A novel approach to collaborative testing in a crowdsourcing environment

Yuan-Hsin Tung\textsuperscript{a,b,*}, Shian-Shyong Tseng\textsuperscript{a,c,*}

\textsuperscript{a} Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, ROC
\textsuperscript{b} Telecommunication Laboratory, Chunghua Telecom Co., Ltd., Taiwan, ROC
\textsuperscript{c} Department of Applied Informatics and Multimedia, Asia University, Taiwan, ROC

\textbf{A B S T R A C T}

Software testing processes are generally labor-intensive and often involve substantial collaboration among testers, developers, and even users. However, considerable human resource capacity exists on the Internet in social networks, expert communities, or internet forums—referred to as crowