while the resulting best parameters were used for the experiments. A population consists of 100 individuals and selection is carried out by stochastic universal sampling given a generation gap of 85%. Evolving a new generation is done by applying Gaussian crossover, Gaussian mutation and elitest reinsertion. For Gaussian crossover the offspring values are randomly chosen from a Gaussian distribution with the mean given by the average of the values of both parents and a standard deviation such that the probability of being outside the Real interval is 10%. Gaussian mutation adds a normally distributed random value with mean 0 to the value of the original individual. Reinsertion is realized in different stages of the evolutionary optimization process. The optimization terminates as soon as a fitness value less than or equal to 0 has been found or the maximum number of generations has been evolved.

Initialization	Type	Uniform
	Individuals	100
Selection	Type	Stochastic Universal Sampling
	Gen. Gap	85%
Crossover	Type	Gaussian
	Rate	85%
Mutation	Type	Gaussian
	Rate	10%
Reinsertion	Type	Elitest (15%), Parents (25%), Offsprings (100%)
Termination	Generations	100
	Fitness	≤ 0

Table I
Configuration of GA used for the experimental case study.

Each test run is executed in real-time and no parallelization is possible, because of only a single ECU and one environmental simulation controller being available. Consequently, a single optimization run results in a total execution time of roughly one week ($60s$ [Duration per Test Run] $\cdot$ 100 [Individuals] $\cdot$ 100 [Generations] $= 600.000s \approx 1$ week). However, if multiple ECUs would be available, optimization time could be improved significantly. Anyway, the test data generation process for both GA and random testing is supposed to be repeated 3 times in order to acquire more reliable results.

B. Results

Figure 9 shows the convergence progress for the optimization runs that applied GA and random data generation respectively. By optimizing the driving scenario as described in section IV-B, the ETF was able to find situations that can be classified as the more critical the more the optimization progresses. This applies to both GA and Random used for optimization. Although the best solutions that have been discovered by both approaches at the beginning of the optimization run are comparable, as expected GA were able to find better solutions overall. However, both approaches were not able to find a driving scenario that would lead to a violation of the safety requirement under investigation. More precisely, the ETF was not able to find driving scenarios for which the ACC controller would incorrectly warn the driver too late, such that a crash would thus be unavoidable. This is indeed not surprising since the controller software is already in a development state of serial production and thus has

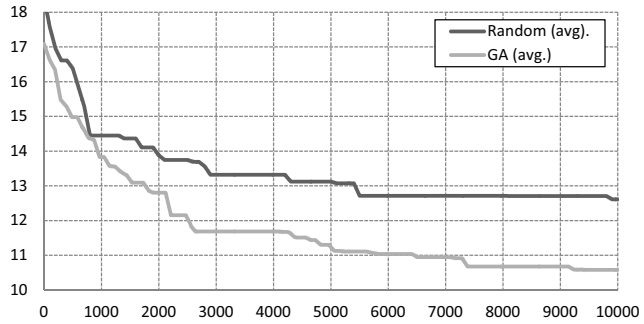


Figure 9. The convergence characteristics of GA and Random for the conducted experiment: the x-axis shows the number of objective function evaluations, the y-axis shows the best fitness value. Results are averaged over 3 runs.

already been tested thoroughly. One approach to evaluate the created fitness function would be to manually inject faults into the controller software, and compile and flash it onto the ECU. For the driving scenario under investigation, the timing parameters for triggering the warning could for instance be modified to easily evoke defective behavior. This was not possible for the research at hand because we did not have access to the model of the SUT. Still, the optimization progress clearly indicates that the approach can be used for automatically generating interesting test data searching for violations of particular requirements in terms of evolutionary functional testing.

Although the ETF was unable to find violations of the reviewed safety requirement, it still found interesting driving scenarios. One of these is illustrated in figure 10. It contains four diagrams which show different signals that were either used as inputs or recorded from executing the SUT and observing its behavior. Both the lever position and the speed of the target vehicle are signals that were optimized by the test environment. The remaining signals were directly taken from observing the behavior of the system. As can be seen, the own car is accelerating up to $27\frac{m}{s}$ during the initialization phase. After that its speed depends on the speed of the preceding car, the resulting distance, and the desired speed which in turn depends on how the driver uses the ACC control lever. At $54seconds$ the system raises a sound warning to the driver and shortly after that triggers a crucial visual warning due to the rapid speed reduction of the preceding car which is accompanied by a noticeable shortening of the vehicle distance. The lower-right diagram visualizes the objective value for each single time step which is calculated as described in section IV-D.

VI. Conclusion and Future Work

This paper reported on the application of evolutionary testing for assessing the compliance of a particular safety requirement of a realistic industrial embedded system. The aim of this assessment was to automatically search for system stimuli which lead to a violation of the selected safety requirement. As testing object we selected an ACC system which is an extended cruise control additionally maintaining a safety distance to vehicles driving in front and is thus particularly relevant to safety. The safety requirement to be assessed demands warning the vehicle's driver in case the safety distance cannot be kept due to, for example, abrupt braking of the preceding vehicle. The testing objective is to find driving situations in which the system warns the driver too late, such that a crash is unavoidable, even if the driver reacted appropriately.

After a brief introduction to evolutionary functional testing we described the design and setup of our HiL test system which was used to carry out the experimental case study. This includes a detailed description of how continuous input signals can be generated and optimized. Both GA and random data generation were used to automatically generate test cases for the selected adaptive cruise control system. Subsequently we elaborated on the ACC system in general, the driving scenario that we wanted to assess and the fitness function guiding the evolutionary search towards test cases violating the requirement under investigation. The results of the experiments show that evolutionary functional testing of embedded systems within a HiL environment is a very promising approach to ensure safety requirements. Even though no driving scenario has been found that violates the requirement, the SUT has still been executed and assessed with a great number of different input stimuli. Hence it was possible to greatly enhance confidence in the system. Another big advantage of this approach is that after defining the fitness function to be used for optimization, no human effort is needed to execute this large amount of test scenarios.

However, more experiments examining further safety requirements or taking into account further test objects must be carried out in order to be able to make more general statements about the applicability of this approach to realistic and complex systems. The effectiveness of this approach is highly dependent on the quality of the underlying optimization technique, thus thoroughly tuning the metaheuristic search engine used for optimization is required. The application of different optimization techniques may also lead to better results [16] and should therefore be analyzed. In order to make the approach applicable to continuous control systems in more business areas, domain-specific signal segment transition types need to be developed in order to expand the signal generation approach. These areas could, for example, be in medical science where a signal transition featuring the characteristics of human nerve impulses would be imaginable to generate spike trains. Finally, the HiL test environment introduced in this paper could also be used to assess real-time safety requirements such as the execution times of an airbag controller, which would then tend to finding worst or best case execution times respectively.

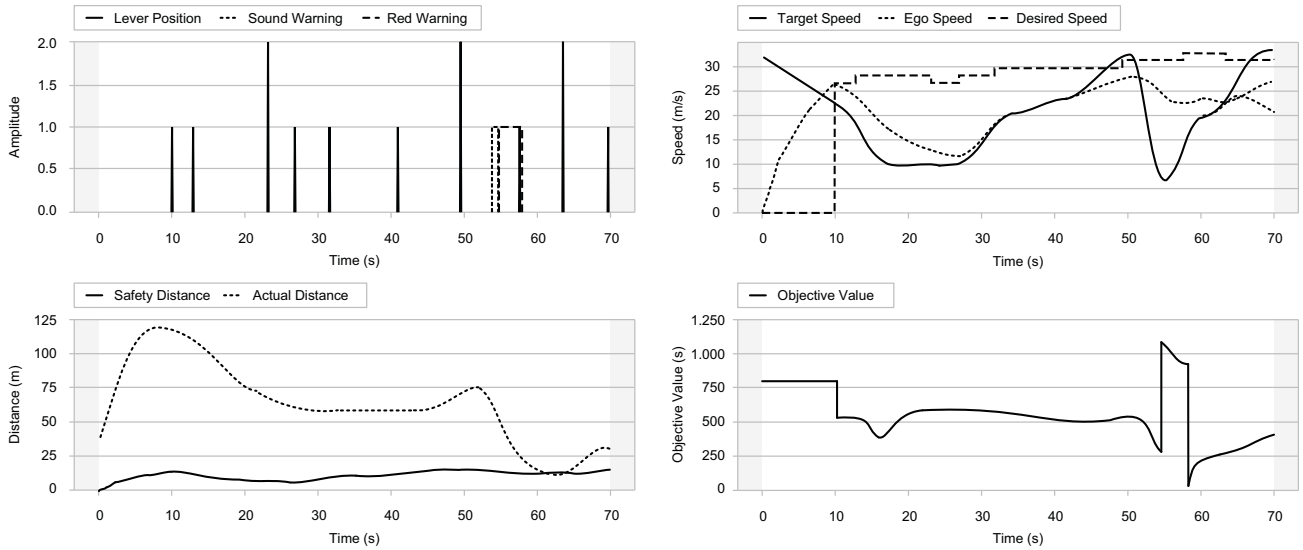


Figure 10. Input and output signals of one critical driving scenario detected by the ETF

VII. Acknowledgments

Parts of this work were supported by EU grant 33472 (EvoTest).

References

[1] W. H. Deason, D. B. Brown, K.-H. Chang, and J. H. Cross, "A rule-based software test data generator," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 1, pp. 108–117, Mrz 1991.

[2] B. Beizer, *Software testing techniques (2nd ed.)*. New York, NY, USA: Van Nostrand Reinhold Co., 1990.

[3] D. R. Graham, "Software testing tools: A new classification scheme," *Software Testing, Verification and Reliability*, vol. 1, no. 3, pp. 17–34, 1991.

[4] D. C. Ince, "The automatic generation of test data," *The Computer Journal*, vol. 30, no. 1, pp. 63–69, 1987.

[5] B. F. Jones, H. Sthamer, and D. E. Eyres, "Automatic test data generation using genetic algorithms," *Software Engineering Journal*, vol. 11, no. 5, pp. 299–306, Sep. 1996.

[6] O. Bühler and J. Wegener, "Automatic testing of an autonomous parking system using evolutionary computation," in *Proceedings of SAE 2004 World Congress*, March 2004.

[7] ——, "Evolutionary functional testing of a vehicle brake assistant system," in *Proceedings of 6th Metaheuristics International Conference*, August 2005.

[8] F. Lindlar, A. Windisch, and J. Wegener, "Integrating model-based testing with evolutionary functional testing," in *3rd International Workshop on Search-Based Software Testing at ICST 2010*, Paris, France, 2010, to be published.

[9] A. Windisch, F. Lindlar, S. Topuz, and S. Wappler, "Evolutionary functional testing of continuous control systems," in *Proceedings of the 11th annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2009, pp. 1943–1944.

[10] A. Windisch and N. Al Moubayed, "Signal generation for search-based testing of continuous systems," in *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation Workshops*, April 2009, pp. 121–130.

[11] J. Wegener and P. M. Kruse, "Search-based testing with in-the-loop systems," in *SSBSE '09: Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 81–84.

[12] A. Baresel, H. Pohlheim, and S. Sadeghipour, "Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2003, pp. 2428–2441.

[13] D. M. Dimitrov, I. Manova, and I. Spasov, "Evotest - framework for customizable implementation of evolutionary testing," in *International Workshop on Software and Services*, Sofia, Bulgaria, October 2008.

[14] EvoTest, http://evotest.iti.upv.es/, Last accessed: June 2010.

[15] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, "Evolving objects: A general purpose evolutionary computation library," in *Artificial Evolution*, 2001, pp. 231–244.

[16] A. Windisch, S. Wappler, and J. Wegener, "Applying particle swarm optimization to software testing," in *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, 2007, pp. 1121–1128.

Using Search Methods for Selecting and Combining Software Sensors to Improve Fault Detection in Autonomic Systems

Maxim Shevertalov, Kevin Lynch, Edward Stehle, Chris Rorres, and Spiros Mancoridis
Department of Computer Science
College of Engineering
Drexel University
3141 Chestnut Street, Philadelphia, PA 19104, USA
{max, kev, evs23, spiros}@drexel.edu, crorres@cs.drexel.edu

Abstract

Fault-detection approaches in autonomic systems typically rely on runtime software sensors to compute metrics for CPU utilization, memory usage, network throughput, and so on. One detection approach uses data collected by the runtime sensors to construct a convex-hull geometric object whose interior represents the normal execution of the monitored application. The approach detects faults by classifying the current application state as being either inside or outside of the convex hull. However, due to the computational complexity of creating a convex hull in multi-dimensional space, the convex-hull approach is limited to a few metrics. Therefore, not all sensors can be used to detect faults and so some must be dropped or combined with others.

This paper compares the effectiveness of genetic-programming, genetic-algorithm, and random-search approaches in solving the problem of selecting sensors and combining them into metrics. These techniques are used to find 8 metrics that are derived from a set of 21 available sensors. The metrics are used to detect faults during the execution of a Java-based HTTP web server. The results of the search techniques are compared to two hand-crafted solutions specified by experts.

***Keywords*-genetic algorithms; genetic programming; search based software engineering; autonomic computing**

I. Introduction

Complex software systems have become commonplace in modern organizations and are critical to their daily operations. They are expected to run on a diverse set of platforms, while interoperating with a wide variety of applications and servers. Although there have been advances in the engineering of software, faults still regularly cause system downtime. The downtime of critical applications can create additional work, cause delays, and lead to financial loss [1]. Faults are difficult to detect before an executing system reaches a point of failure, as the first symptom of a fault is often system failure itself. While it is unrealistic to expect software to be fault-free, actions can be taken to reinitialize the software, quarantine specific software features, or log the software's state prior to the failure.

Many fault detection approaches rely on runtime metrics such as CPU utilization, memory consumption, and network throughput. One detection approach, created by the authors, uses information collected by the runtime sensors and constructs a convex-hull geometric object that represents the normal execution of the monitored application [2]. The convex-hull approach then detects faults by classifying the state of an executing application as inside or outside of the convex hull. If the state is classified as being outside of the hull it is considered to be failing.

Due to the computational complexity of creating a convex hull in multi-dimensional space (one dimension per metric), the convex-hull approach is limited to a few metrics. Therefore, not all sensors can be used to detect faults and some must be dropped or combined with others. For example, instead of using `heap memory` and `non-heap memory` as individual metrics, these sensors can be summed together to create a `total memory` metric. The problem of selecting and combining sensors to create a limited number of metrics can be formulated as a search problem.

This paper compares the effectiveness of genetic-programming, genetic-algorithm, and random-search approaches to solving the sensor selection and combination problem described above. These techniques are used to find 8 metrics derived from a set of 21 available sensors. The metrics are used to detect faults during the execution of NanoHTTPD [3], which is a Java-based HTTP server.

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.23

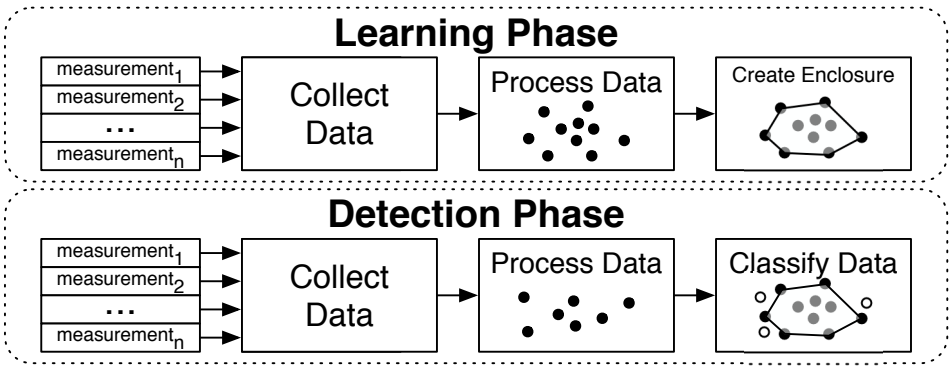


Figure 1. During the learning phase, the convex-hull approach collects and processes runtime measurements. It then constructs a convex-hull enclosure that represents the normal execution of the monitored system. During the detection phase, runtime measurements are collected and processed. Then, each data point (vector of metrics) is classified as normal (gray points) if it is inside the enclosure, or anomalous (white points) if it is outside of the enclosure.

The solutions found by the search techniques are targeted to a specific set of 9 seeded faults and are compared to each other as well as two hand-crafted solutions specified by domain experts. This paper illustrates the relative effectiveness of search based software engineering techniques as well as the solutions proposed by domain experts.

The rest of the paper is organized as follows: Section II presents the background information on fault detection, specifically the convex-hull method; Section III describes the search techniques used in this work; Section IV compares the solutions found using search to each other as well as to the two hand-crafted solutions; finally Section V states conclusions and plans for future work.

II. Background

Autonomic computing is a branch of software engineering concerned with creating software systems capable of self-management [4]. One of the aspects of autonomic computing, and the focus of this work, is fault detection. Fault detection is a step toward creating a self-healing software system, meaning a system that is aware when a fault has occurred and can correct it.

Existing methods of detecting software faults fall into two categories, signature-based methods and anomaly-based methods [5]. Signature-based methods detect faults by matching measurements to known fault signatures. These techniques can be used in static fault-checking software such as the commercial antivirus software McAfee [6] and Symantec [7], as well as network intrusion detection systems such as Snort [8] and Netstat [9]. These techniques can also be used to detect recurring runtime faults [10].

If a set of known faults exists, then training a system to recognize these faults will typically lead to better fault detection. However, that system is unlikely to recognize faults it has not seen before. For example, a fault caused by a zero-day virus is unlikely to be detected by commercial antivirus software because there are no known patterns to recognize.

Anomaly-based methods learn to recognize the normal runtime behavior of the monitored system and classify anomalous behavior as potentially faulty. The advantage of using anomaly-based methods is that they can detect previously unseen faults. However, they risk incorrectly treating any newly encountered good states as faulty. This occurs when insufficient training data is supplied to the method.

Typically, anomaly-detection methods begin by collecting sensor measurements of a system that is executing normally. Then, they construct a representation of the monitored system and compare any future measurements against that representation. A naïve approach will assume that all sensors are independent and determine the safe operating range for each of them. In other words, during the learning phase, this method will record the maximum and minimum values of each sensor and then classify the system as faulty when any of the measurements fall outside of their determined ranges.

While the naïve approach is capable of detecting some faults, it can fail when the metrics are dependent. Therefore, more sophisticated fault detection techniques assume that there are dependencies between metrics. Many of these techniques employ statistical machine learning [11]–[14]. A common approach is to use sensor correlations to represent a monitored system. During detection, if the correlations between sensors become significantly different from the learned correlations, the system is classified to be

in a faulty state.

A. Convex-Hull Approach to Fault Detection

The convex-hull method is an anomaly-detection method and, thus, can detect faults not seen before. However, unlike the previously discussed anomaly-detection methods, it does not use statistical machine learning. Instead, it uses computational geometry and works well independent of whether the metrics are dependent or not.

Figure 1 illustrates the phases of the convex-hull fault-detection technique, which consists of a training phase and a detection phase. During the training phase, the normal execution behavior of the monitored system is modeled by constructing a convex-hull enclosure around good measurements (*i.e.,* when faults were not observed). During the detection phase, observed measurements are classified as normal or anomalous based on whether they are inside or outside of the convex hull.

The convex hull for a finite set of points is the smallest convex polygon that contains all of the points. A convex polygon is a polygon such that for every two points inside the polygon the straight-line segment between those points is also inside of the polygon. A convex hull for a finite set of points in one-dimensional space (one metric) is the line segment between the largest and smallest values. In two-dimensional space (two metrics) one can think of each point as a peg on a board and the convex hull as an elastic band that snaps around all of those pegs. In three-dimensional space (three metrics) one can think of the convex hull as an elastic membrane that encompasses all of the points.

A problem with the convex-hull method, and the reason for this work, is that it is computationally expensive to construct a hull in higher dimensions given a large set of points. As more metrics are considered, the process of computing the convex hull becomes more time consuming. Once constructed, however, the hull can classify new points as in or out at a rate of about 100 classifications a second. The QHull library [15], which this work uses for all of its convex hull calculations, is limited to 8 dimensions. Due to this constraint, a set of sensors must be turned into a set of no more than 8 metrics. The QHull library is the premier convex hull computational library that is used in many mathematical tool kits. Even with its limitation to 8 metrics, it is still the fastest library for computing convex hulls in higher than 3 dimensions.

Evolutionary algorithms [16] have shown promise in finding good solutions in a large space of potential answers. Evolutionary algorithms begin their process by constructing a random population of answers. Next they evaluate all of the solutions using a fitness function. The population is then recombined via a breeding process to construct the following generation. Given a new population the process is repeated until some acceptable answer is found.

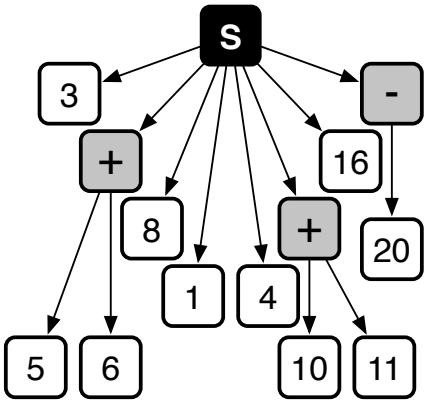


Figure 2. The chromosome is represented as a tree structure. It has three types of nodes: a root node (black); an expression node (gray); and a terminal node (white).

Given the problem of selecting and combining sensor measurements into metrics, our conjecture was that an evolutionary algorithm would find a good solution. The search space is large and the dependencies between different sensors is not known. For these and other reasons, which will become clear further in the paper, an evolutionary algorithm was chosen as the start of our work.

In our previous paper [2], we compared the convex hull approach to several statistical approaches with favorable results. During that work we selected metrics based on our intuition. This extends that work and attempts to find a better set of metrics using search based techniques.

III. Search Techniques

The search library used in this work was implemented using the Ruby programming language [17]. While it was initially developed as a GP library, it supports GA and random search as special cases of the GP.

The description of the library used to find a solution is separated into the following parts: Section III-A presents the chromosome representation; Section III-B describes how the initial population is created; Section III-C presents the process of evaluating a population of chromosomes; finally Section III-D describes how new populations are generated.

A. Chromosome Representation

Figure 2 illustrates the chromosome representation as a tree structure. Each chromosome represents a function that

```
TERMINAL => {1|2|3|...|21}

   UEXP => {- EXP}

   BEXP => {- EXP EXP|+ EXP EXP}

    EXP => {UEXP|DEXP|TERMINAL}

      S => {s EXP EXP EXP EXP
            EXP EXP EXP EXP}
```

Figure 3. The grammar used to generate the initial random population. Generation begins with the `S` rule and proceeds until all branches end with a `TERMINAL` node.

accepts a set of 21 sensor measurements as input and produces a set of 8 metrics as output. The tree is constructed using one of four nodes. The `root node` has 8 branches, one for each metric. `Binary operation nodes` can be either addition or subtraction. `Unary operation nodes` negate the subtrees linked from them. `Terminal nodes` contain the `ID` value of a sensor.

Using the tree structure to represent the entire solution, as opposed to each metric individually, allows the algorithm to consider how metrics are related to each other. Simply finding the best eight metrics independent of each other does not lead to good results because many of the metrics are dependent and tend to have similar classification effectiveness.

Note that if the `root node` is forced to decompose directly into `terminal nodes`, the problem changes from a GP trying to evolve 8 metric formulas to a GA trying to find the optimal subset of 8 sensors.

B. Creating the Initial Population

The initial population is created using the grammar [18], [19] of prefix expressions presented in Figure 3. The production begins with the `root node`, which is expanded into 8 `EXP` nodes. When expanding a node with several options, one of the options is chosen at random. Once all of the productions are expanded into `TERMINALS`, the generated abstract syntax tree (AST) is simplified into the chromosome format, as demonstrated by Figure 2.

It has been shown that restricting the tree height of the initial chromosomes can lead to good convergence rates [20]. Therefore, the initialization accepts in a parameter k that specifies the maximum height of a random tree. When a branch of the tree being generated reaches a height k, the generation algorithm automatically chooses a `terminal node`. Because each production, except `EXP`, is expanded into an `EXP`, which has a production `TERMINAL` as one of its options, the generated AST is guaranteed to have a maximum height of $k + 1$.

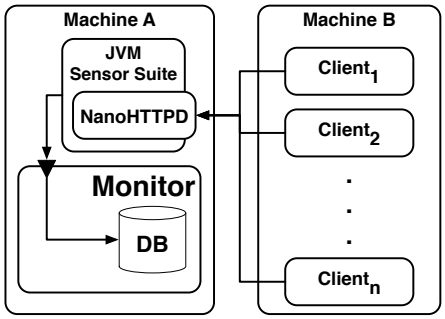


Figure 4. The experimental testbed contains two machines. The first is used to run NanoHTTPD and the autonomic framework. The second houses clients that access NanoHTTPD.

C. Evaluating a Population

A chromosome represents a function that translates a point in 21-dimensional space into a point in 8-dimensional space. Each chromosome is evaluated based on how well it can classify data collected during the execution of NanoHTTPD as normal or anomalous. To perform this evaluation three data sets are needed. The first set, labeled as the `training set`, is a set of points used to construct a convex hull that represents normal execution. The second set, labeled as the `normal set`, is a set of points collected during the normal execution of NanoHTTPD. These points are not part of the `training set` and are used to determine the false positive rate. A false positive occurs when a point known to have been sampled during the normal execution of NanoHTTPD is outside of the convex hull and is therefore falsely classified as an anomaly. The third set, labeled as the `error set`, is a set of points collected during the faulty execution of NanoHTTPD. These points are used to determine the false negative rate. A false negative occurs when a point known to have been sampled during the faulty execution of NanoHTTPD is inside of the convex hull and is therefore incorrectly classified as normal. The `normal` and the `error` sets are divided into two subsets. The first is used during the training phase and second is used to evaluate the final result.

To create the three data sets described above, a case study using NanoHTTPD was conducted. Figure 4 presents the design of the testbed used in this case study. One machine is used to host NanoHTTPD and Aniketos, our autonomic computing framework. Another machine manages clients that request resources from NanoHTTPD. A JVM sensor suite is used to monitor NanoHTTPD's execution and report measurements to the monitoring server. The

monitoring server stores the gathered data and processes it using the QHull library [15].

NanoHTTPD was chosen for this case study because it is open source and manageable in size, thus making it easy to modify and inject with faults. It is a web server that hosts static content. NanoHTTPD spawns a new thread for each HTTP client request. If a thread crashes or goes into an infinite loop, it does not compromise NanoHTTPD's ability to serve other files.

The goal of this case study is to replicate realistic operating conditions. To this end, it uses resources and access patterns of the Drexel University Computer Science Department website. Nine weeks worth of website access logs were collected and all resources accessed in those logs were extracted and converted into static HTML pages. Out of the nine weeks of logs, three weeks were chosen at random to be used in the case study. These were replayed against the statically hosted version of the website and provided the case study with realistic workload access patterns.

One week of request logs was used to create the `training set`. Another week of request logs was used to create the `normal set`. A third week of request logs was used as background activity during the fault-injection experiments, thus creating the `error set`. All measurements were normalized such that each metric's value ranged between 0 and 100.

NanoHTTPD was injected with nine faults. These faults represent coding errors, security vulnerabilities, and attacks. Two of the most common coding errors are the infinite-loop and the infinite-recursion faults. An infinite-loop fault is presented as a `while` loop that iterates indefinitely. Two versions of this fault were created. One, where each iteration of the loop does nothing, and the second, where each iteration of the loop performs a `sleep` operation for 100ms. The goal of the slow-infinite-loop fault is to create a more realistic scenario during which an infinite loop is not overwhelming the CPU. Similar to the infinite-loop fault, the infinite-recursion fault also has two versions, a regular and a slow one that performs a `sleep` operation for 100ms. An infinite recursion is presented as a function calling itself until the thread running it crashes due to a `StackOverflowError` exception.

Another fault injected into NanoHTTPD was the memory-leak fault. Two versions of this fault were created. The first version performed a realistic memory leak and leaked strings containing the requested URLs by adding them to a `vector` stored in memory. The second version of the memory-leak fault doubled the size of the leak `vector` with each request.

Log explosion [21] is another problem common to the server environment and was injected into NanoHTTPD. The log-explosion fault causes NanoHTTPD to write to a log file continuously until there is no more space left on the hard drive. While this does not cause the web server to crash, the log-explosion fault does cause a degradation in the performance of the server.

In addition to faults due to coding errors, two security attacks were perpetrated against NanoHTTPD. In the first, NanoHTTPD was injected with a spambot trojan [22]. Once triggered, the spambot began to send spam email message packets to an outside server at a rate of 3 email messages per second. Each message was 1 of 3 spam messages chosen at random. The messages varied in length between 166 and 2325 characters each.

The second attack perpetrated against NanoHTTPD was a denial of service (DOS) attack [23]. During the DOS attack several processes that continuously requested resources from NanoHTTPD were launched from two separate machines. The attack escalated with about 100 new processes created every second. It continued until NanoHTTPD could no longer process valid user requests.

Each fault was triggered by a specific URL request. For example, the memory leak fault was triggered by accessing the "`/memleak`" resource via a web browser. Faults were triggered after a minimum of 1 minute of fault-free execution.

The entire case study took about three weeks to execute. Most of that time was spent executing fault-free experiments and the memory-leak experiment. While all other fault experiments took only a few minutes, the memory-leak fault took several hours before NanoHTTPD failed irreversibly.

Given these 3 data sets, the fitness of the chromosome was calculated using the following function:

$$f(x) = \frac{a}{b} + \left(1 - \frac{c}{d}\right)$$

Where x is the chromosome being evaluated, a is the number of points from the `normal set` classified as inside the hull, b is the total number of points in the `normal set`, c is the number of points from the `error set` classified as inside the hull, and d is the total number of points in the `error set`. The function will return a maximum value of 2.0 if every point is classified correctly and a minimum value of 0.0 if every point is misclassified. Note that this fitness function weights the false-positive and false-negative rates equally.

Using the error data in selecting a set of metrics alters the goals of Aniketos slightly. While this does not completely transform it from an anomaly detection approach to a signature based approach, it certainly uses the faults as a guide. As one of the goals of this work was to evaluate the performance of expertly chosen metrics, this modification is acceptable.

Once collected, the three data sets had to be randomly subsampled because at their original size the QHull library took too long to compute the convex hull. The original `training set` contained 604,800 sample points, and was randomly subsampled down to 18,000. This dramatically decreased the hull computation time from several

hours to several minutes in the average case. Note that a new hull has to be computed for each chromosome in a population. Therefore, without subsampling, an application of search algorithms would have been infeasible.

However, even when using a reduced set, the time to compute the convex hull varied from a few seconds to several minutes. Therefore, any evaluation that took over 10 minutes was stopped and that chromosome was given a fitness value of 0. This effectively bred out slow chromosomes and consequently the evaluation sped up with each generation.

To further speed up the evaluation process it was distributed over several computation nodes. Five servers with 16 cores each were available for our use. Because the servers are a shared resource, the number of available computation nodes varied between 30 and 80. At its peek performance, the algorithm was able to evaluate a generation every nine minutes, on average.

Because the QHull library was written in `C`, the evaluation function was wrapped in a `C` program that took the chromosome and 3 data sets as input and produced the value of the chromosome as the output. The main controller was implemented in `Ruby` and used the provided `SSH` library to distribute the work across several machines.

The `Ruby` process was initialized with the required information to form an SSH connection to each of the computation nodes. Then, during evaluation, a pool of available nodes was maintained. To evaluate a chromosome, a node from the pool was chosen. Given the node, Ruby created a `green thread`, meaning a thread managed by the `Ruby interpreter`, and used the `SSH` library to connect to the machine that hosted the computation node. Once the connection was made, the `Ruby` process created a new process on that node and executed the `C` program used to evaluate a chromosome. The process was set to timeout after 10 minutes. When the computation was complete, or stopped due to a timeout, the reference to that node was placed back into the pool to be reused.

In addition to the time constraints, not all chromosomes produced a `training set` that could be turned into an 8 dimensional hull. The `QHull` library expects the data to have dimensionality, meaning that the produced hull is expected to have a non-zero width in every dimension. Given the nature of the produced solution it is likely that a chromosome may reduce the original 21-dimension `training set` so that the 8-dimension training set breaks the dimensionality constraint. This occurs when a metric has no width, or two or more metrics are correlated with one another. Any chromosome that produced a set that could not be used to create a convex hull was replaced by a new one. The new chromosome was computed randomly if it replaced a chromosome during the first generation and bred if it replaced a chromosome during subsequent generations.

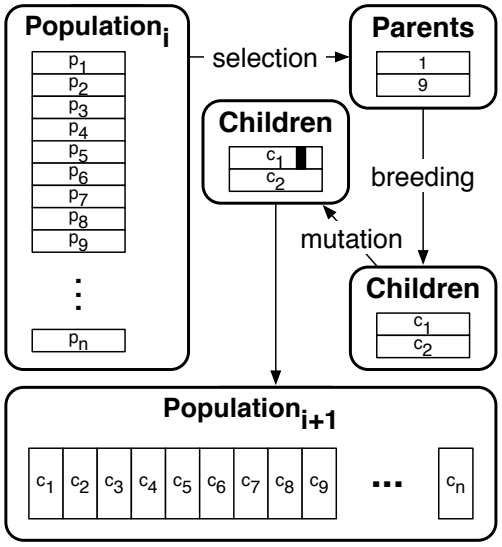


Figure 5. The breeding process to produce a new ($i + 1$) population. Parents are selected via roulette wheel selection and bred 95% of the time. The children chromosomes are then mutated 1% of the time before being placed into the next generation.

D. Generating a New Population

A new generation of 100 chromosomes is bred via the process in Figure 5. Parents are chosen via roulette wheel selection. A fitter chromosome will have a higher probability in being selected. Once selected and bred, chromosomes are placed back into the population so that they can be selected again.

Two selected chromosomes are bred with a probability of 95%, otherwise they are passed into the new population unchanged. The breeding function is a modified single cross-over function. Given that the `root node` points to 8 subtrees, if a traditional single cross-over function was used, most of the changes would occur in the metrics with many branches, leaving metrics that contained a single sensor unmodified. For example, consider a chromosome in which 6 out of 8 metrics led to a single sensor, while the other two had complicated functions with 20 unique branches each. In this case, the probability of choosing a cross-over point that modified one of the 6 single sensor metrics would be only 12%.

The modification added to the single cross-over function is that two metrics, one from each chromosome, are first chosen at random. These represent one of 8 subtrees directly under the `root` node. Once the subtrees are selected a cross-over operation is performed using those subtrees. The links from the `root node` to either of the subtrees are considered during the cross-over, therefore an

		Training Sets			
		1	2	3	average
Arithmetic	score	1.990	1.993	1.988	1.990
	false positive	1.0%	0.6%	1.2%	0.9%
	false negative	0.0%	0.1%	0.1%	0.1%
Subset Selection	score	1.992	1.987	1.992	1.990
	false positive	0.8%	1.3%	0.8%	1.0%
	false negative	0.0%	0.0%	0.0%	0.0%
Random Arithmetic	score	1.987	1.962	1.971	1.973
	false positive	1.3%	3.8%	2.9%	2.7%
	false negative	0.0%	0.0%	0.0%	0.0%
Random Subset Selection	score	1.973	1.987	1.986	1.982
	false positive	1.4%	1.3%	1.4%	1.4%
	false negative	1.3%	0.0%	0.0%	0.4%
Handcrafted$_1$	score	1.837	1.918	1.870	1.875
	false positive	4.0%	3.7%	4.1%	3.9%
	false negative	12.4%	4.5%	9.0%	8.6%
Handcrafted$_2$	score	1.682	1.683	1.683	1.683
	false positive	0.3%	0.6%	0.3%	0.4%
	false negative	31.4%	31.2%	31.4%	31.3%

Table I. The fitness of the best chromosomes discovered using various search techniques. Handcrafted$_1$ is the result when using 7 metrics provided by a software engineer. Handcrafted$_2$ is the result when using 6 metrics provided by a system administrator.

entire tree can be exchanged.

Once the new chromosomes are generated, they are mutated with the probability of 1%. If a mutation operation occurs, a node in the chromosome is chosen at random and modified. If the selected node is a `binary operator node`, it is replaced by a new `binary operator node` chosen at random and based on the provided grammar. If the selected node is a `negation operator`, the node is removed from the tree. If the selected node is a `terminal node`, it is replaced by a new `terminal node` chosen at random and based on the grammar.

IV. Search Algorithms Evaluation

Table I illustrates the results of several search experiments and compares these results to two handcrafted solutions. Due to the time constrains described in Section III, the training set had to be sub-sampled. In order to verify the results, three different sub-sampled sets were used in these experiments. Each training set was generated by choosing 18,000 points randomly from the full training set.

Four different search experiments were conducted as part of this work. All experiments were stopped after 12 hours of execution and the best solution found by each is presented in Table I. The first, labeled `Arithmetic` in

```
TERMINAL  =>  {1|2|3|...|21}

     EXP  =>  {TERMINAL}

       S  =>  {s EXP EXP EXP EXP
               EXP EXP EXP EXP}
```

Figure 6. A modification of the original grammar from Figure 3, changes the problem from a GP to a GA.

Software Engineer
Total CPU Time
Thread Count
TCP Bytes Read + TCP Bytes Written
Number of Open TCP Accept Sockets
Loaded Class Count
Heap Memory + Non-heap Memory
Average Stack Depth

System Administrator
Total CPU Time
Thread Count
TCP Bytes Read + TCP Bytes Written
Number of Open TCP Accept Sockets
Number of Closed TCP Accept Sockets
Daemon Thread Count

Figure 7. Metrics developed by the domain experts. The software engineering expert is one of the the authors of this paper. The system administrator expert is the manager of the computing resources for Drexel University's Computer Science department.

Table I finds the solution via the GP search described in Section III. The second, labeled `Subset Selection`, solves the subset selection problem using the same algorithm as the GP with a different grammar. The grammar is altered such that the root node expands into 8 terminal nodes, thus changing the problem from a GP to GA. This new grammar is presented in Figure 6. The third experiment, labeled `Random Arithmetic`, creates several random chromosomes via the process described in Section III-B and simply selects the fittest chromosome as the solution. The fourth experiment, labeled `Random Subset Selection`, creates several random chromosomes using the same grammar used by the `Subset Selection` experiment (Figure 6). Results using two different handcrafted chromosomes are also presented in Table I.

This work began by having two domain experts, a software engineer and a system administrator, choose up to 8 metrics from a list of 21 sensors, presented in Figure 7.

	3D	4D	5D
score	1.855	1.956	1.988
false positive	2.6%	0.8%	1.2%
false negative	11.9%	3.6%	0.0%

Table II. The best fit subset of 3, 4, and 5 sensors found via exhaustive search.

Notice that while there is some overlap between these two sets of metrics, they were created from two different perspectives. After discussing the selection with the system administrator, it was clear that he was far more concerned with how a failure in the web server would affect all other services he was managing. Therefore, he focused on metrics that expressed server load. The software engineer was more concerned with detecting failures internal to NanoHTTPD. Therefore, he chose metrics such as `Average Stack Depth` and `Total Memory`.

Given that the metrics chosen by the domain experts varied significantly in their results, we decided to automate the metric selection process. Because both experts chose a combination of sensors for at least one of their metrics, the GP search approach was chosen first.

When running the GP process we were surprised by how well it performed. The best GP result was found after only twelve generations. This led us to speculate that using the GP may be excessive and not necessary for finding a good solution. It appears that the problem in not a "needle in a haystack" type problem that GPs are so good at solving. Therefore, we decided to simplify the problem to subset selection.

The best subset selection solution was found during the 36^{th} generation. However, good solutions appeared as early as the 8^{th} generation. This led us to further explore the search space via random search.

While random search was not able to find solutions that were as good as those found via the guided search techniques, it was able to get surprisingly close. Upon further examination, roughly 0.3% of the solutions explored by random search had a value greater than 1.95. Each random search experiment explored about 3,500 solutions.

Having observed such good results using random search, we wanted to further investigate the search space. To that end, we conducted three exhaustive-search experiments. Taking a single subsample as the training set, we computed the value of every possible combination of 3, 4, and 5 metrics. More metrics would be impractical to compute. The best solutions for each are presented in Table II. Histograms demonstrating the break down of all solutions are presented in Figure 8. As expected, higher dimensions provided better results. However, the

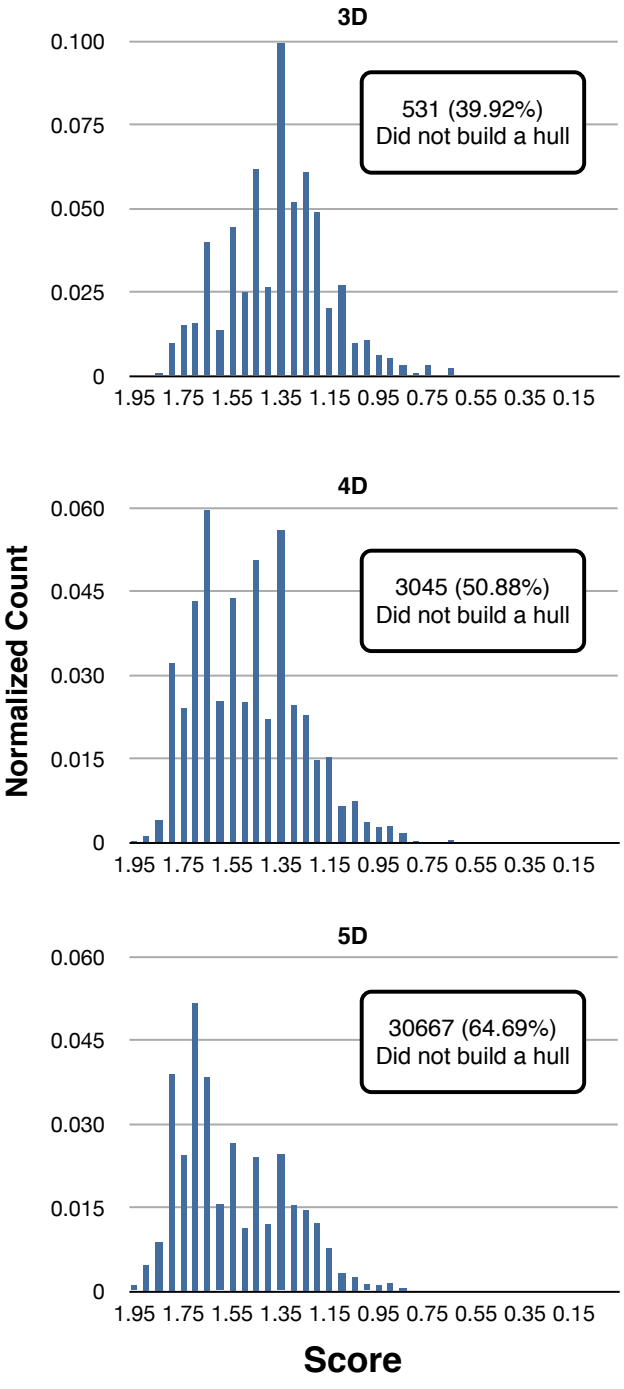


Figure 8. Normalized histograms showing the values of all subsets of 3, 4, and 5 sensors.

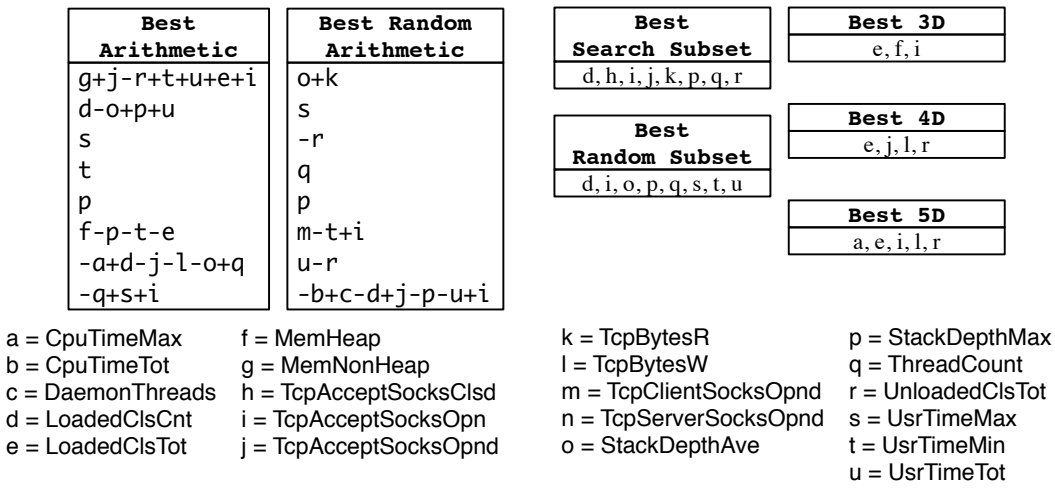


Figure 9. The most fit solutions found by the various search algorithms discussed in this paper.

high quality of the results when using only 5 metrics was a surprise. Since 5 metrics produced such good results it is not surprising that the search algorithms were able to find equally good solutions in 8 dimensions.

In addition to comparing the quality of the results it is also interesting to compare the best metrics chosen during each experiment. Figure 9 summarizes the best solutions found in each case. While several of the sensors can be observed in most solutions, there is no single sensor that stands out as the most important. This is probably because some sensors are correlated with one another and therefore can be used interchangeably. For example, each new request causes NanoHTTPD to start a new thread and open a new TCP socket. Therefore, the `ThreadCount` and `TcpAcceptSocksOpn` sensors are highly correlated.

These experiments where repeated with `training sets` as small as 500 points. In all cases the results were analogous to those reported here. However, these experiments took 4 hours, as opposed to 12 hours, to search through as many solution as the experiments using training sets of 18,000 points.

V. Conclusions and Future Work

This paper presents a search-based approach for deriving a set of metrics from a larger set of sensors. These metrics are intended to be used in a computational-geometry technique for run-time fault detection in an autonomic computing framework. The technique treats run-time measurements as points in n-dimensional space and constructs a convex hull to represent the normal execution of the application being monitored. Then, during fault detection, it categorizes new points as anomalous when they fall outside of the convex hull. This technique is limited by the QHull library to 8 metrics.

A case study using a Java-based web served name NanoHTTPD was conducted. During the case study, search-based techniques found solutions that were superior to those chosen by the domain experts. However, the solutions were found with relatively little computation and, upon further exploration, it became evident that many good solutions exist in the space being searched. This is best observed by the fact that an exhaustive search of every 5 sensor combination was able to find a solution equal to those found by the various search techniques described in this paper.

This observation can be explained because many of the sensors are dependent with one another and are therefore redundant. This is likely to be a consequence of the simplicity of NanoHTTPD. NanoHTTPD is a bare-bones web server that does not have any advanced functionality for managing its resources.

The results presented in this paper are promising as they demonstrate the relative effectiveness of search-based software-engineering techniques as compared to solutions proposed by domain experts. Even using a system as simple as NanoHTTPD, the computer significantly outperformed the domain experts. The search algorithms were able to discover non-obvious esoteric relationships between sensors. We expect the difference between the quality of human and computer-generated solutions to increase as more sophisticated applications begin using this technique.

In future work we will test this hypothesis by employing the techniques presented in this paper on industrial-strength software systems. These will include more complex web servers that perform some form of resource

management, application servers such as TomCat [24] and GlassFish [25], as well as standalone desktop applications.

It is also worth noting that the fitness function used in this work weighs the false-positive and false-negative rates equally. While this may be useful in the general case, there are scenarios in which a failure may be more costly than false fault detection. In such scenarios the fitness function would need to be adjusted to take that cost into account.

Acknowledgments

The authors would like to thank the Lockheed Martin Corporation for sponsoring this research.

References

[1] T. Sweeney, "No Time for DOWNTIME – IT Managers feel the heat to prevent outages that can cost millions of dollars," *InternetWeek*, vol. 807, 2000.

[2] E. Stehle, K. Lynch, M. Shevertalov, C. Rorrest, and S. Mancoridis, "On the use of Computational Geometry to Detect Software Faults at Runtime," in *Proceedings of the 7th IEEE/ACM International Conference on Autonomic Computing and Communications*, June 2010.

[3] J. Elonen, "NanoHTTPD," 2007, http://elonen.iki.fi/code/nanohttpd/.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] Y. Al-Nashif, A. Kumar, S. Hariri, Y. Luo, F. Szidarovsky, and G. Qu, "Multi-Level Intrusion Detection System (ML-IDS)," in *Autonomic Computing, 2008. ICAC'08. International Conference on*, 2008, pp. 131–140.

[6] McAfee, "McAfee-Antivirus Software and Intrusion Prevension Solutions," http://www.mcafee.com/us/.

[7] Symantec, "Symantec - AntiVirus, Anti-Spyware, Endpoint Security, Backup, Storage Solutions," http://www.mcafee.com/us/.

[8] M. Roesch, "Snort - lightweight intrusion detection for networks," in *LISA '99: Proceedings of the 13th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.

[9] G. Vigna and R. A. Kemmerer, "Netstat: a network-based intrusion detection system," *J. Comput. Secur.*, vol. 7, no. 1, pp. 37–71, 1999.

[10] M. Brodie, S. Ma, G. Lohman, L. Mignet, M. Wilding, J. Champlin, and P. Sohn, "Quickly finding known software problems via automated symptom matching," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, June 2005, pp. 101–110.

[11] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 16–16.

[12] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, June 2006, pp. 259–268.

[13] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward, "System monitoring with metric-correlation models: problems and solutions," in *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*. New York, NY, USA: ACM, 2009, pp. 13–22.

[14] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt, "Self-correlating predictive information tracking for large-scale production systems," in *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*. New York, NY, USA: ACM, 2009, pp. 33–42.

[15] C. Barber and H. Huhdanpaa, "Qhull," *The Geometry Center, University of Minnesota, http://www.geom.umn.edu/software/qhull*.

[16] D. Goldberg, *Genetic Algorithms in Search and Optimization*. Addison-wesley, 1989.

[17] D. Flanagan and Y. Matsumoto, *The ruby programming language*. O'Reilly Media, Inc., 2008.

[18] J. Koza and J. Rice, *Genetic programming*. Springer, 1992.

[19] P. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, vol. 16, no. 3. Citeseer, 1995, pp. 33–41.

[20] M. Garcia-Arnau, D. Manrique, J. Rios, and A. Rodriguez-Paton, "Initialization method for grammar-guided genetic programming," *Knowledge-Based Systems*, vol. 20, no. 2, pp. 127–133, 2007.

[21] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, "Experiences building planetlab," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006, pp. 351–366.

[22] D. Barroso, "Botnets–The Silent Threat," *European Network and Information Security Agency (ENISA)*, 2007.

[23] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, p. 139, 2006.

[24] J. Brittain and I. Darwin, *Tomcat: the definitive guide*. O'Reilly Media, Inc., 2003.

[25] D. Heffelfinger, *Java EE 5 Development using GlassFish Application Server*. Packt Pub Ltd, 2007.

Search-based resource scheduling for bug fixing tasks

Junchao Xiao
Laboratory for Internet Software Technologies,
Chinese Academy of Sciences, Beijing 100190, China.
xiaojunchao@itechs.iscas.ac.cn

Wasif Afzal
Blekinge Institute of Technology,
PO Box 520, SE-372 25 Ronneby, Sweden.
wasif.afzal@bth.se

***Abstract*—The software testing phase usually results in a large number of bugs to be fixed. The fixing of these bugs require executing certain activities (potentially concurrent) that demand resources having different competencies and workloads. Appropriate resource allocation to these bug-fixing activities can help a project manager to schedule capable resources to these activities, taking into account their availability and skill requirements for fixing different bugs. This paper presents a multi-objective search-based resource scheduling method for bug-fixing tasks. The inputs to our proposed method include *i)* a bug model, *ii)* a human resource model, *iii)* a capability matching method between bug-fixing activities and human resources and *iv)* objectives of bug-fixing. A genetic algorithm (GA) is used as a search algorithm and the output is a bug-fixing schedule, satisfying different constraints and value objectives. We have evaluated our proposed scheduling method on an industrial data set and have discussed three different scenarios. The results indicate that GA is able to effectively schedule resources by balancing different objectives. We have also compared the effectiveness of using GA with a simple hill-climbing algorithm. The comparison shows that GA is able to achieve statistically better fitness values than hill-climbing.**

I. Introduction

Software bugs correspond to mistakes by the programmers due to an incorrect step, process, or data definition. One estimate is that a professional programmer is responsible for 5 bugs per 1000 lines of code (LoC) written on average [1]. This might not be the case with every software application but there are always a certain number of bugs in almost every software application that causes incorrect results.

Software testing is one major bug finding activity that improves software quality to a certain extent before the software application is released to the end-users. As bugs are reported, they must be triaged in a cost-effective manner, considering the resources and the requirements of bugs. Triage of bugs in a cost-effective manner is an important decision-making task whereby competing objectives of technical, resource and budget constraints need to be balanced to provide maximum business value for the organization.

Anvik et al. [2] report that 3426 reports were submitted to the bug database of Eclipse open source project between Jan. 1, 2005 to Apr. 30, 2005, averaging 29 reports per day. Assuming that it takes 5 minutes to triage a report, this activity costs 2 person hours per day. This indicates that we have a need to support efficient and effective bug-assignment policies that can schedule different bug-fixing tasks by taking into account the available resource constraints and bug requirements. Laplante and Ahmad [3] further emphasize the value of having an efficient and effective bug assignment policy: "Bug assignment policies can affect customer satisfaction, employee behavior and morale, resource use, and, ultimately, a software product's success". But triaging of bugs for repairing is fraught with challenges since the number of problem variables is diverse, e.g. the severity and priority of a bug has to be balanced with resource skills and availability for finding a reasonable bug-fix schedule.

Optimal resource scheduling for bug fixing is an example of resource constrained scheduling problem [4]. The scheduling problem in general is NP-hard, i.e., finding optimal solutions in polynomial time is hard [4], [5]. This is because the search-space becomes vast as problem size increases or more constraints are added. These properties naturally make scheduling problems a suitable problem domain for evolutionary computation approaches like genetic algorithms.

In this paper we attempt to (at least approximately) formalize the problem of appropriately scheduling developers and testers to bug-fixing activities, keeping in view the capabilities of resources and requirements of bugs. We therefore seek an answer to the following research question:

RQ: How to schedule developers and testers to bug-fixing activities taking into account both human properties (skill set, skill level and availability) and bug characteristics (severity and priority) that satisfies different value objectives by using a search-based method such as GA?

The contributions of this paper are: *i)* It presents models for bugs and human resources that form the basis for scheduling resources for bug-fixing. *ii)* It presents a method of capability matching between the bug-fixing activities and human resources as well as how to evaluate an organization's value of a bug-fixing process. *iii)* It provides a bug-fixing scheduling method by using a GA and discusses several scenarios of practical use.

The paper is organized as follows. Section II describes some related work. Section III presents the basis of scheduling resources for bug-fixing. It specifies models of bugs and human resources. Section IV discusses the design of

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.24

the scheduling method using a GA. Section V presents the industrial data used while results of applying the proposed method are given in Section VI. The GA approach is compared with hill-climbing search in Section VII. Section VIII presents a discussion while the Section X presents conclusions and suggests future work.

II. Related work

Search techniques have been successfully used to solve different scheduling related software project management problems [6], such as software project planning [7], [8], [9], [10], software project effort prediction [11] and software fault prediction [12]. Hart et al. [13] have written a review on evolutionary scheduling. However, the application of search techniques for implementing an efficient bug repair policy is very much unexplored.

A study by Mockus et al. [14] predicted defect effort schedule based on observed new feature changes. They fitted a probability model to the observed data from 11 releases of a large real-time high availability software system and found the predicted effort to be close to reality. Cubranic and Murphy [15] applied naive Bayes classifier to automatically assign the bug reports to developers and achieved a 30% classification accuracy for reports entered into Eclipse's bug tracking system between Jan. 1, 2002 and Sep. 1, 2002. Zeng and Rine [16] used a self-organizing neural network approach to estimate defects fix effort. A feature map having different clusters was created after training the weights of the self-organizing neural networks. They computed the probability distributions of effort from the clusters and then compared them with those from the test set. For projects having similar development environments the approach gave acceptable performance with average mean relative error (MRE) values between 7% to 23%. Canfora and Cerulo [17] used a probabilistic text similarity approach to assign change requests to developers. Song et al. [18] presented association rule mining based methods to predict defect correction effort. Using data from more than 200 projects, their approach was found to be better than partial regression trees (PART), C4.5 and naive Bayes. Anvik et al. [2] applied support vector machine algorithm as a text categorization technique to suggest assignment of a new bug report to a small number of developers. Precision levels of 57% and 64% were obtained for the Eclipse and Firefox development projects. Recently, Weiss et al. [19] used a text similarity technique to predict bug-fixing effort based on title and description of bug. Their approach beat the naive approach using the defect data from the JBoss project.

This paper differs from related work in some important ways. Since software development is human-dependent, this work incorporates human factors such as competencies and available time-slots to schedule resources for bug-fixing. This is done by using models for the bugs and human-resources; moreover the use of a search-based technique such as GA is presented that can bring a near-optimal value in scheduling by balancing multiple competing objectives.

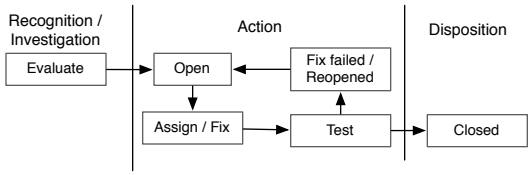


Figure 1. The bug-fixing process.

III. The bug fixing process

A typical bug-tracking system such as Bugzilla [20] keeps track of a reported bug through assigned status. So the bug is marked "new" when reported, "assigned" when assigned to a developer for fixing, "verified" when testing of the bug fix is done and "resolved" when the bug is closed. This is in line with the anomaly (bug) classification process proposed by the IEEE standard classification for software anomalies (IEEE Std 1044-1993) [21], whereby the bug life cycle is divided in to four steps: *i)* Recognition, *ii)* Investigation, *iii)* Action and *iv)* Disposition. If we assume that a bug is valid (i.e. it is not a duplicate, not incomplete/needing more information and therefore is required to be fixed), the following events describe one instance of the above four steps in more detail (also shown in Figure 1):

1) A new bug is reported in the bug database which has been evaluated as a valid bug.
2) The bug is assigned to a developer for fixing.
3) The developer fixes the bug.
4) The bug is assigned to a tester for verification.
5) The bug is verified and closed or alternatively is reopened due to an incorrect fix.

We have restricted the scope of this paper to schedule resources for a single round of these five events. So if a bug-fix from a developer fails at testing and is re-opened, a second round of events need to be taken, however, this is not dealt with in this paper since it is not known in advance how many of these round of activities would be required.

It is clear from the different bug life cycle events that given a number of bugs reported in the bug data base, there are two resource consuming activities taking place: development activity for fixing bugs and testing activity for verifying these bug-fixes. The criticality and resource demands for various bugs require resources with desired competencies and skills; moreover this has to be balanced with availability/ workload of resources for getting the job done. Due to all that common constraint of limited resources to play with and engagement of resources in multiple projects concurrently, the bug-fixing events are in a contention for finding resources that have the availability and competence to fix/ verify reported bugs. To schedule the capable and available

resources, to balance the competing objectives and to bring near-optimal value by using scheduling, we need a degree of formalism to describe the reported bugs and required human resources. This is done by describing a bug and a human resource model.

A. The bug model

This paper only focuses on the bugs found during system testing. This is however more of a constraint rather than a rule and our proposed methodology should be equally applicable to bugs found at other testing levels.

We define a bug data repository, BR, as a collection of reported bugs, $BR = \{B_1, B_2, \ldots, B_n\}$, where each bug B_i in this repository has the following attributes:

1) *Bug ID:* A unique identification of the bug.
2) *Bug description:* A short description of the bug.
3) *Bug severity:* The perceived impact of the bug, having possible values of High, Medium and Low.
4) *Bug priority:* A classification indicating the order in which the bugs are addressed, having possible values of High, Medium and Low.
5) *Required skills:* The skills required for bug-fixing, which are used to select the candidate resources. Each required skill is described as a triplet (SKT, SKN, SKL), where
   a) SKT: The type of required skills, which in our context are two, namely development skills and testing skills.
   b) SKN: The name of a specific required skill, e.g. , programming language skills for the skill type: development and test design skills for the skill type: testing.
   c) SKL: The minimum required competency in a particular skill for fixing/verifying the bug, having possible values of Low, Medium and High.

   It is to be noted that the definition of skills required to fix/verify a bug would be different across software companies, however the skill structure defined above is flexible to incorporate different skill types.
6) *Estimated effort for fixing the bug:* The estimated required effort, in number of person-hours, to be invested in development and testing of the bug-fix. Note that this estimated required effort is for one round of events, as described in Section III.
7) *Assigned time:* The date when the bug-fixing activity can be started.
8) *Deadline for bug-fixing:* The date by which the bug has to be fixed and verified. This attribute is used as a constraint for scheduling.
9) *Actual bug fixing time:* The actual date when the bug fixing is finished. This includes both the actual development and the actual testing time taken by a bug for resolution. This is given by the scheduling results.
10) *Coefficient of schedule benefit (CSB):* If the actual bug fixing time is before the deadline for bug-fixing, there is an incurred benefit given by CSB which is described by the benefit for each day before the deadline.
11) *Coefficient of schedule penalty (CSP):* If the actual bug fixing time is later than the deadline, it is expected that some other work activity would get affected. Thus there is a penalty, CSP, involved in this case for each day used up later than the deadline.

    The values for the coefficients CSB and CSP are configurable parameters chosen accordingly by the stakeholders. For instance, the penalty in missing a deadline might have higher impact than the benefit in fixing the bug before deadline; so CSP might get a higher stakeholder value than CSB.

From the above attributes we can determine the value of each bug based on the following value function:

$$Value(B_i) = f(B_i.priority, B_i.severity, B_i.deadline, B_i.actual\_fixing\_time, B_i.CSB, B_i.CSP)$$

Among these parameters, actual bug-fixing time is decided by the scheduling results and the others are determined by the value objectives of stakeholders before the scheduling. The summation of the values of all the bugs gives us the overall value of the bug-fixing process:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

B. The human resource model

The human resource model captures the competencies and availability of development and testing personnel to undertake bug-fixing/verification. We make use of the human resource model proposed by Xiao et al. [22] to describe different attributes of human resources in the bug-fixing process. A human resource, HR, is defined in terms of four attributes:

1) HR ID: A unique identification of the human resource.
2) SKLS: The set of skills possessed by a human resource, $SKLS = \{skl_1, skl_2, \ldots, skl_n\}$. Each skl_i $(1 \leq i \leq n)$ is defined by a triplet as $skl_i = (SKT, SKN, SKL)$. The elements in this triplet are the skill type (SKT), skill name (SKN) and skill level (SKL). For example, an experienced testing resource (SKT) might be highly competent (SKL) in a certain test design technique (SKN).
3) EXPD: The work experience figure, in number of years, for the human resource. This figure can given an indication of the skill level of a particular resource.
4) STMW: The time and the workload that can be scheduled for a human resource. STMW consists of

all free time periods and the workload per day in each of these time periods:

$$STMW = \{([T_{s1}, T_{e1}], W_1), ([T_{s2}, T_{e2}], W_2), \ldots, ([T_{sk}, T_{ek}], W_k), \}$$

where T_{si} and T_{ei} represent the start and end date of the i^{th} free time period respectively, W_i represent the workable hours per day that can be scheduled in the i^{th} free time period. The unit for W_i is person hour. For example, $([25-Mar-2010, 07-Apr-2010], 6)$ indicates that the resource is available between $25-Mar-2010$ and $07-Apr-2010$ for 6 hours per day, excluding the weekends.

If a human resource has all the skills required for fixing a bug, then depending upon the available time periods, this human resource can be scheduled for the bug-fixing task. Thus according to the human resource descriptions and skill requirements of bugs, the capable resources for each bug-fixing event/activity can be scheduled.

However the organizations might lack the required competencies for fixing certain bugs within the stipulated deadline. For example, if the verification of a bug-fix requires a high skill level of domain knowledge on part of the testing resource but the one available has medium or low skill levels. In such a scenario, we setup certain rules aimed at relaxing the skill requirements in order to provide additional candidate capable resources for bug-fixing. Thus the organization can take a risk of lowering the skill requirements in an attempt to close a bug on deadline. Following are the rules to relax the skill requirements and provide additional candidate capable resources for bug-fixing if:

- the skill level gap between what is required and what is available is less than a specific number such as '1'. This number indicates the scale of gap, so e.g. the gap is '1' if there is a requirement of high level of a certain skill but only a medium one is available.
- the number of skills possessed by resources, having levels lower than the requirement, is less than a given value, e.g., 3, that is, at most a resource is lacking in 3 skill levels than what is the requirement.

IV. Scheduling with a genetic algorithm (GA)

Scheduling resources for bug-fixing activities represent a problem with different competing constraints and even with a moderate number of bugs, the search space can become vast as the number of combinations grow. To deal with the complexity of such a combinatorial optimization problem we apply a genetic algorithm (GA) to achieve maximum possible value out of the bug-fixing process.

GA is an evolutionary algorithm that uses simulated evolution as a search strategy to evolve potential solutions and uses operators inspired by genetics and natural selection [23]. A GA encodes the candidate solutions to the search problem as finite length strings called chromosomes. The chromosomes are made up of components called genes while the values of these genes are called alleles. A fitness measure discriminates good candidate solutions from bad ones and guides the search towards feasible areas in the search space. A genetic algorithm maintains a population of solutions which is iteratively recombined and mutated to evolve successive generations of candidate solutions.

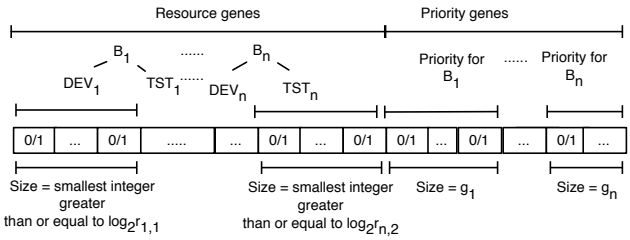


Figure 2. The bug-fixing chromosome structure.

A. Encoding and decoding of chromosome

Before encoding the scheduling problem as a GA chromosome, the following assumptions are made:

- Only one development resource and one testing resource can be allocated to each bug.
- One developer can only fix one bug at a time. Similarly one tester can verify one bug-fix at a time.

We use a binary representation of integers to encode the bug-fixing problem as a chromosome, an approach similar to the one in [8]. We establish a set of resource genes and a set of priority genes. For each bug B_i in bug repository $BR = \{B_1, B_2, \ldots, B_n\}$, the fixing of bug B_i includes two activities, development (DEV) and testing (TST). For each of these two activities, there are number of r_i capable (or additional candidate capable) resources that can be allocated to it. We encode these capable resources as a set of binary genes, where the size of the set is smallest integer greater than or equal to $log_2 r_{i,j}$ where i represents bug i and j represents activity type (DEV or TST). The binary values of these genes are used to represent the decimal number that identifies a scheduled resource as shown on the left part of Figure 2.

When two or more activities described by resource genes contend for the same resource, which activity can first acquire resources should be determined. Thus a group of genes named as priority genes are set, describing the activity priority (shown on the right part of Figure 2, where g is the priority gene size for each bug). The activity with higher priority is assigned to the resource first while if two activities have the same priority, the one placed to the left in the chromosome is assigned to the resource first.

Decoding is the reverse process of encoding. First, resource genes of each activity are decoded to a real number,

giving us scheduled resource for the activity. Second, the priority genes for each activity are decoded to a real number, giving us the priority of each activity. Third, the start time and end time for each activity is calculated. This calculation satisfies the following constraints:

1) If two activities require the same resource, the one with higher priority will be scheduled first.
2) The availability constraints of the human resources should be satisfied.

B. Multi-objective fitness evaluation of candidate solutions

Each scheduling result decoded by a chromosome is evaluated by means of a fitness function. The fitness function is designed to keep in view the scheduling objectives. Two generic and two specific objectives are taken in to consideration for scheduling. The generic objectives are:

- Objective 1: Bugs with higher priority and severity bring higher value on fixing.
- Objective 2: From a scheduling perspective, the maximum total value of fixing all the bugs should be obtained.

Besides these generic objectives, two specific objectives are used, each bringing a different value return for getting a bug fixed. One specific objective is the strict deadline objective:

- Objective 3: The deadline for each bug is strict. If a bug cannot be fixed before a deadline, the value for fixing this bug is minimum, i.e., 0. If it can be fixed before deadline, the value for fixing it is computed by its priority, severity and preference weight.

By using objectives 1, 2 and 3, the value of a bug B is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * HasFinished(B)$$

where α and β are the preference weights for priority and severity respectively; $HasFinished(B)$ is an operator:

$$HasFinished(B) = \begin{cases} 1 & B\,is\,fixed\,before\,deadline \\ 0 & B\,cannot\,be\,fixed\,before\,deadline \end{cases}$$

The other specific objective is the relaxed deadline objective:

- Objective 4: If bug-fixing is finished before the deadline, there is an incurred benefit. If bug-fixing is finished later than deadline, a penalty is applied.

By using objectives 1, 2 and 4, the value of a bug B is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * ScheduleValue(B)$$

where α and β are the preference weights of priority and severity respectively, while $ScheduleValue(B)$ is computed as follows:

$$ScheduleValue(B) = \begin{cases} (B.Deadline - B.FixedTime) * B.CSB \\ \quad for\,Deadline \geq FixedTime \\ (B.Deadline - B.FixedTime) * B.CSP \\ \quad for\,Deadline < FixedTime \end{cases}$$

where CSB and CSP are configurable parameters, set by the user. The strength of these coefficients indicate the impact of benefit or otherwise on the bug-fixing process so e.g. if the impact of missing a deadline is more, the corresponding coefficient is set to a higher value.

No matter whether the deadline of a bug is strict or not, the value function for the bug-fixing process is:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

This value function is used as a fitness function during the GA evolution process.

V. Industrial case study

Our proposed methodology for scheduling resources for bug-fixing activities is evaluated using real-world data from a large Enterprise Resource Planning (ERP) software developed by a global provider of geo-technology and information technology services. The company consists of over 600 skilled professionals and have successfully been certified as CMMi level 3 compliant. The ERP project has completed several releases while the data used in this paper comes from a batch of bugs reported by the testing team for an upcoming release. This upcoming release incorporates customized functionality for one of their telecom clients. The project team working on the upcoming release have to schedule appropriate resources to cut-down the back-log of reported bugs from the testing team. The project leader plans for fixing every bug by a set deadline (keeping in view the release date for the customer) and estimates the required effort using expert judgement. The project leader is responsible for triaging the bugs to resources having the required skills (along with required skill levels) and available times. The skill set and associated levels for every resource in the project is maintained by the human resource department and the project leader also has own qualitative assessments regarding the skill levels of resources under him. The empty time slots for every resource is available through a centralized calendar application.

Therefore, having bugs with different priority, severity, time constraints, resource constraints and having resources with varying skill sets with associated skill levels and available time slots, an automated mechanism to triage bugs with maximum possible value for the organization is required.

A. Description of bugs and human resources

We evaluated our approach on a set of 25 bugs logged in the bug repository during system testing. The ID, description, severity, priority, assigned time, deadline and estimated effort are shown in Table I. The bug descriptions have been modified to protect privacy. As discussed in Section III, there are two resource consuming activities taking place during the bug-fixing process: development (DEV) and testing (TST); each of these activities require relevant skill-set. Although there can be different ways of classifying skills required for both development and testing, we use more general skill

Table I
BUG DESCRIPTIONS.

Bug ID	Bug Description	Severity (H:High, M: Medium, L:Low)	Priority (H:High, M: Medium, L:Low)	Assigned time	Deadline	Estimated effort
1	Reservations removed.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
2	Built-in redundancy is lost.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
3	Replication too slow.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
4	Scheduled periodic account management job not working.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
5	The scheduled job cannot perform evaluation.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
6	Modifying existing schedule not allowed.	H	H	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
7	History of customer profile not loaded.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
8	Too low processing performance.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
9	Req002 not fulfilled.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
10	Volume input/output not working.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
11	CustomerHandler crashed.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
12	Fallback fails in step 2 of use case 1.	H	M	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
13	User data not updated in customerHandler memory.	H	M	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
14	Failed to generate customer request.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
15	Configuration file corrupted.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
16	The configuration log is missing latest settings.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
17	Too low performance for handling batch requests.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
18	Page not displayed on server authentication.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
19	Notification email not send.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
20	Database replication error.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
21	Incorrect error code.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
22	Incorrect salary shown.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
23	Report taking too long to generate.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
24	Message update required.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
25	Usage profile not updated.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours

Table II
SKILL REQUIREMENTS OF EACH BUG.

	Development skills						Testing skills						
Bug ID	Analytical	Programming language	Debugging	Refactoring	Use of IDE	Configuration management	Use of libraries and frameworks	Test planning (TP)	Test design (TD)	Test execution (TE)	Test review (TR)	use of bug tracking tool	Domain knowledge (DK)
1	H	M	H	M	M	M	M	H	H	H	M	M	H
2	H	M	H	M	M	M	M	H	H	H	M	M	H
3	H	M	H	M	M	M	M	H	H	H	M	M	H
4	H	H	H	H	M	M	M	H	H	H	M	M	H
5	H	H	H	H	M	M	M	H	H	H	M	M	H
6	H	H	H	H	M	M	M	H	H	H	M	M	H
7	H	M	H	M	M	M	M	H	H	H	M	M	H
8	H	M	H	M	M	M	M	H	H	H	M	M	H
9	H	H	H	H	M	M	M	H	H	H	M	M	H
10	H	M	H	M	M	M	M	H	H	H	M	M	H
11	H	M	H	M	M	M	M	H	H	H	M	M	H
12	H	H	H	H	M	M	M	H	H	H	M	M	H
13	H	H	H	H	M	M	M	H	H	H	M	M	H
14	H	M	H	M	M	M	M	H	H	H	M	M	H
15	H	M	H	M	M	M	M	H	H	H	M	M	H
16	H	M	H	M	M	M	M	H	H	H	M	M	H
17	H	M	H	M	M	M	M	H	H	H	M	M	H
18	M	M	M	L	L	L	L	M	M	M	L	L	M
19	M	M	M	L	L	L	L	M	M	M	L	L	M
20	H	M	H	M	M	M	M	H	H	H	M	M	H
21	M	M	M	L	L	L	L	M	M	M	L	L	M
22	M	M	M	L	L	L	L	M	M	M	L	L	M
23	M	M	M	L	L	L	L	M	M	M	L	L	M
24	M	M	M	L	L	L	L	M	M	M	L	L	M
25	M	M	M	L	L	L	L	M	M	M	L	L	M

requirements that could easily be mapped to more specific skill-set at our subject company. The skill requirements for each bug are described in Table II where H: High, M: Medium and L: Low. Human resource attributes for available development and testing personnel (as discussed in Section III-B) are shown in Table III.

VI. THE SCHEDULING RESULTS

We applied the GA proposed in Section IV to schedule capable resources for bug-fixing activities, based on the bug model and human resource model data given in Section III. The GA used the following parameters: Population size: 100, total number of generations: 500, cross-over rate: 0.8, mutation rate: 0.01, selection method: ratio. These parameters were obtained after some experimentation, however, in future we need a more systematic mechanism of tuning them.

We assume that delaying the bug-fixing after the deadline has greater impact than fixing it before, therefore, CSB is set as 10 and CSP as 30 for every bug, i.e., one day delay in bug-fixing has three times effect on the value than completing the bug-fixing one day before. Based on the configuration of coefficients and weights in balancing objectives (Section IV-B) and resources (Section V-A), different scenarios suggest strategies for managing resources for the bug-fixing tasks. We then discuss the scheduling results out of these scenarios.

A. Scenario 1: Priority preference weight, α = 20; Severity preference weight, β = 5

With priority weight, α, set to 20 and severity preference weight, β, set to 5, we first use objectives 1, 2 and 3 from Section IV-B. That is, we use the strict deadline as an objective and find that, using data from Section V-A, only 11 out of 25 bugs can be scheduled for fixing. These bugs are listed in Table IV and the corresponding Gantt chart plan for fixing these bugs is shown in Figure 3. Gantt chart is an easy way to illustrate a project schedule and provides an intuitive interface for the project leader to monitor scheduling elements. As is clear that using a strict deadline objective, only a limited number of bugs can be fixed.

We now use the relaxed deadline objective to schedule more bugs by relaxing the deadline constraint. We assume that all the resources are available after 20-Apr-2010 and

Table III
HUMAN RESOURCE DESCRIPTIONS.

HR ID	(SKT, SKN, SKL)	EXPD (Years)	STMW
HR1	(Developer, Analytical, H)	6	([25-Mar-2010, 07-Apr-2010], 6)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		([12-Apr-2010, 15-Apr-2010], 6)
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR2	(Developer, Analytical, H)	6	([20-Mar-2010, 04-Apr-2010], 8)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		([12-Apr-2010, 14-Apr-2010], 8)
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR3	(Developer, Analytical, M)	2	([23-Mar-2010, 12-Apr-2010], 8)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		([15-Apr-2010, 22-Apr-2010], 8)
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR4	(Developer, Analytical, M)	2	([23-Mar-2010, 05-Apr-2010], 6)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		([12-Apr-2010, 17-Apr-2010], 6)
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR5	(Tester, TP, H)	4	([25-Mar-2010, 08-Apr-2010], 4)
	(Tester, TD, H)		
	(Tester, TE, H)		
	(Tester, TR, M)		([12-Apr-2010, 14-Apr-2010], 4)
	(Tester, Bug Tracking Tool, M)		
	(Tester, DK, M)		
HR6	(Tester, TP, M)	2	([25-Mar-2010, 06-Apr-2010], 3)
	(Tester, TD, M)		
	(Tester, TE, M)		
	(Tester, TR, L)		([09-Apr-2010, 16-Apr-2010], 3)
	(Tester, Bug Tracking Tool, L)		
	(Tester, DK, M)		

Table IV
BUGS THAT CAN BE FIXED UNDER A STRICT DEADLINE (α=20, β=5).

Bug ID	Value	DEV	TST
2	70	HR2: ([25-Mar-2010, 26-Mar-2010], 8) ([29-Mar-2010, 29-Mar-2010], 8)	HR5: ([30-Mar-2010, 31-Mar-2010], 4)
3	70	HR2: ([30-Mar-2010, 01-Apr-2010], 8)	HR5: ([06-Apr-2010, 07-Apr-2010], 4)
9	75	HR1: ([29-Mar-2010, 01-Apr-2010], 6)	HR5: ([02-Apr-2010, 02-Apr-2010], 4) ([05-Apr-2010, 05-Apr-2010], 4)
10	75	HR1: ([02-Apr-2010, 02-Apr-2010], 6) ([05-Apr-2010, 07-Apr-2010], 6)	HR5: ([08-Apr-2010, 08-Apr-2010], 4) ([12-Apr-2010, 12-Apr-2010], 4)
18	45	HR4: ([25-Mar-2010, 26-Mar-2010], 6)	HR5: ([29-Mar-2010, 29-Mar-2010], 4)
19	45	HR3: ([25-Mar-2010, 26-Mar-2010], 8)	HR6: ([29-Mar-2010, 30-Mar-2010], 3)
21	25	HR1: ([25-Mar-2010, 26-Mar-2010], 6)	HR6: ([31-Mar-2010, 01-Apr-2010], 3)
22	25	HR3: ([29-Mar-2010, 30-Mar-2010], 8)	HR5: ([01-Apr-2010, 01-Apr-2010], 4)
23	25	HR3: ([01-Apr-2010, 02-Apr-2010], 8)	HR6: ([05-Apr-2010, 06-Apr-2010], 3)
24	25	HR3: ([05-Apr-2010, 06-Apr-2010], 8)	HR5: ([13-Apr-2010, 13-Apr-2010], 4)
25	25	HR3: ([07-Apr-2010, 08-Apr-2010], 8)	HR6: ([09-Apr-2010, 09-Apr-2010], 3) ([12-Apr-2010, 12-Apr-2010], 3)

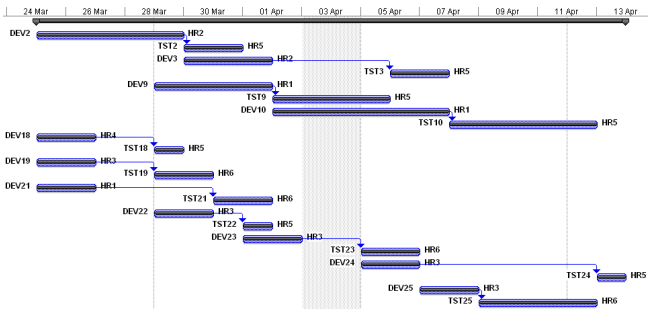


Figure 3. Strict deadline bug-fixing plan.

Table V
BUGS THAT CAN BE FIXED UNDER A RELAXED DEADLINE (α=20, β=5).

Bug ID	Value	Precedent days compared to the deadline	Bug ID	Value	Precedent days compared to the deadline
1	-18900.0	-9	14	-37500.0	-25
2	-23100.0	-11	15	-24000.0	-16
3	4900.0	7	16	-33000.0	-22
4	-4500.0	-2	17	-42000.0	-28
5	-40500.0	-18	18	4950.0	11
6	-33750.0	-15	19	4500.0	10
7	-21000.0	-10	20	-25500.0	-17
8	6300.0	9	21	2000.0	8
9	3750.0	5	22	1500.0	6
10	-16800.0	-8	23	1750.0	7
11	-31500.0	-15	24	1250.0	5
12	-52800.0	-32	25	2250.0	9
13	-51150.0	-31			

each workday comprises of 8 working hours. Using objectives 1, 2 and 4 from Section IV-B, the simulation results appear in Table V. The data in Table V indicates that relaxing the deadline enables all the bugs to be scheduled for fixing but many of them are delayed as indicated by negative integers in the third and sixth columns of Table V. The corresponding Gantt chart plan is shown in Figure 4 that can help show the project leader that negotiating a relaxation in deadline would help fix all the bugs.

B. Scenario 2: Priority preference weight, α = 5; Severity preference weight, β = 20

In this scenario, we change the priority and severity preference weights as α = 5 and β = 20 respectively; that is to say that we now consider severity as more important than the priority of a bug. Using the strict deadline as an objective, the simulation results indicate that there are still 11 out of 25 bugs that can be scheduled before deadline. Although the total number of bugs that could be fixed before deadline remains the same for both the scenarios, a comparison of two schedules indicate that the two scheduling plans differ at bug IDs 3, 5, 10 and 16. This is shown in Table VI. The data in Table VI show that increasing the severity preference weight β (scenario 2) has resulted in scheduling bug ID 5 with highest priority but at the cost of not fixing bug IDs 3 and 10 from scenario 1. Since bug IDs 3 and 10 cannot be fixed, the available resources are enough to fix bug ID 16.

Scenarios 1 and 2 indicate that by plugging different combinations of priority and severity preference weights, the project leader can balance the importance of fixing certain bugs at the cost of others (provided that the deadline is strict). This, in our view, suggest valuable strategies for resource scheduling.

C. Scenario 3: Priority preference weight, α = 20; Severity preference weight, β = 5; Simulating virtual resources

In the previous two scenarios we see that, using a strict deadline, the resources are not enough to fix all the bugs on or before the deadline. We also saw that one way to

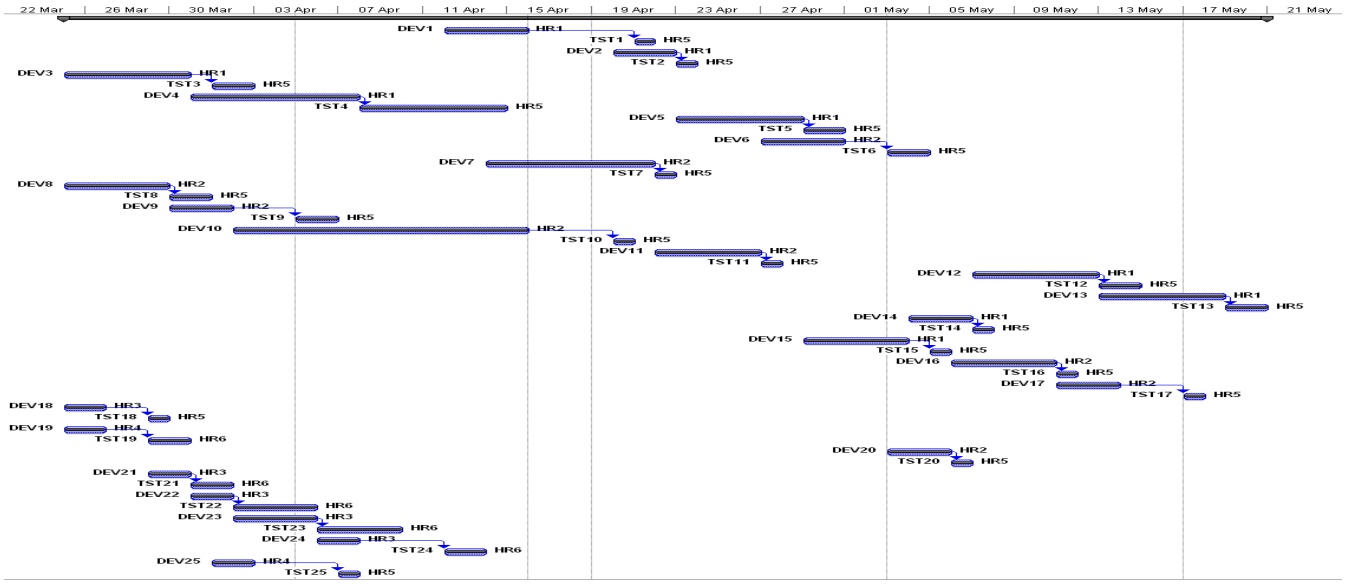


Figure 4. Relax deadline bug-fixing plan.

Table VI
COMPARISON OF BUG-FIXING SCHEDULES UNDER DIFFERENT PRIORITY AND SEVERITY PREFERENCE WEIGHTS.

Bug ID	Priority	Severity	Value preference weight	
			$\alpha = 20; \beta = 5$	$\alpha = 5; \beta = 20$
2	3	2	70	55
3	3	2	70	0
5	3	3	0	75
9	3	3	75	75
10	3	2	75	0
16	2	2	0	50
18	2	1	45	30
19	2	1	45	30
21	1	1	25	25
22	1	1	25	25
23	1	1	25	25
24	1	1	25	25
25	1	1	25	25

provide more candidate resources is to relax the deadline. The other way to achieve the deadline is, of course, addition of more resources. Therefore, we add virtual resources for fixing bugs in this scenario. Initially we have 6 resources and are able to schedule 11 out of 25 bugs for fixing. Increasing the resources to two more by adding one development resource and one testing resource, with high skill levels in all skills, enables scheduling over 15 bugs for fixing. Similarly increasing the resources to 10 by adding two highly-skilled development resources and two highly-skilled testing resources allow scheduling more than 20 bugs before deadline (Figure 5).

This scenario gives another option to the project leader for viewing the scheduling outcome if more resources were available than initially assigned. Therefore, the simulation

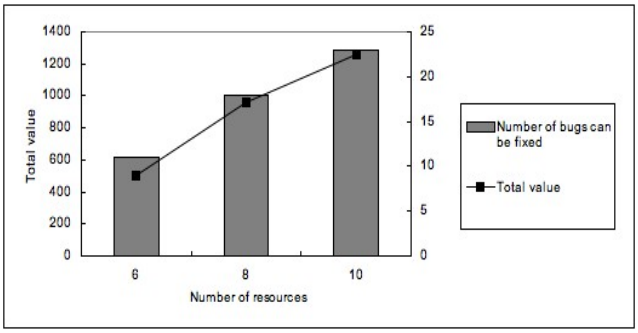


Figure 5. Effects of increasing number of resources.

of virtual resources can provide schedules under varying circumstances, keeping in view the available resource pool.

VII. COMPARISON WITH HILL-CLIMBING SEARCH

Hill-climbing (HC) is a basic local search algorithm and, likewise GA, is used to compute the value obtained in scheduling resources for bug-fixing tasks. We have used the three scenarios discussed in Section VI to compute the total bug value by using HC and have compared it with GA. The results are shown in Figure 6. The figure shows that if the number of bugs is small (i.e. 1 to 3), GA and HC obtain the same optimal value. But when the scale of problem increases with an increase in number of bugs, GA gives better results than HC. In order to test whether any significant differences exist between the bug values from two algorithms, we used Wilcoxon rank sum test. The p-value of 0.004 confirmed that the bug values from HC and GA do not have equal

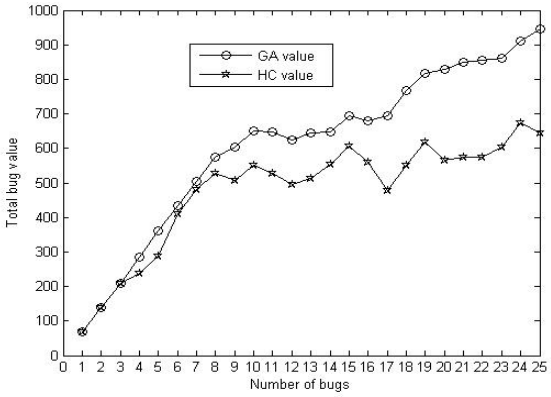


Figure 6. A comparison of total bug value obtained by using GA and HC.

medians at 0.05 significance level. Thus bug values from GA are significantly different and better than those of HC.

VIII. Discussion

Considering that we have a need to support efficient and effective bug-assignment policies, this paper has provided early results as to how a GA can help strike a balance between competing constraints to achieve near-optimal value for the organization. Due to the dynamic nature of the bug-fixing schedules, different scenarios are possible and these changing scenarios have to be modeled effectively for near-optimal solutions. The multi-objective fitness function proposed in this work attempts to model the uncertainty in the scheduling problem. The scenarios discussed here provide a way to schedule resources under different circumstances, e.g., having a strict/flexible deadline, having assigned different weights to severity and priority and last but not least, the ability to foresee the resource requirements by adding virtual resources to meet the deadline. GA is able to effectively suggest different strategies to tackle the bug-fixing process and is found to be more effective than hill-climbing. Consequently the project leaders can use these results to support their resource scheduling decisions.

We are also aware of certain limitations of our study. First we have some assumptions that might get violated, e.g., it is common that the bug-fix is verified not to be correct by the testing team and a second round of bug-fixing activities is undertaken. If this is the case, then the different elements of the bug model would require new data for the second round of activities. We, however, limit ourselves to only one round of bug-fixing activities in this paper. Second, there are some rules that are followed for the relaxed deadline objective. While these rules would differ with respect to the expectations of the project leader of her team members, we followed some intuitive ones. Any change in this rules is, however, possible. Third, there is a possibility that a resource works concurrently on more than one assignment. However we only consider the empty time slots that a particular resource has for dealing with one bug at a time. If such a division of workload is not possible then it is expected that the human resource model needs to incorporate this change. Fourth, a company might face the difficulty to quantify the skills and the associated levels. As our subject company is on the path of CMMi Level 4, such a quantification is seen as a continuing improvement opportunity for the workforce. The human resource model presented here uses a simple classification of skills which can be changed to suit specific needs. There is a possibility that the human resource model in this paper has ignored relevant human performance factors. An important point to make here is that the company using such an approach needs to continuously update the skill database of its resources since it is common for the resources to educate themselves and learn as part of the project experience.

IX. Validity threats

There can be three types of validity threats to the kind of study we have conducted. *Construct validity* threats refer to the extent the experiment setting actually reflects the construct under study [24]. These threats might arise due to the assumptions we made in the study and the way we modeled the problem. However, a search-based technique such as GA is independent of the way the problem is modeled; it is the fitness function that contains the crucial information and needs to be adapted for a more complex model. The assumptions in this paper made sense for the type of case study discussed, however, as mentioned in Section VIII, the bug and human resource model might change if a different process of bug-fixing is followed. *Internal validity* threats refer to any sources of bias that might have affected the results. Since GA is a stochastic algorithm, different runs produces different solutions. The different GA parameters were obtained after careful experimentation and taking into account that further changing the parameter values do not have significant impact on the results. Moreover, the GA was run multiple times (30) to overcome randomness inherent in the GA. *External validity* threats are concerned with generalization. The results obtained in this paper as such should be applicable to the situations where our assumptions are held. Otherwise, the bug and the human resource model needs to be adapted accordingly.

X. Conclusions and future work

We have presented a search-based resource allocation method for bug-fixing tasks. We proposed models for the bug-fixing process, the human resources and the capability matching method between bug-fixing activities and human resources. On the basis of these models and our proposed method, the resources were allocated for bug-fixing activities using a GA. Depending on differing objectives, three scenarios were discussed using an industrial data set and the results

showed that GA was able to give schedules having balanced different objectives and entailing maximum value for the organization. Comparison with hill-climbing showed that GA gave statistically better results in terms of maximizing the value objective.

Based on this paper, some interesting future work can be undertaken:

- Combining the bug-fixing process with other resource consuming activities that might happen concurrently, e.g., testing of newly implemented requirements might take place in parallel with bug-fixing activities, probably needing similar resources.
- Increasing the generalizability of the proposed method by considering scheduling a larger set of bugs.
- Supplementing the scheduling with cost issues, i.e., the cost incurred in investing resources to perform different activities might impact the value objective.
- Analyzing the sensitivity of parameters in the GA and the fitness function, such as population size of the GA, priority preference weight and severity preference weight of the fitness function.

XI. Acknowledgements

Junchao Xiao is supported by the National Natural Science Foundation of China under grant Nos. 90718042, 60903051, the 863 Program of China under grant No. 2007AA010303, as well as the 973 program under grant No. 2007CB310802. Wasif Afzal is supported by the Swedish research school in verification and validation (www.swell.se).

References

[1] H. Pham, *Software reliability*. Singapore: Springer-Verlag, 2000.

[2] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Procs. of the Int. Conf. on Software Engineering*, 2006.

[3] P. Laplante and N. Ahmad, "Pavlov's bugs: Matching repair policies with rewards," *IT Professional*, vol. 11, no. 4, 2009.

[4] M. B. Wall, "A GA for resource-constrained scheduling," Ph.D. dissertation, Dept. of Mechanical Eng., Massachusetts Institute of Technology, USA, 1996.

[5] L. Ozdamar and G. Ulusoy, "A survey on the resource-constrained project scheduling problem," *IIE Transactions*, vol. 27, pp. 574–586, 1995.

[6] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Dept. of CS, King's College London, Tech. Report TR-09-03, 2009.

[7] G. Antoniol, M. Di Penta, and M. Harman, "Search-based techniques applied to optimization of project planning for a massive maintenance project," in *Procs. of the 21st IEEE Int. Conf. on SW Maintenance*. IEEE, 2005.

[8] J. Xiao, Q. Wang, M. Li, Q. Yang, L. Xie, and D. Liu, "Value-based multiple software projects scheduling with GA," in *Procs. of the 2009 Int. Conf. on SW Process*, 2009.

[9] E. Alba and J. Chicano, "SW project management with GAs," *Information Sciences*, vol. 177, no. 11, pp. 2380–2401, 2007.

[10] C. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for SW proj. scheduling with genetic algorithms," *IST*, vol. 50, no. 11, pp. 1142–1154, 2008.

[11] C. Kirsopp, M. J. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction," in *Procs. of the Genetic and Evolutionary Computation Conf.* Morgan Kaufmann Publishers Inc., 2002.

[12] W. Afzal, R. Torkar, R. Feldt, and T. Gorschek, "Genetic programming for cross-release fault count predictions in large and complex software projects," in *Evolutionary Computation and Optimization Algorithms in Software Engineering*, M. Chis, Ed. IGI Global, Hershey, USA, 2010.

[13] E. Hart, P. Ross, and D. Corne, "Evolutionary scheduling: A review," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 191–220, 2005.

[14] A. Mockus, D. M. Weiss, and P. Zhang, "Understanding and predicting effort in software projects," in *Procs. of the 25th Int. Conf. on SW Eng.* IEEE Computer Society, 2003.

[15] D. Cubranic and G. Murphy, "Automatic bug triage using text categorization," in *Procs. of the 16th Int. Conf. on SW Eng. and Knowledge Eng.*, 2004.

[16] H. Zeng and D. Rine, "Estimation of software defects fix effort using neural networks," in *Procs. of the 28th Annual Int. Computer SW and Applications Conf. - Workshops and Fast Abstracts*. IEEE Computer Society, 2004.

[17] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Procs. of the 2006 ACM symp. on Applied computing*. ACM, 2006.

[18] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "SW defect association mining and defect correction effort prediction," *IEEE Trans. on SW Eng.*, vol. 32, no. 2, 2006.

[19] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Procs. of the 4th Int. WS on Mining SW Repositories*. IEEE, 2007.

[20] "Bugzilla – A bug tracking tool." http://www.bugzilla.org/, Last checked 21 Mar 2010.

[21] *IEEE std. classification for SW anomalies, IEEE Std. 1044-1993*, IEEE, Inc., USA, 1993.

[22] J. Xiao, Q. Wang, M. Li, Y. Yang, F. Zhang, and L. Xie, "A constraint-driven human resource scheduling method in software development and maintenance process," in *24th IEEE Int. Conf. on SW Maintenance*. IEEE, 2008.

[23] J. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992.

[24] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: An introduction*. USA: Kluwer Academic Publishers, 2000.

The Human Competitiveness of Search Based Software Engineering

Jerffeson Teixeira de Souza, Camila Loiola Maia, Fabrício Gomes de Freitas and Daniel Pinto Coutinho
Optimization in Software Engineering Group (GOES.UECE)
State University of Ceará (UECE)
Fortaleza, 60740-903, Ceará, Brazil
jeff@larces.uece.br, camila.maia@gmail.com, fabriciogf@uece.br, daniel.pintocoutinho@gmail.com

Abstract - **This paper reports a comprehensive experimental study regarding the human competitiveness of search based software engineering (SBSE). The experiments were performed over four well-known SBSE problem formulations: next release problem, multi-objective next release problem, workgroup formation problem and the multi-objective test case selection problem. For each of these problems, two instances, with increasing sizes, were synthetically generated and solved by both metaheuristics and human subjects. A total of 63 professional software engineers participated in the experiment by solving some or all problem instances, producing together 128 responses. The comparison analysis strongly suggests that the results generated by search based software engineering can be said to be human competitive.**

Keywords - human competitiveness; search based software engineering; SBSE; human subjects.

I. INTRODUCTION

Search Based Software Engineering (SBSE) has emerged as a promising research field. This recent discipline involves the modeling and resolution of complex software engineering problems as optimization problems, especially with the use of metaheuristics [1]. Its wide applicability to various problems in nearly all software development life cycle phases, has drawn interest from the software engineering research community and beyond.

The question regarding the human competitiveness of search based software engineering results has already been raised [1]. However, even with the high impact this issue may have in this novel research field, no comprehensive work has been published to date. To explain this research void, one may argue that the human competitiveness of search based software engineering is not in doubt by the SBSE community. Even if that is the case, strong research results regarding this issue would likely, in the least, contribute to the increasing acceptance of SBSE outside its research community.

Therefore, the research summarized in this paper aims at answering mainly the following general question:

> RQ1. – *SBSE human competitiveness*: Can the results generated by Search Based Software Engineering be said to be human competitive?

In order to process this human competitiveness analysis in a fair manner, it is important to appropriately select the metaheuristics which will represent SBSE. With that in mind, additional experiments reported in this paper were designed to examine the ability of different mono- and multi-objective metaheuristics to solve some software engineering problems modeled as optimization problems. Therefore, the following secondary research question will also be tackled:

> RQ2. – *SBSE algorithms comparison*: how do different metaheuristics compare in solving a variety of search based software engineering problems?

It is worth clarifying at this point that such SBSE algorithms comparison is not meant to be exhaustive in any sense. In fact, the experiments were designed solely to allow an informative selection of algorithms to be used in the human competitiveness study.

The remaining of the paper is organized as follows: Section II discusses the human competitiveness evaluation process. Section III describes prior work on human competitiveness of search based software engineering. Section IV discusses in detail the design of the experiments aimed at answering the two research questions raised, including an exposition on ethical issues. Next, Section V presents results from the experiments and describes the analyses of these results. Finally, Section VI concludes the work and points out some future directions to this research.

II. EVALUATING HUMAN COMPETITIVENESS

As recognized by the Genetic and Evolutionary Computation Conference (GECCO), one of the eight criteria [2] which can be used to characterize the human competitiveness of automatically created results is:

> *The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).*

The research reported in this paper addresses human competitiveness of search based software engineering results precisely under this criterion. Therefore, controlled experiments, as described later, designed to compare the automatically generated results with the results obtained from human subjects were performed and analyzed.

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.25

III. Related Works

As previously mentioned, the literature lacks consistent and comprehensive research regarding human competitiveness of search based software engineering. Though, this issue has been sporadically addressed by some papers, discussed next.

Reference [3] studies both the selection and prioritization versions of the Next Release Problem. Through experimentation, the authors compared the performance of a panel of human experts to greedy and simulated annealing algorithms over a large-scale real-world instance. Results showed that the automated search outperformed the human experts.

In [4], authors introduced a new multi-objective formulation to the Software Release Planning Problem, extending a previous formulation by taking into account other aspects considered to be important. In this work, the results generated by the metaheuristic NSGA-II were contrasted with solutions produced by five experienced professional software engineers. A graphical analysis indicated that the results generated by all human subjects were outperformed by NSGA-II.

Finally, reference [5] proposed an optimization-based approach to support staffing software projects by modeling this problem as a constrained satisfaction problem. In order to evaluate the relevance of using the proposed constrained satisfaction approach, sixteen graduate students were asked to solve a staffing problem instance by using a technique of their choice. Their solutions were then compared with the solutions produced by a decision support tool which implemented the proposed approach. The results showed that the support tool could be useful to staff software projects, since several participants were unable to find the best solution generated by such tool.

IV. Experimental Design

This section describes all aspects related to the design of the experiments. It first introduces which problems were tackled, followed by a description of the employed data and algorithms. Finally, it discusses the human subjects and how ethical concerns were overcome.

A. The Problems

For the experiments, four search based software engineering problems were selected: Next Release Problem [6], Multi-Objective Next Release Problem [7], Workgroup Formation Problem [8] and the Multi-Objective Test Case Selection [9].

The general motivation for the election of these peculiar problems was twofold. First, they can be considered "classical" formulations in search based software engineering, especially because of the considerable amount of published research addressing or specializing them [3,4,5,10-43]. Finally, they cover together a range of three different general phases in the software development life cycle, i.e., requirements engineering, software planning and testing, besides dealing with mono and multi-objective contexts.

Next, each one of these four problems is concisely described.

The Next Release Problem (NRP), in its original formulation as a constrained mono-objective optimization problem [6], involves determining a set of customers which will have their selected requirements delivered in the next software release. This selection prioritizes customers with higher importance to the company ($maximize\ \sum_{i \in S} w_i$), and must respect a pre-determined budget ($subject\ to\ cost(\bigcup_{i \in S} R_i^*) \leq B$). The NRP has been shown to be NP-hard even when costumer requirements are independent.

The Multi-Objective Next Release Problem (MONRP) [7] can be seen as a generalization of the NRP. In the MONRP, the cost of implementing the selected requirements is taken as an independent objective to be optimized ($minimize\ \sum_{i \in S} cost_i$), not as a constraint, along with a score representing the importance of a given requirement ($maximize\ \sum_{i \in S} score_i$).

The Workgroup Formation Problem (WORK) deals with the allocation of human resources to project tasks [8]. The search based formulation displays a single objective function to be minimized, which composes both salary costs, skill and preference factors ($minimize\ \ SalCost - \lambda SkillPref - \eta PrefFactor$). For salary, solutions which minimize its cost ($\sum_{p=1}^{P} \sum_{a=1}^{N} Sal_p\, A_{ap}\, Dur_a$) are sought. The skill factor ($\sum_{p=1}^{P} \sum_{s=1}^{S} \sum_{a=1}^{N} R_{ps} A_{ap} SI_s$) models the predilection for highly qualified people. Yet for the preference factors, three aspects are considered: the preference one has to work on a certain task ($\sum_{a=1}^{N} \sum_{p=1}^{P} P_{pa} A_{ap}$), the manager preference ($\sum_{a=1}^{N} \sum_{p=1}^{P} P_{mpa} A_{ap}$) and the interpersonal preference ($\sum_{a=1}^{N} \sum_{p1=1}^{P} \sum_{p2=1}^{P} P_{p1p2} A_{ap1} A_{ap2} X_{p1p2}$).

Finally, the Multi-Objective Test Case Selection Problem (TEST) [9] extends previously published mono-objective formulations. The paper discusses two variations, one which considers two objectives (code coverage and execution time), used here, and the other covering three objectives (code coverage, execution time and fault detection).

B. The Data

For each problem (NRP, MONRP, WORK and TEST), two instances, A and B, with increasing sizes, were synthetically generated. In each case, the instance indicated with A is the smaller one, as shown in Tables I, II, III and IV.

TABLE I. Instances for Problem NRP

Instance Name	Instance Features	
	# Customers	*# Tasks*
NRPA	10	20
NRPB	20	40

TABLE II. Instances for Problem MONRP

Instance Name	Instance Features	
	# Customers	*# Requirements*
MONRPA	10	20
MONRPB	20	40

TABLE III. INSTANCES FOR PROBLEM WORK

Instance Name	Instance Features		
	# Persons	*# Skills*	*# Activities*
WORKA	10	5	5
WORKB	20	10	10

TABLE IV. INSTANCES FOR PROBLEM TEST

Instance Name	Instance Features	
	# Test Cases	*# Code Blocks*
TESTA	20	40
TESTB	40	80

C. The Algorithms

Each problem instance was solved by the metaheuristics discussed below. Due to space constraints, only the main features of each algorithm are pointed out.

For Mono-Objective Problems

I. Genetic Algorithm (GA) [44]: it is a widely applied evolutionary algorithm, inspired by Darwin´s theory of natural selection, which simulates biological processes such as inheritance, mutation, crossover, and selection.

II. Simulated Annealing (SA): it is a procedure for solving arbitrary optimization problems based on an analogy with the annealing process in solids [45].

For Multi-Objective Problems

I. NSGA-II (Non-Dominated Sorting Genetic Algorithm II) [46]: this multi-objective variation of the genetic algorithm employs two sorting procedures, the fast non-dominated and the crowding distance, in order to, respectively, seek for solutions close to the optimal Pareto Front and maintain the diversity of these solutions.

II. MoCell (Cellular Genetic Algorithm for Multi-objective Optimization): it is a cellular genetic algorithm, where the cooperation amongst individuals may only occur between neighbors [47]. It employs an external and limited size archive, where non-dominated solutions are stored and eventually used to replace individuals in the population through a feedback procedure. Similarly to NSGA-II, MoCell applies the crowding distance to seek for diverse solutions.

For Mono and Multi-Objective Problems

I. Random Search (Rand): it was applied to solve all instances as a form of "sanity check", since all metaheuristics should effortlessly outperform this unintelligent procedure.

As for the configuration of each metaheuristic, the following values were used: the maximum number of evaluations for NRPA and WORKA was set to 1,000, while for all other instances this number was 10,000. For GA, the crossover rate was 0.9 and mutation rate, 0.05. The population size for NSGA-II and MOCell was set to 10 for NRPA and WORKA, and was set to 100 on the other problem instances. The archive size in MOCell was defined as 10 for NRPA and WORKA, and as 100 for the other cases.

These particular configurations for each metaheuristic were empirically obtained through a preliminary experimentation process.

D. The Human Subjects

A total of 63 professional software engineers solved some or all of the instances described earlier. Table V shows the number of respondents per problem instance.

TABLE V. NUMBER OF HUMAN RESPONDENTS PER INSTANCE

NRP		MONRP		WORK		TEST	
A	**B**	**A**	**B**	**A**	**B**	**A**	**B**
21	13	21	13	18	10	22	10

Besides solving the problem instance, each respondent was asked to answer the following questions related to each problem instance:

1. *How hard was it to solve this problem instance?*
2. *How hard would it be for you to solve an instance twice this size?*
3. *What do you think the quality of a solution generated by you over an instance twice this size would be?*

In addition to these specific questions regarding each problem instance, general questions on the respondent theoretical and practical experience over software engineering were answered by each human participant.

Once again, due to space constraints, this paper does not present nor discusses all responses to the questions above. Such a detailed description and analysis will be presented in a future publication.

E. Comparison Metrics

When comparing solutions, several metrics could be employed. Below, the metrics used in the experiments are discussed.

For Mono-Objective Problems

A. Quality: it relates to the quality of each generated solution. For maximization problems (NRP, MONRP and TEST), higher values indicate higher quality. Conversely, for WORK, lower values are desired.

For Multi-Objective Problems

A. Hypervolume: for multi-objective problems where all objectives are to be minimized, this metric calculates the volume covered by members of a non-dominated solution set. A higher hypervolume value is preferred, as it indicates solutions closer to the optimal Pareto Front.

B. Spread: this metric measures the diversity of a non-dominated solution set. Thus, a solution set with spread close to zero is desired, as it indicates that the solutions are uniformly distributed.

C. Number of Solutions: it represents the number of different solutions present in the non-dominated solution set.

For Mono and Multi-Objective Problems

A. Execution Time: it measures the required execution time of each strategy.

F. Ethical Issues

Since the experimentation involves human participants, ethical concerns become an important factor. In that regard, all steps, as discussed next, have been taken to guarantee that fundamental ethical principles in empirical studies of software engineering [48] have been respected.

First, full informed and voluntary consent to participate in the research were obtained from all human subjects. Thus, all participants received detailed information regarding the purpose of the research, as well as its anticipated benefits and risks. Additionally, in order to guarantee that the consents were given under conditions without influence, even if indirect or unintentional, it was not allowed the participation of subjects over which any of the researchers of the present work held any current power, including the researchers' students and employees. Furthermore, by considering the insignificant risks to the human participants and the anticipated benefits of this research, its potential scientific value seems irrefutable, as well as its beneficence. Finally, in terms of confidentiality, both anonymity and confidentially of data were preserved. In order to assure the protection of anonymity, no data that could be used to identify any of the participants was collected.

Furthermore, both the ACM Code of Ethics and Professional Conduct [49], the IEEE Code of Ethics [50] and the IEEE-CS/ACM Software Engineering Code of Ethics and Professional Practice [51] were thoroughly consulted and respected.

V. Experimental Results and Analyses

This section presents all experimental results on the algorithms comparison and human competitiveness of SBSE. Initially, the results related to the comparison of algorithms for both mono- and multi-objective problems is described and analyzed. These results relate to the research question RQ2 discussed earlier. Next, the results obtained from the human subjects are presented and contrasted with the results from the metaheuristics, shedding light to our research question RQ1.

A. RQ2. – SBSE algorithms comparison

For the mono-objective problems NRP and WORK, Tables VI and VII summarize the results, including the averages and standard deviations, over 100 executions, for the Genetic Algorithm (GA), Simulated Annealing (SA) and Random Search (RAND), regarding the quality of the generated solutions and the execution times, respectively.

1 It is worth reminding that the Next Release Problem (NRP) is a maximization problem, while the Workgroup Formation Problem (WORK) attempts at minimizing the objective function.

As can be seen in Table VI, in terms of quality, the Genetic Algorithm outperformed the Simulated Annealing metaheuristic in all cases[1]. Not surprisingly, the Random strategy had the worst performance of all. However, time wise, as shown in table VII, Simulated Annealing performed significantly better than the Genetic Algorithm.

TABLE VI. Quality of Results for NRP and WORK

Problem	GA	SA	RAND
NRPA	26.45±0.500	25.74±0.949	15.03±5.950
NRPB	95.41±0.190	90.47±7.023	45.74±11.819
WORKA	16,026.17±51.700	18,644.71±1,260.194	19,391.34±1,220.17
WORKB	24,831.23±388.107	35,174.19±2,464.733	36,892.64±2,428.269

TABLE VII. Time (in miliseconds) Results for NRP and WORK

Problem	GA	SA	RAND
NRPA	40.92±11.112	23.01±7.476	0.00±0.002
NRPB	504.72±95.665	292.62±55.548	0.06±0.016
WORKA	242.42±44.117	73.35±19.702	0.04±0.010
WORKB	4,797.89±645.360	2,211.28±234.256	1.75±0.158

Figure 1 below shows boxplots presenting average, maximum and minimum values and 25% - 75% quartile ranges of quality for all instances of NRP and WORK. This graphical representation clearly shows the similar behavior of GA and SA, in terms of average, for the NRP problem instances, even though SA had significantly worse minimum values in these instances. As for WORK, the SA performance deteriorated, relatively to GA, both in terms of average, as well as for maximum and minimum values, in case of instance B. This loss in performance for SA on problem WORK, as can be seen, approximates its results to the Random Search. This poor SA performance over WORK may be attributed to chance or to a specific parameterization.

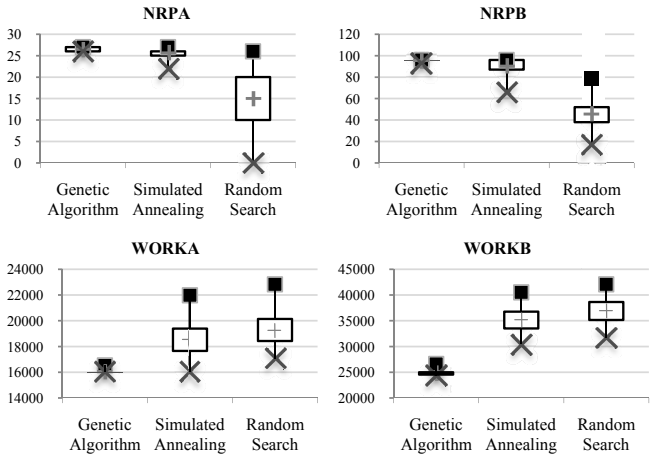


Figure 1. Boxplots showing average (+), maximum (■), minimum (×) and 25% - 75% quartile ranges of quality for mono-objective problems NRP and WORK, instances A and B, for GA, SA and Random Search.

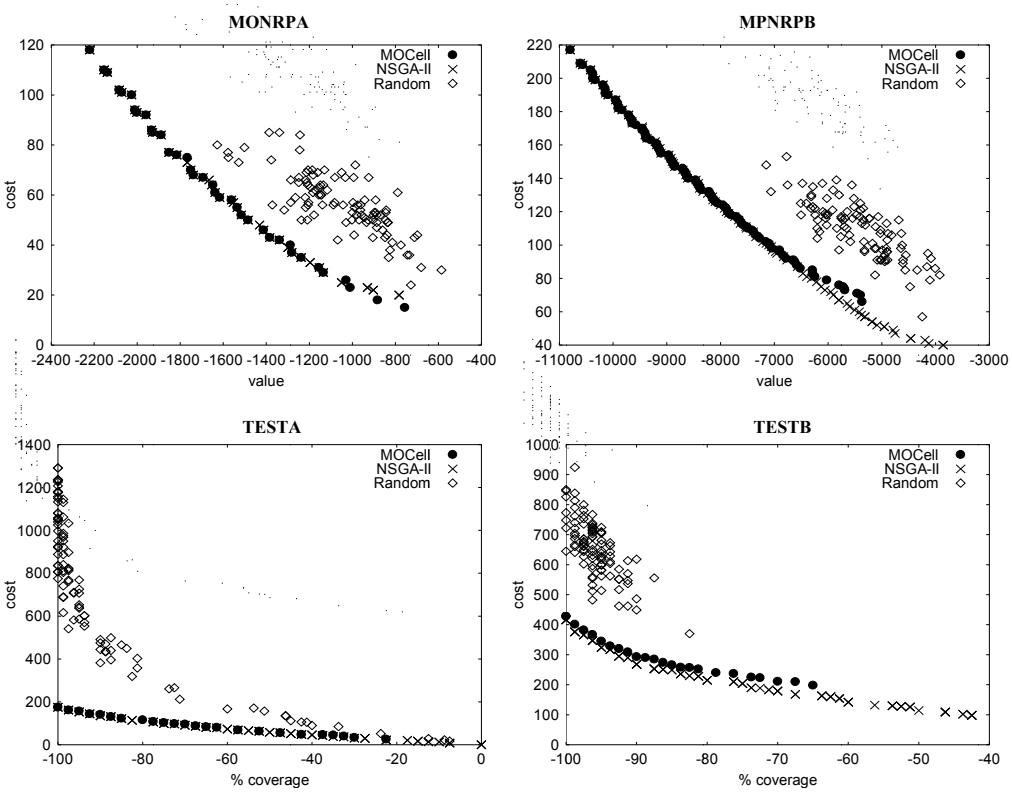


Figure 2. Example of the obtained solution sets for NSGA-II, MOCell and Random Search over problems MONRP and TEST, instances A and B.

For multi-objective problems, Tables VIII, IX, X and XI, present, respectively, averages and standard deviations of hypervolume, spread, execution time and number of solutions, for metaheuristics NSGA-II, MOCell and the Random Search, over 100 executions.

TABLE VIII. HYPERVOLUME RESULTS FOR MONRP AND TEST

Problem	NSGA-II	MOCell	RAND
MONRPA	0.6519±0.009	0.6494±0.013	0.5479±0.0701
MONRPB	0.6488±0.015	0.6470±0.017	0.5462±0.0584
TESTA	0.5997±0.009	0.5867±0.019	0.5804±0.0648
TESTB	0.6608±0.020	0.6243±0.044	0.5673±0.1083

TABLE IX. SPREAD RESULTS FOR MONRP AND TEST

Problem	NSGA-II	MOCell	RAND
MONRPA	0.4216±0.094	0.3973±0.031	0.5492±0.1058
MONRPB	0.4935±0.098	0.3630±0.032	0.5504±0.1081
TESTA	0.4330±0.076	0.2659±0.038	0.5060±0.1029

TESTB	0.3503±0.178	0.2963±0.072	0.4712±0.1410

NSGA-II outperformed MOCell, in terms of hypervolume in all cases, as can be verified in Table VIII. However, on spread, MOCell had a regular and significantly better performance (Table IX). Therefore, all results described on these two tables are consistent to show both the general abilities of NSGA-II to generate solutions close to the optimal Pareto Front and of MOCell to distribute solutions uniformly through the non-dominated solution set. These results are compatible with the findings of previous works [47][52][53]. As expected, the Random Search had the worst performance both in terms of hypervolume as well as for spread.

TABLE X. TIME (IN MILISECONDS)RESULTS FOR MONRP AND TEST

Problem	NSGA-II	MOCell	RAND
MONRPA	1,420.48±168.858	993.09±117.227	25.30±10.132
MONRPB	1,756.71±138.505	1,529.32±141.778	30.49±7.204
TESTA	1,661.03±125.131	1,168.47±142.534	25.24±11.038
TESTB	1,693.37±138.895	1,370.96±127.953	32.89±9.335

In terms of execution time (Table X), MOCell performed more efficiently than NSGA-II. On average, MOCell executed 30.88% faster. The Random Search, as expected, was significantly faster than both metaheuristics.

TABLE XI. NUMBER OF SOLUTIONS RESULTS FOR MONRP AND TEST

Problem	NSGA-II	MOCell	RAND
MONRPA	31.97±5.712	25.01±5.266	12.45±1.572
MONRPB	60.56±4.835	48.04±4.857	20.46±2.932
TESTA	35.43±4.110	26.20±5.971	12.54±1.282
TESTB	41.86±9.670	19.93±8.514	11.58±2.184

As for the number of generated solutions in the non-dominated solution set, NSGA-II found, in average, roughly 12 solutions more than MOCell. This increased number of solutions is important because it gives a broader choice to a user, who can, this way, consider more tradeoffs. As for the Random Search, when counting only the non-dominated solutions, this method produced around half of the solutions created by MOCell.

When specifically analyzing the results obtained for the multi-objective next release problem (MONRP), all results regarding hypervolume, spread and number of generated solutions are consistent with the finding of Durillo et. al. [53]. These similar results help validating the behavior of NSGA-II and MOCell over this problem, especially since the data employed in both studies were different, which indicates that the obtained behavior may be generalized.

Figure 2 presents an example of the obtained solution sets generated from the executions of metaheuristics NSGA-II, MOCell and the Random Search, over instances MONRPA, MONRPB, TESTA and TESTB. First, it is clear that both metaheuristics outperformed the random procedure. In addition, the graphs show the slight differences in the Pareto Fronts generated by NSGA-II and MOCell.

B. RQ1. – SBSE human competitiveness

Regarding our main research question, the results presented and discussed next evaluate the human competitiveness of search based software engineering.

TABLE XII. QUALITY AND TIME (IN MILLISECONDS) FOR SBSE AND HUMANS FOR NRP AND WORK

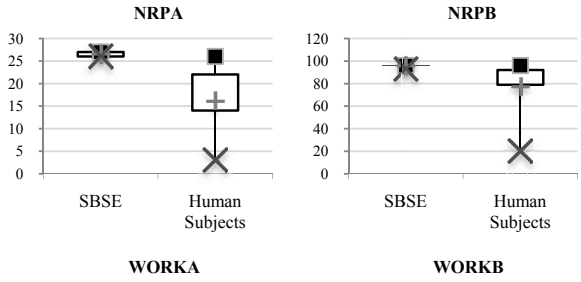
Problem	SBSE		Humans	
	Quality	Time	Quality	Time
NRPA	26.48 ±0.512	40.57 ±9.938	16.19 ±6.934	1,731,428.57 ±2,587,005.57
NRPB	95.77 ±0.832	534.69 ±91.133	77.85 ±23.459	3,084,000.00 ±2,542,943.10
WORKA	16,049.72 ±121.858	260.00 ±50.384	28,615.44 ±12,862.590	2,593,846.15 ±1,415,659.62
WORKB	25,047.40 ±322.085	4,919.30 ±1,219.912	50,604.60 ±20,378.740	5,280,000.00 ±3,400,588.14

Table XII compares the performances of SBSE and humans when solving the mono-objective problems. The results represent the averages and standard deviations of the accuracies of the obtained solutions. For Humans, Table V, presented earlier, shows the number of considered responses. In order to represent SBSE, GA was selected, due to its better performance in terms of quality, as discussed above. For GA, the same number of solutions was generated and averaged. In addition, Table XII presents the averages and standard deviations of the time required by each approach.

SBSE was more accurate than the human subjects in all cases. In terms of time, as expected, the performance of the automated approach is only a fraction of the time required by humans, which, in average, took around 28 minutes to solve each mono-objective problem instance A, and around 69 minutes for instances B.

In order to statistically evaluate the differences on quality averages obtained in the experiments, we have applied the Student´s T-test considering a confidence level of 95%. In all cases, the collected averages were significantly different.

A graphical representation of the results, with respect to quality, can be seen in Figure 3. This graph clearly demonstrates the higher quality of the results generated by SBSE. In addition, it stresses the considerably greater variance in precision of the results generated by humans.



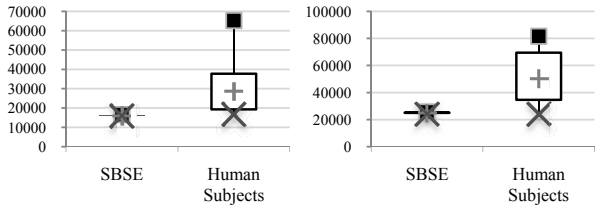


Figure 3. Boxplots showing average (+), maximum (■), minimum (×) and 25% - 75% quartile ranges of quality for mono-objective problems NRP and WORK, instances A and B, for SBSE and Human Subjects.

For multi-objective problems, Table XIII shows, for humans, the hypervolume calculated over the non-dominated set generated from all human solutions and the average time required to solve each instance. For SBSE, the NSGA-II metaheuristic was, once again, employed because of its better performance in terms of hypervolume.

TABLE XIII. HYPERVOLUME AND TIME (IN MILLISECONDS) RESULTS FOR SBSE AND HUMANS FOR MONRP AND TEST

Problem	SBSE		Humans	
	HV	Time	HV	Time
NRPA	0.6519±0.009	1,420.48 ±168.858	0.4448	1,365,000.00 ±1,065,086.42
NRPB	0.6488±0.015	1,756.71 ±138.505	0.2870	2,689,090.91 ±2,046,662.91
WORKA	0.5997±0.009	1,661.03 ±125.131	0.4878	1,472,307.69 ±892,171.07
WORKB	0.6608±0.020	1,693.37 ±138.895	0.4979	3,617,142.86 ±3,819,431.52

Once more, the experimental results presented in Table XIII demonstrate the relative efficiency of SBSE over humans. Both on quality, estimated by the hypervolume metric, as well as for execution time, the metaheuristic NSGA-II significantly outperformed humans in all cases.

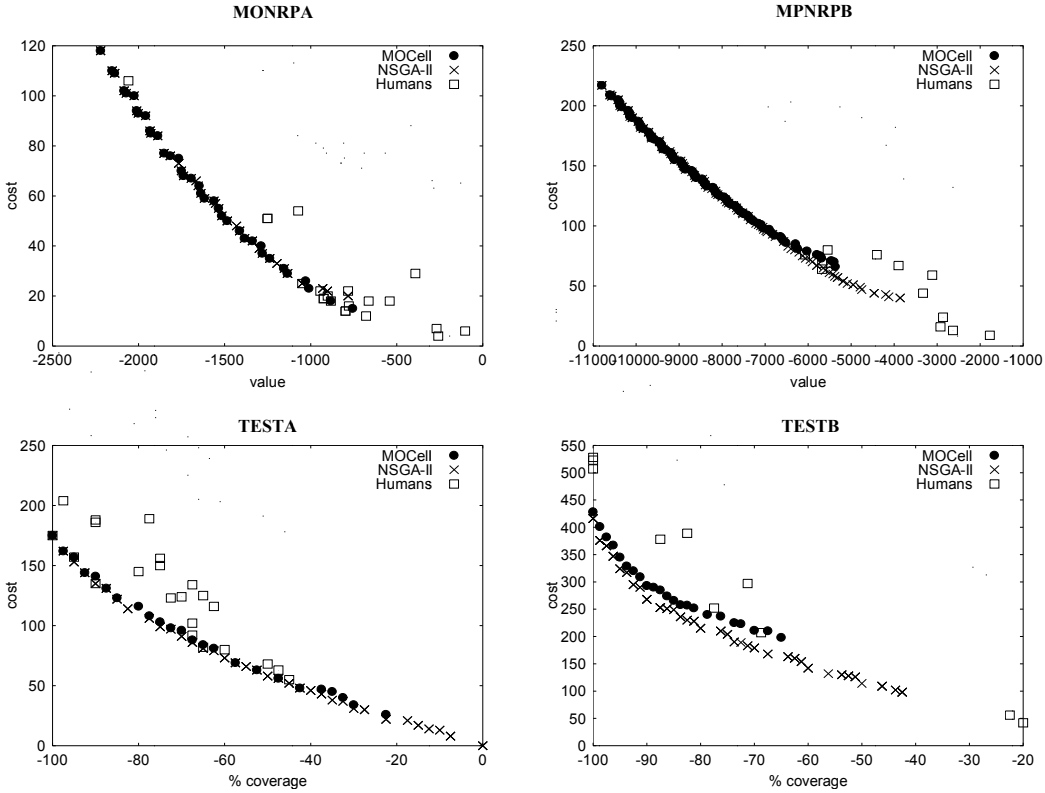


Figure 4. Solutions generated by humans, and non-dominated solution sets produced by NSGA-II and MOCell for problems MONRP and TEST, instances A and B.

The solution set generated by humans is drawn in Figure 4, along with two examples of Pareto Fronts produced by SBSE (NSGA-II and MOCell). As can be seen, graphs confirm the higher quality of the solutions generated by SBSE, as they are closer to the bottom left corner.

All experimental results presented and discussed above indicate the ability of SBSE to generate precise solutions with very little computational effort relative to the results produced by humans, not only for mono- but also for multi-objective problems. These results, in fact, suggest that the results generated by SBSE, are, indeed, human competitive.

C. Further Human Competitiveness Analyses

A more meticulous examination at the experimental results allows one to draw some further interesting conclusions.

1. Human participants were asked to rate how difficult they found each problem instance and how confident they were

on their solutions. For NRP, 35.48% of the respondents answered that this problem was "hard" or "very hard", see Figure 5. For MONRP, this number was 35.29%. Yet for the WORK problem, the percentage of human participants which considered this problem "hard" or "very hard" reached 69.23%. Finally, for TEST, this percentage was 39.28%. These numbers reflect the complexity of the WORK problem, probably because of the relatively high number of aspects (costs, skills and different types of preferences) involved in solving the problem.

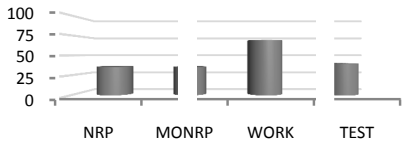


Figure 5. Bar chart showing percentage of human respondents who considered each problem "hard" or very "hard".

2. On the confidence levels (Figure 6), the numbers seem to follow, not surprisingly, the percentages for complexity. For the NRP problem, 87.09% of respondents were "confident" or "very confident" on the quality of their solutions. These percentages are 67.64%, 46.15% and 75.00% for MONRP, WORK and TEST, respectively. Thus, because of the apparent complexity of WORK, humans felt very little confidence in their responses to this problem.

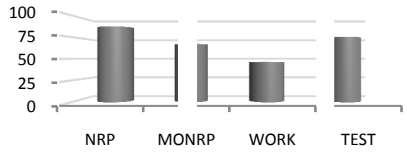


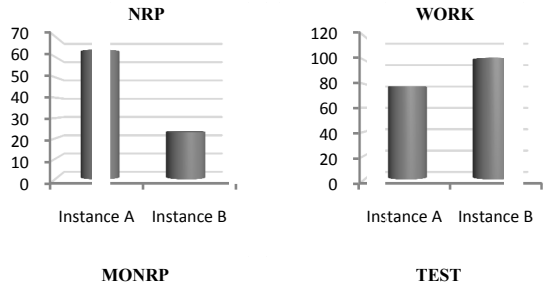
Figure 6. Bar chart showing percentage of human respondents who were "confident" or "very confident" on the quality of his/her solution.

3. As can be drawn from the numbers presented above regarding the complexity of each problem, as pointed out by the human participants, and their confidence levels, there does not seem to exist a feeling of higher complexity for multi-objective problems, as could be expected. In fact, the WORK problem was indicated as the hardest, even by dealing with a single objective.
4. For mono- and multi-objective problems, Figure 7 below shows the percentage differences in quality between the results generated by SBSE and the human subjects. For NRP, the percentage difference varied from 63.55% for instance A to 23.01% for instance B. Yet for problem WORK, this difference grew from 78.29% to 102.03%. For MONRP, the percentages were 46.56% for instance A and 126.06% for the twice-larger instance B. Finally, for TEST, the values were 22.93% and 32.71%, for instances A and B, respectively. Therefore, with the exception of problem NRP, these percentages indicate an increase in the performance difference between SBSE and humans when considering the larger instance. In addition, while 57.33% of the human participants which responded instance A indicated that solving instance B would be "harder" or "much harder", and 55.00% predicted that their solution for this instance would be "worse" or "much worse", 62.50% of the instance B respondents pointed out the increased difficulty of a problem instance twice larger, and 57.14% that their solution would be "worse" or "much worse". These results strongly suggest that, for larger problem instances, the potential of SBSE to generate even more accurate results, when compared to humans, increases. In fact, this suggests that SBSE may be particularly useful in solving real-world large-scale software engineering problems.

D. *Threats to Validity*

As in any research, there are some aspects which could affect the validity of our conclusions. Next, we discuss some of these threats to the results reported in this paper.

1. Small instance sizes: when compared to real-world situations, the instance sizes considered in our experiments may be considered trivial. Even with the indication, based on the collected data, that SBSE can be effective even for large-scale problems, the lack of experimentation on instances of real-world magnitude is a problem.
2. Artificial instances: Similarly, since the instances were synthetically generated, they do not incorporate possible peculiarities of real-world problems. This artificiality could, this way, alter the behavior of the results.
3. Number and diversity of human participants: The number and diversity of human participants is essential to the generalization of the conclusions. Even though we feel that the 128 responses generated from 63 professional software engineers configures a reasonable sample, since the participants were all drawn from same city, and, consequently, had similar educational backgrounds and professional experience, a more diverse sample of human subjects would be beneficial to the validity of the conclusions.



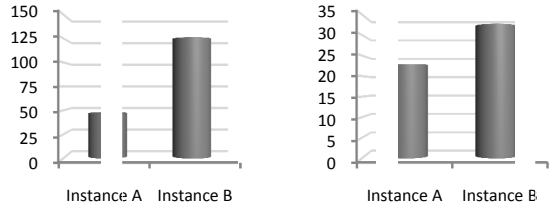


Figure 7. Bar charts showing percentage differences in quality for mono-multi-objective problems generated by SBSE and the human subjects.

4. Number of problems: in order for the research conclusions to be generalized for arbitrary software engineering problems, it is essential to consider as many software engineering problems as possible. In the experiments, four well-known SBSE problems were considered. Since the number of software engineering problems formulated as search problems is considerably higher, the results obtained from this particular set of problems may, somehow, differ for other software engineering problems.

VI. Conclusion and Future Work

This paper reports the results of an extensive experimental research aimed at evaluating the human competitiveness of search based software engineering. Secondarily, several tests were performed over four classical SBSE problems in order to evaluate the performance of well-known metaheuristics in solving both mono- and multi-objective software engineering problems formulated as optimization problems.

Regarding the comparison of algorithms, all results were consistent to show the ability of the Genetic Algorithm to generate more accurate solutions for mono-objective problems than SA. For problems with more than one objective, NSGA-II consistently outperformed MOCell in terms of hypervolume and number of generated solutions, while MOCell outperformed when considering the spread metric and the execution time. All of these results are compatible with previously published research.

On the main research question raised by this work, the results generated by the experiments and reported in this paper strongly suggest that the results generated by search based software engineering can, indeed, be said to be human competitive. In addition, the results indicate that for real-world large-scale software engineering problem, the benefits from applying SBSE may be even greater.

As future works, the experiments could be extended to cover other problems and problem instances. Additionally, the use of real-world instances should evoke other interesting insights as to the applicability of SBSE in real situations.

Acknowledgment

The authors wish to thank all of those who contributed to the production of this paper. We would particularly like to thank the humans who participated in the experiments, dedicating their precious time to this research for believing its relevance and potential impact to the search based software engineering field. We also acknowledge the essential contribution of the following people: Felipe Colares, Thiago Nepomuceno, Thiago de Albuquerque, Rafael Carmo, Isac Albuquerque, Davi França, Leonardo Lopes, Fabiano Gadelha, Marcos Antonio Brizeno, Renan Henry, Samarony Barros, Gizelle Pauline, Márcia Brasil, Igor Sales and Elisa Paollet

References

[1] M. Harman,"The Current State and Future of Search Based Software Engineering", Proceedings of International Conference on Software Engineering / Future of Software Engineering 2007 (ICSE/FOSE '07), Minneapolis: IEEE Computer Society, pp. 342-357, 2007.

[2] J. Koza, F. H. Bennett, and M. A. Keane, "Genetic Programming III : Darwinian Invention and Problem Solving", Morgan Kaufmann Publishers, 1999.

[3] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis, "Search Based Approaches to Component Selection and Prioritization for the Next Release Problem". Proceeding of the 22nd International Conference on Software Maintenance, pp. 176–185, 2006.

[4] F. Colares, J. T. Souza, R. A. Carmo, C. I. P. S. Padua, and G. R. Mateus, "A New Approach to the Software Release Planning". Proceedings of the XXII Brazilian Symposium on Software Engineering Brazilian Symposium on Software Engineering (SBES´2009), Fortaleza, 2009.

[5] A. Barreto, M. Barros, and C. Werner, "Staffing a software project: a constraint satisfaction approach". SIGSOFT Softw. Eng. Notes 30, vol. 4, pp 1-5, July 2005.

[6] A. Bagnall, V. Rayward-Smith, I. Whittley. "The next release problem". Information and Software Technology, vol. 43, pp 883–890, December 2001.

[7] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem". Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07), pp. 1129-1137, July 2007.

[8] G. Burdett, and R. K. Li, R. K, "A quantitative approach to the formation of workgroups". Proceedings of the 1995 ACM SIGCPR Conference on Supporting Teams, Groups, and Learning inside and Outside the IS Function Reinventing IS, pp. 202-212, April 1995.

[9] S. Yoo, and M. Harman, "Pareto efficient multi-objective test case selection". Proceedings of the 2007 international Symposium on Software Testing and Analysis, pp. 140-150, July 2007.

[10] O. Saliu and G. Ruhe, "Software release planning for evolving systems", Innovations in Systems and Software Engineering, vol. 1, no. 2, pp. 189–204, July 2005.

[11] J. Mc Elroy, and G. Ruhe, "Software release planning with time-dependent value functions and flexible release dates", Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, pp. 429-238, 2007.

[12] A. Ngo-The, and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning", Soft Computing - A Fusion of Foundations, Methodologies and Applications, vol. 12, no. 1, pp. 95–108, June 2007.

[13] M. Harman, J. Krinke, J. Ren, and S. Yoo, "Search based data sensitivity analysis applied to requirement engineering", Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 1681-1688, 2009.

[14] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang, "A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making", Requirements Engineering, vol. 14, no. 4, pp. 231–245, December 2009.

[15] J. Ren, "Sensitivity Analysis in Multi–objective Next Release Problem and Fairness Analysis in Software Requirements Engineering", MSc Project Thesis, Dept. Computer Science, King's College, London, 2007.

[16] M. Harman, "Search Based Software Engineering for Program Comprehension", Proceedings of the 15th IEEE International Conference on Program Comprehension, pp.3-13, 2007.

[17] Y. Zhang, A. Finkelstein, and M. Harman, "Search Based Requirements Optimisation: Existing Work and Challenges", Lecture Notes in Computer Science, vol. 5025/2008, Springer Berlin/Heidelberg, pp. 88–94, 2008.

[18] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang, "Fairness Analysis in Requirements Assignments", Proceeding of the 16th IEEE International Requirements Engineering Conference, pp.115-124, 2008.

[19] M. O'Keefe, and M. Cinnéide, "Search-based refactoring: an empirical study", Journal of Software Maintenance and Evolution: Research and Practice, vol. 20, no. 5, pp. 345–364, September 2008.

[20] S. Gueorguiev, M. Harman, and G. Antoniol, "Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering", Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, pp. 1673-1680, 2009.

[21] M. Harman, S. A. Mansouri and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications", Dept. Computer Science, King's College, London, Tech. Rep. TR-09-03, April 2009.

[22] M. Harman, "Automated Test Data Generation using Search Based Software Engineering", Proceedings of the Second International Workshop on Automation of Software Test, pp. 2, May 2007.

[23] H. Zhong, L. Zhang and H. Mei, "An experimental study of four typical test suite reduction techniques", Information and Software Technology, vol. 50, no. 6, pp. 534–546, May 2008.

[24] L. Zhang, S. Hou, C. Guo, T. Xie, and H. Mei, "Time-aware test-case prioritization using integer linear programming", Proceedings of the eighteenth international symposium on Software testing and analysis, pp. 213-224, 2009.

[25] P. McMinn, "Search-based failure discovery using testability transformations to generate pseudo-oracles", Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 1689-1696, 2009.

[26] S. Biswas, R. Mall, M. Satpathy and S. Sukumaran, "A model-based regression test selection approach for embedded applications", ACM SIGSOFT Software Engineering Notes, vol. 34, no. 4, pp. 1–9, July 2009.

[27] J. C. B. Ribeiro, M. A. Zenha-Rela, and F. F. Vega, "Test Case Evaluation and Input Domain Reduction strategies for the Evolutionary Testing of Object-Oriented software", Information and Software Technology, vol.51, pp.1534-1548, November 2009.

[28] M. Harman, "Automated Test Data Generation using Search Based Software Engineering", Proceedings of the Second International Workshop on Automation of Software Test, pp. 2, 2007.

[29] W. B. Langdon, M. Harman and Y. Jia, "Multi Objective Higher Order Mutation Testing with Genetic Programming", 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, pp. 21-29, 2009.

[30] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the Class Responsibility Assignment Problem in Object-oriented Analysis with Multi-Objective Genetic Algorithms", Carleton University, Ottawa, Tech. Rep. SCE-07-02, August 2008.

[31] S. P.G., and H. Mohanty, "Automated Test Scenario Selection Based on Levenshtein Distance", Lecture Notes in Computer Science, vol. 5966/2010, Springer Berlin/Heidelberg, pp. 255–266, 2010.

[32] K. Praditwong, M. Harman, X. Yao, "Software Module Clustering as a Multi-Objective Search Problem", IEEE Transactions on Software Engineering, Jan. 2010.

[33] C. Maia, R. Carmo, F. Freitas, G. Campos, and J. Souza, "Automated Test Case Prioritization with Reactive GRASP", Advances in Software Engineering, vol. 2010, January 2010.

[34] M. Harman, S. G. Kim, K. Lakhotia, P. McMinn and S. Yoo, "Optimizing for the Number of Tests Generated in Search Based Test Data Generation with an Application to the Oracle Cost Problem", Proceeding of the 3rd International Workshop on Search-Based Software Testing, 2010.

[35] L. Raamesh, and G. V. Uma, "Knowledge Mining of Test Case System", International Journal on Computer Science and Engineering, vol. 2, no. 1, pp. 69–73, January 2008.

[36] H. Wang, W. K. Chan, and T. H. Tse, "On the Construction of Context-Aware Test Suites", Dept. Computer Science, University of Hong Kong, Hong Kong, Tech. Rep. TR-2010-01, March 2010.

[37] S. Yoo and, M. Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation", Journal of Systems and Software, vol. 83, no. 4, pp. 689–701, April 2010.

[38] S. Poulding, and J. A. Clark, "Efficient Software Verification: Statistical Testing Using Automated Search", IEEE Transactions on Software Engineering, Jan. 2010.

[39] J. Ribeiro, M. Zenha-Rela, and F. Vega, Test Case Evaluation and Input Domain Reduction strategies for the Evolutionary Testing of Object-Oriented software", Information and Software Technology, vol. 51, no. 11, pp. 1534–1548, November 2009.

[40] U. Faroog, and C. P. Lam, "Evolving the Quality of a Model Based Test Suite", Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops, pp. 141-149, 2009.

[41] S. Yoo, M. Harman and, S. Ur, "Measuring and Improving Latency to Avoid Test Suite Wear Out", in Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops, pp. 101-110, 2009.

[42] H. Wang, and W. K. Chan, "Weaving Context Sensitivity into Test Suite Construction", Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, pp.610-614, 2009.

[43] S. Yoo, and M. Harman, "Regression Testing Minimisation, Selection and Prioritisation - A Survey", Dept. Computer Science, King's Collge, London, Tech. Rep. TR-09-09, October 2009.

[44] J. Holland, "Adaptation in natural and artificial systems". University of Michigan Press, 1975.

[45] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by simulated annealing", Science, vol. 220, pp. 671-680, 1983.

[46] K. D. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm : NSGA-II", Evolutionary Computation, IEEE Transactions on, vol. 6, no. 2, pp. 182-197, August 2002.

[47] Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. "MOCell: A cellular genetic algorithm for multiobjective optimization". Int. J. Intell. Syst. vol. 24, pp. 726-746, Jul. 2009.

[48] J. Singer, and N. G. Vinson, "Ethical Issues in Empirical Studies of Software Engineering". IEEE Trans. Softw. Eng., vol. 28, pp. 1171-1180, Dec. 2002.

[49] ACM Executive Council, "ACM Code of Ethics and Professional Conduct", Communications of the ACM, vol. 36, pp. 99-105, 1993.

[50] IEEE Ethics Committee, "IEEE Code of Ethics", 2006,

[51] IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices. "Software Engineering Code of Ethics and Professional Practice", 1998.

[52] A. J. Nebro, J. J. Durillo, C. A. Coello, F. Luna, and E. Alba, E. "A Study of Convergence Speed in Multi-objective Metaheuristics". Proceedings of the 10th international Conference on Parallel Problem Solving From Nature: PPSN X (Dortmund, Germany, September 13 - 17, 2008). G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, Eds. Lecture Notes In Computer Science, vol. 5199. Springer-Verlag, Berlin, Heidelberg, pp. 763-772, 2008.

[53] J. J. Durillo, Y. Zhang, E. Alba and, A. J. Nebro, "A Study of the Multi-objective Next Release Problem", Proceedings of the 2009 1st International Symposium on Search Based Software, pp. 49-58, 2009.

2nd International Symposium on Search Based Software Engineering

Concept Location with Genetic Algorithms: A Comparison of Four Distributed Architectures

Fatemeh Asadi*, Giuliano Antoniol*, Yann-Gaël Guéhéneuc†
*SOCCER Lab. – DGIGL, École Polytechnique de Montréal, Québec, Canada
†Ptidej Team – DGIGL, École Polytechnique de Montréal, Québec, Canada
{fatemeh.asadi,yann-gael.gueheneuc}@polymtl.ca
antoniol@ieee.org

***Abstract*—Genetic algorithms are attractive to solve many search-based software engineering problems because they allow the easy parallelization of computations, which improves scalability and reduces computation time. In this paper, we present our experience in applying different distributed architectures to parallelize a genetic algorithm used to solve the concept identification problem. We developed an approach to identify concepts in execution traces by finding cohesive and decoupled fragments of the traces. The approach relies on a genetic algorithm, on a textual analysis of source code using latent semantic indexing, and on trace compression techniques. The fitness function in our approach has a polynomial evaluation cost and is highly computationally intensive. A run of our approach on a trace of thousand methods may require several hours of computation on a standard PC. Consequently, we reduced computation time by parallelizing the genetic algorithm at the core of our approach over a standard TCP/IP network. We developed four distributed architectures and compared their performances: we observed a decrease of computation time up to 140 times. Although presented in the context of concept location, our findings could be applied to many other search-based software engineering problems.**

***Keywords*-Concept location; dynamic analysis; information retrieval; distributed architectures.**

I. Introduction

Genetic algorithms (GAs) are an effective technique to solve complex optimization problems. GAs are effective in finding approximate solutions when the search space is large or complex, when mathematical analysis or traditional methods are not available, and—in general—when the problem to be solved is NP-complete or NP-hard [1]. Informally, a GA may be defined as an iterative procedure that searches for the best solution to a given problem among a constant-size population, represented by a finite string of symbols, the *genome*. The search starts from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a *fitness function*. Highly fit individuals have the highest probability to reproduce in the next generation. GAs have been applied to many software engineering problems; from library miniaturization [2], to project staffing [3], to test input data generation [4], to software refactorings [5].

One of the attractive feature of GAs is that the evaluation of the fitness function is often performed on each individual in isolation: to assign its fitness value to an individual, the GA only needs its genome representation because there are no interactions with other individuals in the population. Such an isolation in the evaluation of the fitness function leads naturally to parallelize computations of the fitness function to reduce computation time [6], [7], [8].

In this paper, we report our experience in distributing the computation of a fitness function to parallelize a GA to solve the concept location problem. To the best of our knowledge, this is the first time that GA parallelization via the distribution of the fitness function computation is applied to solve the concept location problem.

Concept location approaches help developers perform their maintenance and evolution tasks by identifying abstractions (*i.e.*, concepts or features) and the location of the implementation of these abstractions in source code. They aim at identifying *code fragments*, *i.e.*, set of method calls in traces and the related method declarations in the source code, responsible for the implementation of domain concepts and–or user-observable features [9], [10], [11], [12], [13].

In [14], we presented an approach to identify cohesive and decoupled fragments in execution traces, which likely participate in implementing concepts related to some features. The approach builds upon previous concept location approaches [15], [16], [13], [12], [17] and uses a GA to automatically locate cohesive and decoupled fragments. Although promising, our approach is computationally intensive and suffers from scalability issues.

To resolve the scalability issues of our approach, we developed, tested, and compared four different architectures where a client (master) distributes the computation of the fitness function among servers (slaves) over a TCP/IP network. To our surprise, the most effective architecture to reduce computation time defines servers that only use local data and do not share data and–or results with other servers.

Consequently, the contribution of this paper is an application of GA parallelization to a software engineering problem and the comparison and discussion of our findings for four different architectures. Although presented in the context

978-0-7695-4195-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SSBSE.2010.26

of concept location, our findings could be applied to other search-based software engineering problems.

The remainder of the paper is organized as follows: Section II presents related work followed by Section III where the concept location problem is summarized. Section IV describes the approach to speed up computation. Section V reports the results and some discussions. Section VI concludes the paper and outlines some future work.

II. Related Work

This paper focuses on the parallelization of a GA using a distributed architecture to reduce the computation time of an approach to solve the concept location problem. Therefore, we focus in the following on previous work related to the concept location problem (*i.e.*, feature identification), to the distribution of optimizations in software engineering, and to the parallelization of GAs in other domains.

A. Feature Identification

In their pioneering work, Wilde and Scully [16] presented the first approach to identify features by analyzing execution traces. They used two sets of test cases to build two execution traces, one where a feature is exercised and another where the feature is not. They compared the execution traces to identify the feature in the system. Similarly, Wong *et al.* [18] analyzed execution slices of test cases to identify features in source code. Wilde's original idea was later extended in several works [9], [12], [19], [20] to improve its accuracy by introducing new criteria on selecting execution scenarios and by analyzing the execution traces differently. Search based techniques have been used by Gold *et al.* [21] for concept binding, the work extends a previous contribution [22] and uses hill climbing and GA to locate (possibly overlapping) concepts in the source code.

More recent works focused on a combination of static and dynamic data [17], [12], in which, essentially, the problem of features identification from multiple execution traces is modelled as an information-retrieval (IR) problem, which has the advantage to simplify the identification process and, often, improves its accuracy [12]. Yet, Liu *et al.* [23] showed that a single trace suffices to build an IR system and identify useful features. Execution traces were also used to mine aspects by Tonella and Ceccato [13].

We share with this previous work the use of dynamic data and IR techniques to identify features. In our approach, we determine, in an execution trace the cohesive and decoupled fragments likely to be relevant to a feature using the values of the conceptual cohesion and coupling [24], [25] metrics of the methods participating in each fragment. The computational costs of conceptual cohesion and coupling together with the size of the execution traces are at the root of the scalability issues of our approach.

B. GA Parallelization in Software Engineering

A limited number of works in software engineering addressed complex optimization problems by distributing computations among several servers. Mitchal *et al.* [26] proposed an approach to remodularize large systems by grouping together related components by means of clustering techniques. They used different search strategies based on hill-climbing and GAs. To improve the performance of their approach, they distributed the hill-climbing computations.

More recently, Mahdavi *et al.* [27] used a distributed hill-climbing for software module clustering. The fitness function clusters together modules that are cohesive and decoupled from the other clusters. The algorithm was parallelized on 23 processing units running Linux.

C. GA Parallelization in Other Domains

The literature on the parallel implementation of GAs reports that parallelization does not influence the quality of results but makes GA execution much faster.

Parallel GAs have been to solve problems in different domains. For example, parallel GAs were used for shortest-path routing [28], multi-objective optimization [29], finding roots of complex functional equation [30], image restoration [31], service restoration in electric power distribution [32], and rule discovery in large databases [33].

The scalability of a parallel system refers to its ability to use an increasing number of processors (and–or computers) in an effective way. Rivera [7] discussed the scalability of parallel GAs based on their *iso-efficiency*, which is defined according to the problem size, number of processors, and the execution time of the parallel algorithm. A parallel system is scalable iff it uses an iso-efficient fitness function.

Stender *et al.* [6] classified parallel GAs into three categories, each one using a different parallelization strategy. In the category of global parallelization, only the evaluation of the individuals' fitness is parallelized: a computer acting as master applies the genetic operators on the individuals' genomes and distributes the individuals among slave computers, which compute the fitness values of the individuals.

In the category of coarse-grained parallelization (island model), a computer divides a population into sub-populations and assigns each sub-population to another computer. A GA is executed on each sub-population. When it is needed, the computers exchange data related to the sub-populations using a migration process. This model inspired Zorman *et al.* [34]: they used a Java service-oriented architecture to implement the island model using a migration process to solve the knapsack problem.

In the category of fine-grained parallelization, each individual is assigned to a computer and all the GA operations are performed in parallel. Our approach to GA parallelization of the concept location problem falls under this category: it is essentially a global parallelization where servers are in charge of computing fitness values. Moreover,

our work is the first to presents four distributed architectures and their related trade-offs for the computation of the fitness function to parallelize the concept location problem.

III. Background

This section summarizes our approach [14] to locate concepts by analyzing execution traces. We provide details of our approach for the sake of completeness and because they are necessary to understand the rationale behind the four different architectures that we implemented.

Our concept location approach consists of five steps. First, the system under analysis is instrumented. Second, it is exercised to collect execution traces. Third, the collected traces are compressed to reduce the search space that must be explored to identify concepts. Fourth, each method of the system is represented by means of the text that it contains. Fifth, a GA-based technique is used to identify, within execution traces, sequences of method invocations that are related to a concept.

A. Steps 1 and 2 – System Instrumentation and Trace Collection

First, the software system is instrumented using the *instrumentor* of MoDeC. MoDeC is a tool to extract and model sequence diagrams from Java systems [35]. MoDeC instrumentor is a dedicated Java bytecode modification tool implemented on top of the Apache BCEL bytecode transformation library[1]. It inserts appropriate and dedicated method invocations in the system to trace method/constructor entries/exits, taking care of exceptions and system exits. It also allows the user to add tags containing meta-information to the traces, *e.g.*, tags delimiting and labelling sequences of method calls related to some specific features being exercised. Resulting traces are text files listing method invocations and including the class of the object caller, the unique ID of the caller, the class of the receiver, the unique ID of the callee, and the complete signature of the method.

B. Step 3 – Pruning and Compressing Traces

Usually, execution traces contain methods invoked in most scenarios, *e.g.*, methods related to logging or start-up and shut-down. In the execution trace of a system with a graphical user interface, mouse tracking methods will largely exceed all other method invocations. Yet, it is likely that such methods are not related to any particular concept, *i.e.*, they are utility methods. We filter out these utility methods using the distributions of the frequencies of their occurrences.

Moreover, traces often contain repetitions of one or more method invocations, for example `m1(); m1(); m1();` or `m1(); m2(); m1(); m2();`. A repetition does not introduce a new concept and makes a trace longer that necessary to locate concepts. Consequently, we compress traces using the Run Length Encoding (RLE) algorithm to remove repetitions and keep one occurrence of any repetition only. The previous examples would become `m1()` and `m1(); m2()`, respectively. We compression any sub-sequences of method invocations having an arbitrary length.

[1] http://jakarta.apache.org/bcel/

Table I
Example of GA individual representation (second column).

Method Invocations	Repr.	Segments
TextTool.deactivate()	0	
TextTool.endEdit()	0	
FloatingTextField.getText()	0	
TextFigure.setText-String()	0	1
TextFigure.willChange()	0	
TextFigure.invalidate()	0	
TextFigure.markDirty()	1	
TextFigure.changed()	0	
TextFigure.invalidate()	0	
TextFigure.updateLocation()	0	2
FloatingTextField.endOverlay()	0	
CreationTool.activate()	1	
JavaDrawApp.setSelectedToolButton()	0	
ToolButton.reset()	0	
ToolButton.select()	0	
ToolButton.mouseClickedMouseEvent()	0	3
ToolButton.updateGraphics()	0	
ToolButton.paintSelectedGraphics()	0	
TextFigure.drawGraphics()	0	
TextFigure.getAttributeString()	1	

C. Step 4 – Textual Analysis of Method Source Code

To determine the conceptual cohesion and coupling of invoked methods, our approach uses the metrics defined by Marcus *et al.* [24], [25]. We extract a set of terms from each method by tokenizing the method source code and comments, pruning out special characters, programming language keywords, and terms belonging to a stop-word list for the English language. (We assume that comments appearing on top of the method declaration belong to the following method.)

We then split compound terms based on the Camel Case naming convention at each capitalized letter, *e.g.*, `getBook` is split into `get` and `book`. Then, we stem the obtained simple terms using a Porter stemmer [36].

Once terms belonging to each method extracted, we index these terms using the *tf-idf* indexing mechanisms [37]. We thus obtain a term–document matrix, where documents are all methods of all classes belonging to the system under study and where terms are all the terms extracted (and split) from the method source code. Finally, we apply Latent Semantic Indexing (LSI) [38] to reduce the term–document matrix into a concept–document matrix.

We follow previous process and suggestion [24], [25] when computing the conceptual cohesion and coupling of methods in a class in the LSI subspace to deal with synonymy, polysemy, and term dependency. We choose a size of 50 for the LSI subspace.

$$SegmentCohesion_k = \frac{\sum_{i=begin(k)}^{end(k)-1}\sum_{j=i+1}^{end(k)} similarity(method_i, method_j)}{(end(k)-begin(k)+1)\cdot(end(k)-begin(k))/2} \quad (1)$$

$$SegmentCoupling_k = \frac{\sum_{i=begin(k)}^{end(k)}\sum_{j=1,j<begin(k)\ or\ j>end(k)}^{l} similarity(method_i, method_j)}{(l-(end(k)-begin(k)+1))\cdot(end(k)-begin(k)+1)} \quad (2)$$

$$fitness(individual) = \frac{1}{n}\cdot\sum_{k=1}^{n}\frac{SegmentCohesion_k}{SegmentCoupling_k} \quad (3)$$

D. Step 5 – Search-based Concept Location

We now have all the data to segment execution traces into conceptually-cohesive and -decoupled segments related to a feature being exercised and, thus, to a specific concept.

1) Problem Definition: Suppose that the collected trace contains N methods; determining a (near) optimal solution (splitting a trace into segments) means exploring a search space of all possible binary strings, of length N, that do not contain two consecutive 1. In other words, the order of the problem search space is 2^N and, therefore, we use a GA to perform the splitting.

At each step of the GA, individuals are evaluated using a *fitness function* and selected using a *selection mechanism*. Highly fit individuals have the highest reproduction probability. The evolution (*i.e.*, the generation of a new population) is affected by the *crossover operator* and the *mutation operator*.

2) Problem Representation: Our representation of an individual is a bit-string of the length of the compressed execution trace in which we want to identify some feature-related concepts. Each method invocation is represented as a "0", except the last method invocation in a segment, which is represented as a "1". For example, the bit-string

$$\underbrace{00010010001}_{11}$$

means that the trace, containing 11 method invocations, is split into three segments (*i.e.*, concepts) composed by the first four method invocations, the next three, and the last four. Table I shows an example of a real segment splitting[2].

Other representations could be more compact, for example, a book keeping of segments beginnings and ends. The disadvantage of such representation is that mutation and crossover would be more complex and costly in time. Among different representations, we found that the bit-string representation is suitable to large traces: even for a trace of one million method calls and hundreds of individuals, memory requirement is still manageable on a standard PC. Moreover, the bit-string representation allows to easily understand the size of the search space, which is roughly related to the number of bit strings.

[2]The segment splitting shown in Table I has been obtained randomly and does not correspond to actual concepts.

3) Mutation: The mutation operator prevents the convergence to a local optimum: it randomly modifies an individual's genome (*e.g.*, by flipping some of its symbols). The mutation operator randomly chooses one bit in the representation and flips it over. Flipping a "0" into a "1" means splitting an existing segment into two segments, while flipping a "1" into a "0" means merging two consecutive segments. Mutation operator is thus implemented with constant time complexity.

4) Crosssover: The crossover operator takes two individuals (the *parents*) of one generation and exchanges parts of their genomes, producing one or more new individuals (the *offspring*) in the new generation. The crossover operator is the standard 2-point crossover. Given two individuals, two random positions x, y with $x < y$ are chosen in one individual's bit-string and the bits from x to y are swapped between the two individuals to create two new offsprings. Crossover operator is thus implemented with linear time complexity in the length of the bit-string individual representation.

5) Fitness Function: A fitness function drives the GA to produce individuals that represent a splitting of the trace into segments that are related to some concepts. We use the software design principles of cohesion and coupling, already adopted in the past to identify modules in systems [39].

However, instead of structural cohesion and coupling measures, we use conceptual (*i.e.*, textual) cohesion and coupling measures [24], [25]. Segment cohesion is the average (textual) similarity between any pair of methods in a segment k and is computed using the formulas in Equation 1 where $begin(k)$ is the position (in the individual's bit-string) of the first method invocation of the k^{th} segment and $end(k)$ the position of the last method invocation in that segment. The similarity between two methods is computed using the cosine similarity measure over the LSI matrix extracted in the previous step. Thus, it is the average of the similarity [24], [25] to all pairs of methods in a given segment.

Segment coupling is the average similarity between a segment and all other segments in the trace, computed using Equation 2, where l is the trace length. Segment coupling represents, for a given segment, the average similarity between methods in that segment and those in different ones.

Cohesion and coupling have quadratic costs in the trace

length, plus each similarity computation between a pair of methods involves a scalar product in the LSI subspace, with a cost proportional to d, the number of retained LSI dimensions. Thus, when compared with the bit operations required to perform mutation (constant time) and crossover (linear time), it is evident that the main source of complexity and computation costs come from Equations 1 and 2 that have polynomial time complexity in the bit-string individual representation, *i.e.*, number of methods in the trace. For a trace split into n segments, the fitness function is shown in Equation 3.

6) GA Parameters: We use a simple GA with no elitism, *i.e.*, it does not guarantee to retain best individuals across subsequent generations; the selection operator is the roulette-wheel selection. We set the population size to 200 individuals and a number of generations of 2,000. Crossover and mutation are respectively performed on each individual of the population with probability *pcross* and *pmut* respectively, where $pmut \ll pcross$. The crossover probability was set to 70% and the mutation to 5%, which are values widely used in many GA applications.

IV. GA and Distributed Architecture

We started our experiments with a basic GA implementation running on a single computer. We found that computations were overly time consuming, impairing the possibility to actually obtain results in a reasonable amount of time. As an example, running an experiment with a compressed trace from JHotDraw v5.4b2, and the scenario *Start-DrawRectangle-Quit*, that contains 240 method calls, with a number of iterations equal to 2,000, took about 12 hours.

We could expect a substantial improvement by parallelizing computations on several computers. However, according to Amdahl's law [40], the performance increase is not linear with the number of computers due to the sequential code, *e.g.*, mutation and crossover. In addition, network latency, available bandwidth between computers and, in general, available resources complicate performance prediction and could lessen time reduction. A detailed study of performance in function of network latency, number of computers, and speed-up is out of scope of this paper and will be treated in a future work. Yet, we report the user-experienced speed-up obtained with different architectures.

Table II
Example of individual coding and segment redundancy

Id1	0001 0001 0000001 0001 0001
Id2	0001 0001 001 0001 0001 0001
Id3	001 00001 0000000001 0001
Id4	00001 0001 0000001 000001
Id5	0001 0001 0000001 0001 001 01

To reduce computation time, we decided to resort on the client–server architectural style [41], customized into more

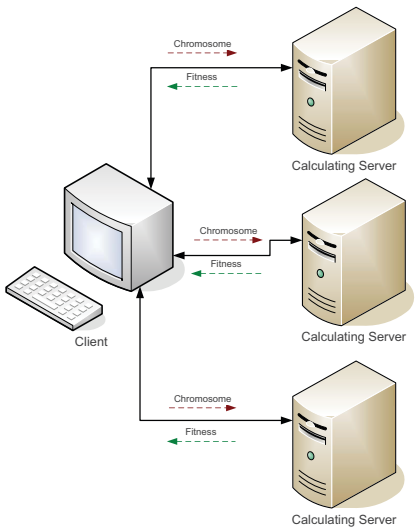


Figure 1. Baseline Client Server Configuration.

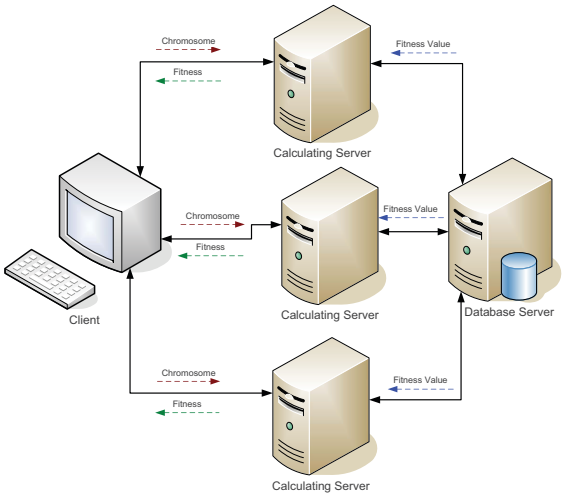


Figure 2. DBMS Client Server Configuration.

specific architectures detailed in the following. The rationale behind the different architectures comes from the illustrative population shown in Table II: several individuals share some segments. For example, the first two segments of individuals Id1, Id2, and Id5 are identical (*i.e.*, beginning and end are the same); Id1 and Id5 are almost identical but for their last segments. Thus, once Id1's fitness value is calculated, if segment cohesion and coupling were stored, they could be reused to compute the fitness values of Id2 and Id5.

In the following, we minimally define that a client computer (master in Stender's work [6]), performing mutation, crossover, and population evolution, distributes fitness computation to multiple servers, which compute the received individual's fitness value and return it to the client.

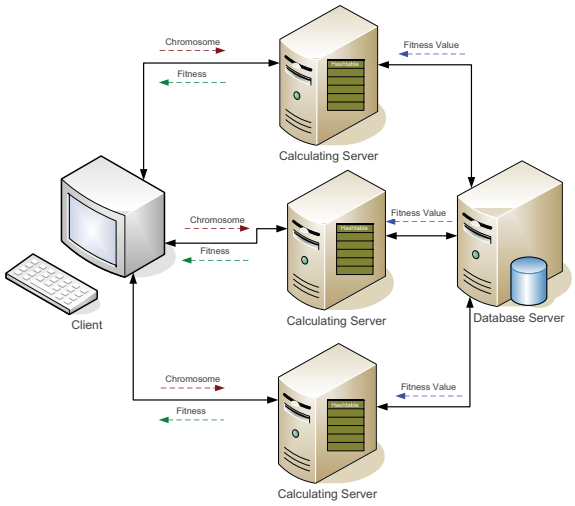


Figure 3. Database-Hash table Configuration for GA-Concept Identifier

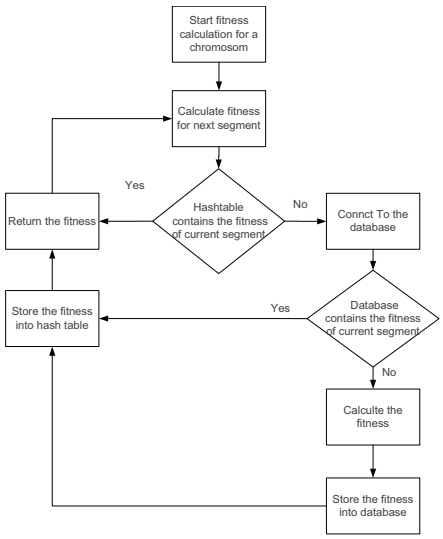


Figure 4. Flow chart of Database-Hash table configuration process

A. A Simple Client Server Architecture

The simplest distributed client–server architecture is shown in Figure 1. The servers have no local memory, do not communicate among themselves or store data locally or on a global and shared device. The client sends the individuals' encodings to the servers and waits for the fitness values to be returned. Each server has only its own local LSI matrix and computes fitness values based on the equations presented in the previous section.

B. A Database Client Server Architecture

Figure 2 shows the architecture of a client–server in which a database server stores global shared storage device. When a segment cohesion or coupling value is required, a server

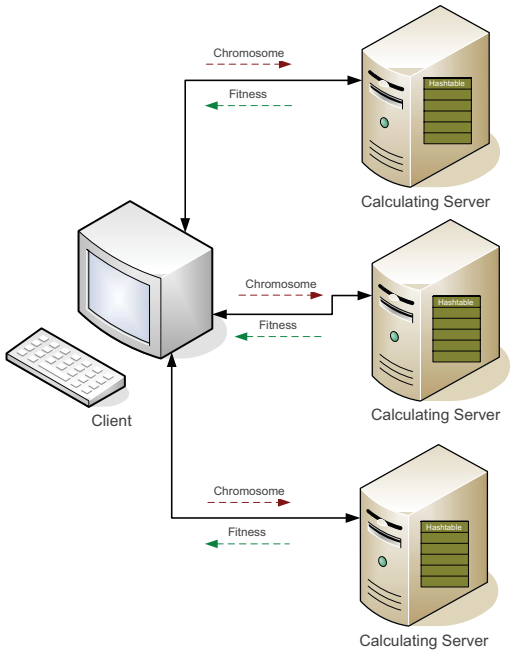


Figure 5. Hash Table Client Server Configuration.

first queries the database before computing it if missing.

The database holds two tables: a cohesion table and a coupling table, each with three columns. Each record in these tables keeps a similarity/coupling value for one segment. The first column, called beginning, keeps the index of the first method invocation in a segment and the second column keeps the index of the last method invocation in the same segment. The third column contains the cohesion/coupling value of that segment.

Whenever the fitness value for a new individual must be computed, the responsible server checks first the database. If it can find the needed values (already calculated in the last iterations or by other servers for other individuals), it uses these to compute the fitness value using a simple division. Else, it computes cohesion and coupling for the new segment and stores the values in the database. Thus, computation is performed if and only if the values can not be retrieved from the database: as much data as possible is shared between servers to reduce computation times.

There is an extra cost due to database queries and network communication. A central database implies that all servers write in and read from the same database. Yet, we would expected that using a database reduces the computation times by caching already-calculated values. However, sending data over the network, acquiring and releasing locks, and performing queries are also time consuming operations.

C. A Hash-database Client Server Architecture

To limit the possible communication between servers and the database, the architecture shown in Figure 3 was devised.

The goal of this architecture is to further reduce computation time by decreasing the number of accesses to the central database using a local cache on each server, implemented with a hash table.

The architecture works as follows: whenever a server wants to compute the fitness value of a segment, it searches its hash table. If the required data does not exist in its local hash table, then the server queries the central database. If the server finds the required data in the database, it uses it to compute the fitness value and and stores it in its hash table, else it computes the required data and stores the results in both its hash table for its future use and in the central database for the other servers use. Figure 4 reports the flowchart of the process of this architecture.

D. A Hash Client Server Architecture

This last architecture is a compromise between the two previous ones: only local data is stored in the local hash table of servers. No data is shared among servers. As shown in Figure 5, servers only communicate with the client and no global data is kept and available.

Each server has two hash tables: one for similarity cohesion and the other for coupling values for each segment. The key of the hash tables is a combination of the indexes of the first and last method invocations of a segment. Each server uses its own hash tables and thus cannot benefit from the computation results of others. However, because all the data is stored locally and there is no access policy using locking algorithms, the access to the already-calculated data as well as their storage is efficient.

V. Results and Discussion

We now report the typical timing obtained with the different architectures on two compressed traces from JHotDraw.

JHotDraw[3] is a Java framework for drawing 2D graphics. JHotDraw started in October 2000 with the main purpose of illustrating the use of design patterns in a real context. Version v5.4b2 used in our previous work [14] has a size of about 413 KLOCs.

The traces were collected by instrumenting JHotDraw and executing the scenarios *Start-DrawRectangle-Quit* and *Start-Spawn-Window-Draw-Circle-Stop*. These scenarios generated respectively traces of 6,668 and 16,366 method calls; once utility methods were removed their sizes are reduced to 447 and 670 calls. Finally RLE compression brought down the numbers of distinct calls to 240 and 432.

In our experiments, we distributed computations over a sub-network of 14 workstations. Five high-end workstations, the most powerful ones, are connected in a Gigabit Ethernet LAN; low-end workstations are connected to a LAN segment at 100 MBit/s and talk among themselves at 100 Mbit/s. Each experience was run on a subset of ten computers: nine servers and one client.

[3]http://www.jhotdraw.org

Workstations run CentOS v5 64 bits; memory varies between four to 16 Gbytes. Workstations are based on Athlon X2 Dual Core Processor 4400; the five high-end workstations are either single or dual Opteron. Workstations run basic Unix services (*e.g.*, network file system, SAMBA, mySQL) and user processes. User processes are typically editing and compilation of programs, e-mail clients, Web browsers, and so on. No special care was taken to ensure a specific network condition (*e.g.*, priorities were not altered) and thus times and ratios between times can be considered typical of a industrial or research environment. However, the sizes of the GA processes never exceeded the physical memory of the workstations to avoid paging; workstations were managed to ensure that each computationally-intensive user processes had a dedicated CPU.

The client computer was also responsible to measure execution times and to verify the liveness of connections; connections to servers as well as connections to the database were implemented on top of TCP/IP (AF_INET) sockets. All components have been implemented in Java 1.5 64bits. The database server, shown in Figures 2 and 3, was MySQL server v5.0.77.

Table III
Computation times for desktop solution and the different architectures of Figures 1, 2, and 5 with the *Start-DrawRectangle-Quit* scenario – Compressed trace length of 240 methods

Time Measurement			
Architectures	Runs #	Measures	Average
Desktop	1	12:09 h	12:07 h
	2	11:39 h	
	3	12:21 h	
	4	11:50 h	
	5	12:38 h	
Client–server	1	1:44 h	2:01 h
	2	2:36 h	
	3	1:53 h	
	4	1:40 h	
	5	2:13 h	
Database	1	16:36 h	13:50 h
	2	15:3 h	
	3	9:52 h	
Hash Table	1	5:13 m	5:17 m
	2	5:19 m	
	3	5:20 m	
	4	5:27 m	
	5	5:10 m	

Table III reports computation times for the different architectures. The times reported for the single-computer architecture come from an optimized implementation of our approach. In our first implementation, we reused the Java GALib library, which is freely available from SourceForge and implements a simple GA. GALib makes no assumptions on crossover and mutation operators and assumes that the fitness of an individual must be recomputed even if it was passed unchanged from the old generation to the new one. This recomputation resulted in about 30% of computation-time increase because between 20% and 30% of individuals

Table IV
COMPUTATION TIMES FOR DESKTOP SOLUTION AND THE ARCHITECTURE OF FIGURE 5 WITH THE *Start-Spawn-Window-Draw-Circle-Stop* SCENARIO – COMPRESSED TRACE LENGTH OF 432 METHODS

Time Measurement			
Architectures	Runs #	Measures	Average
Desktop	1	45:38 h	44:07
	2	41:28 h	
	3	45:07h	
Hash Table	1	7:21 m	7:24 m
	2	7:21 m	
	3	7:32 m	

are not subject to mutation or crossover between generations. Thus, to reduce computation time, we modified GALib to compute only the fitness values of individuals that have changed between the last generation and the current one.

Distributing the computation, shown in Figure 1, clearly results in an important reduction of computation time; as shown in the second row of Table III. Computation time went from 12 hours to about two hours; however, the gain in terms of time reduction is considerably lower than expected as we had nine computers available (excluding the client) and, thus, expected computation times close to one hour.

We felt that there was still room for improvement and Amdahl's law [40] was only partially the reason for the reduced gain. We observed that the nature of our problem was such that crossover and mutation preserve a large fraction of segments unchanged and that for those segments, previous cohesion and coupling values could be reused.

Thus, we tested the two architectures in Figures 2 and 3. Table III in its third row reports results for such database client–server architecture: to our surprise, sharing data among servers via a central database increased computation times.

Finally, Table III, in its last row, reports the computation times for the architecture in Figure 5, which is the fastest architectures. The gain in computation times obtained is of about 140 times. The implementation of this GA parallelization is moreover relatively simple.

We obtained similar gains with other traces. For example, the trace generated by the scenario *Start-Spawn-Window-Draw-Circle-Stop*, with the desktop architecture, was split in about 44 hours while, with the fastest architecture, the client–server with the hash table, computation time is of about 7 minutes. Table IV reports the results of splitting the trace with two architectures.

A. Discussion

We conjecture that poor performance of the database architecture, in Figure 2, is mainly due to the database accesses (reading, writing, and locking) for the computation of each coupling and cohesion values. These frequent accesses are responsible for the increase in computation times. To limit the number of database accesses, we introduced the hybrid architecture in Figure 3. Results have not been reported in Table III because they are not substantially different (better) then those of the database. We are investigating the reason of this unexpected behavior to locate the bottleneck cause.

Indeed, in our current implementation, accesses to the local hash table and the database are managed serially. Performance could improve by parallelizing writing in the database and access to the hash table and by loading the hash table only once at the beginning of each computation. Unfortunately, given the size of the search space and the huge number of possible segments, the probability that in two consecutive runs a relevant number of the segments will be exactly the same is very low. This fact makes the architecture in Figure 3 interesting from a theoretical point of view but not practical.

Despite the decrease in computation time, the very definition of the concept location problem makes it hard to obtain acceptable computation times for traces longer than few thousands of methods even with the fastest architecture, unless a higher number of servers is available. The definition of this problem is tied to the size of the search space, Equations 1 and 2, and the bit-string representation. Indeed, the longer the trace, the higher the number of methods contributing to the segment coupling. However, we believe that if concepts are indeed implemented in cohesive and decoupled segments, then computing coupling with Equation 2 is overly conservative and redefining the problem could substantially reduce computation time. We are currently working to restate the concept location problem in using audio digital signal processing and time windowing.

We have reported data of two traces of one software system, namely JHotDraw, therefore we cannot generalize to other traces though the performance issue is likely to be non-specific to JHotDraw or the used traces. Indeed, we experienced similar results with traces of different lengths of ArgoUML. Much in the same way, we cannot generalize to other search-based software engineering problems. However, we observed that the trade-off between the complexity of the fitness function and the local and global knowledge representations; similar trade-offs are known to be general and common to many application of optimization techniques to software engineering.

VI. CONCLUSION AND FUTURE WORK

GAs have been successfully applied to many complex software engineering problems. To the best of our knowledge, no previous work distributed fitness computation on several servers to exploit the intrinsic parallel nature of GAs to reduce computation times for concept location.

This paper presented and discussed four client–server architectures conceived to improve performance and reduce GA computation times to resolve the concept location problem. To our surprise, we discovered that on a standard

TCP/IP network, the overhead of database accesses, communication, and latency may impair a dedicated solutions. Indeed, in our experiments, the fastest solution was an architecture where each server kept track only of its computations without exchanging data with other servers. This simple architecture reduced GA computation by about 140 times when compared to a simple implementation, in which all GA operations are performed on a single machine.

Future work will follow different directions. First, we are working on reformulating the concept location problem. Also, we want to experiment different communication protocols (*e.g.*, UDP) and synchronization strategies. We will carry out other empirical studies to evaluate the approach on more traces, obtained from different systems, to verify the generality of our findings. Finally, we will reformulate other search-based software engineering problems to exploit parallel computation to verify further our findings.

VII. Acknowledgements

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chair in Software Evolution and in Software Patterns and Patterns of Software) and by G. Antoniol's Individual Discovery Grant.

References

[1] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[2] G. Antoniol and M. D. Penta, "Library miniaturization using static and dynamic information," in *Proceedings of IEEE International Conference on Software Maintenance*. Amsterdam The Netherlands: IEEE Press, Sep 22-26 2003, pp. 235–244.

[3] G. Antoniol, A. Cimitile, A. D. Lucca, and M. D. Penta, "Assessing staffing needs for a software maintenance project through queuing simulation," *IEEE Transactions on Software Engineering*, vol. 30, no. 1, pp. 43–58, Jan 2004.

[4] P. McMinn and M. Holcombe, "Evolutionary testing using an extended chaining approach," *Evol. Comput.*, vol. 14, no. 1, pp. 41–64, 2006.

[5] M. O'Keeffe and M. O. Cinnéide, "Search-based refactoring: an empirical study," *J. Softw. Maint. Evol.*, vol. 20, no. 5, pp. 345–364, 2008.

[6] J. Stender, *Parallel Genetic Algorithm: Theory and Applications*, 1993, vol. 14 Frontiers in Artificial Intelligence and Applications.

[7] W. Rivera, "Scalable parallel genetic algorithms," *Artif. Intell. Rev.*, vol. 16, no. 2, pp. 153–168, 2001.

[8] E. Alba, *Parallel Metaheuristics*. John Wiley and Sons Inc, 2005.

[9] G. Antoniol and Y.-G. Guéhéneuc, "Feature identification: An epidemiological metaphor," *Transactions on Software Engineering (TSE)*, vol. 32, no. 9, pp. 627–641, September 2006, 15 pages.

[10] T. Biggerstaff, B. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in *Proceedings of the International Conference on Software Engineering*, IEEE Computer Society Press, May 1993, pp. 482–498.

[11] V. Kozaczynski, J. Q. Ning, and A. Engberts, "Program concept recognition and transformation," *IEEE Transactions on Software Engineering*, vol. 18, no. 12, pp. 1065–1075, Dec 1992.

[12] Denys Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *Transactions on Software Engineering (TSE)*, vol. 33, no. 6, pp. 420–432, June 2007, 14 pages.

[13] P. Tonella and M. Ceccato, "Aspect mining through the formal concept analysis of execution traces," in *Proceedings of IEEE Working Conference on Reverse Engineering*, 2004, pp. 112–121.

[14] F. Asadi, M. D. Penta, G. Antoniol, and Y.-G. Gueheneuc, "A heuristic-based approach to identify concepts in execution traces," in *European Conference on Software Maintenance and Reengineering*, Madrid (Spain), Mar 15-18 2010, pp. 31–40.

[15] A. Nicolas and L. Timothy, "Extracting concepts from file names; a new file clustering criterion," in *Proceedings of the International Conference on Software Engineering*, April 1998, pp. 84–93.

[16] N. Wilde and M. Scully, "Software reconnaissance: Mapping program features to code," *Journal of Software Maintenance - Research and Practice*, vol. 7, no. 1, pp. 49–62, Jan 1995.

[17] M. Eaddy, A. Aho, G. Antoniol, and Y.-G. Gueheneuc, "Cerberus: Tracing requirements to source code using information retrieval dynamic analysis and program analysis," in *ICPC '08: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*. Washington DC USA: IEEE Computer Society, 2008, pp. 53–62.

[18] W. E. Wong, S. S. Gokhale, and J. R. Horgan, "Quantifying the closeness between program components and features," *Journal of Systems and Software*, vol. 54, no. 2, pp. 87 – 98, 2000, special Issue on Software Maintenance.

[19] D. Edwards, S. Simmons, and N. Wilde, "An approach to feature location in distributed systems," Software Engineering Research Center, Tech. Rep., 2004.

[20] A. D. Eisenberg and K. D. Volder, "Dynamic feature traces: Finding features in unfamiliar code," in *proceedings of the* 21^st *International Conference on Software Maintenance*. IEEE Press, September 2005, pp. 337–346.

[21] N. Gold, M. Harman, Z. Li, and K. Mahdavi, "Allowing overlapping boundaries in source code using a search based approach to concept binding," *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pp. 310–319, 2006.

[22] N. Gold, "Hypothesis-based concept assignment to support software maintenance," *17th IEEE International Conference on Software Maintenance (ICSM'01)*, pp. 545–548, 2001.

[23] L. Dapeng, M. Andrian, P. Denys, and R. Vaclav, "Feature location via information retrieval based filtering of a single scenario execution trace," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York NY USA: ACM, 2007, pp. 234–243.

[24] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, 2008.

[25] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *Proceedings of 22nd IEEE International Conference on Software Maintenance*. Philadelphia Pennsylvania USA: IEEE CS Press, 2006, pp. 469 – 478.

[26] B. S. Mitchell, M. Traverso, and S. Mancoridis, "An architecture for distributing the computation of software clustering algorithms," *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA '01)*, pp. 181–190, 2001.

[27] K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to software module clustering," *Proceedings of the International Conference on Software Maintenance (ICSM '03)*, pp. 315–324, 2003.

[28] S. Yussof, R. A. Razali, and O. H. See, "A parallel genetic algorithm for shortest path routing problem," *Proceedings of the 2009 International Conference on Future Computer and Communication*, pp. 268–273, 2009.

[29] W. Zhi-xin and J. Gang, "A parallel genetic algorithm in multi-objective optimization," *Control and Decision Conference, 2009. CCDC '09. Chinese*, pp. 3497–3501, 2009.

[30] F. Liu, X. Chen, and Z. Huang, "Parallel genetic algorithm for finding roots of complex functional equation," *2nd International Conference on Pervasive Computing and Applications, 2007. ICPCA*, pp. 542–545, 2007.

[31] Y.-W. Chen, Z. Nakao, and X. Fang, "A parallel genetic algorithm based on the island model for image restoration," *Proceedings of the 1996 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing [1996] VI.*, pp. 109 – 118, 1996.

[32] Y. Fukuyama and H.-D. Chiang, "A parallel genetic algorithm for service restoration in electric power distribution systems," *In Proceedings of the 1995 IEEE International Conference on Fuzzy Systems*, pp. 275–282, 1995.

[33] D. L. A. de Araujo, H. S. Lopes, and A. A. Freitas, "A parallel genetic algorithm for rule discovery in large databases," *1999 IEEE International Conference on Systems, Man, and Cybernetics, 1999 (IEEE SMC99)*, vol. 3, pp. 940–945, 1999.

[34] B. Zorman, G. M. Kapfhammer, and R. Roos, "Creation and analysis of a javaspace-based distributed genetic algorithm," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 3, pp. 1107 – 1112, 2002.

[35] Janice Ka-Yee Ng, Y.-G. Guéhéneuc, and G. Antoniol, "Identification of behavioral and creational design motifs through dynamic analysis," *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, December 2009, under publication.

[36] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[37] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[38] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[39] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool." *IEEE Trans. Software Eng.*, vol. 32, no. 3, pp. 193–208, 2006.

[40] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS '67: Proceedings of the spring joint computer conference*. New York, NY, USA: ACM, 1967, pp. 483–485.

[41] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering*, vol. 1. New York: World Scientific, 1993.

Author Index

Afzal, Wasif 79, 133
Antoniol, Giuliano 153
Asadi, Fatemeh 153
Bahsoon, Rami 3
Coutinho, Daniel Pinto 143
de Freitas, Fabrício Gomes 143
de Souza, Jerffeson Teixeira 143
del Águila, Isabel María 67
del Sagrado, José 67
Farzat, Fábio de A. 31
Feldt, Robert 79
Ferrucci, Filomena 89
Gravino, Carmine 89
Gross, Hamilton 101
Guéhéneuc, Yann-Gaël 153
Harman, Mark 47, 101
Lakhotia, Kiran 101
Letko, Zdeněk 36
Lindlar, Felix 111
Lu, Guanzhou 3
Lynch, Kevin 120
Magdaleno, Andréa Magalhães 40
Maia, Camila Loiola 143
Mancoridis, Spiros 120
McMinn, Phil 9
Oliveto, Rocco 89
Orellana, Francisco Javier 67
Palma, Francis 57
Rorres, Chris 120
Sarro, Federica 89
Shevertalov, Maxim 120
Stehle, Edward 120
Susi, Angelo 57
Tonella, Paolo 57
Torkar, Richard 79
Wikstrand, Greger 79
Windisch, Andreas 111
Xiao, Junchao 133
Yao, Xin 3
Yoo, Shin 19
Zhang, Yuanyuan 47

IEEE Computer Society Conference Publications Operations Committee

IEEE Computer Society Publications

The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the CS Store at *http://www.computer.org/portal/site/store/index.jsp* for a list of products.

IEEE Computer Society *Conference Publishing Services* (CPS)

The IEEE Computer Society produces conference publications for more than 250 acclaimed international conferences each year in a variety of formats, including books, CD-ROMs, USB Drives, and on-line publications. For information about the IEEE Computer Society's *Conference Publishing Services* (CPS), please e-mail: cps@computer.org or telephone +1-714-821-8380. Fax +1-714-761-1784. Additional information about *Conference Publishing Services* (CPS) can be accessed from our web site at: *http://www.computer.org/cps*

Revised: 1 March 2009

CPS Online is our innovative online collaborative conference publishing system designed to speed the delivery of price quotations and provide conferences with real-time access to all of a project's publication materials during production, including the final papers. The ***CPS Online*** workspace gives a conference the opportunity to upload files through any Web browser, check status and scheduling on their project, make changes to the Table of Contents and Front Matter, approve editorial changes and proofs, and communicate with their CPS editor through discussion forums, chat tools, commenting tools and e-mail.

The following is the URL link to the ***CPS Online*** Publishing Inquiry Form:
http://www.ieeeconfpublishing.org/cpir/inquiry/cps_inquiry.html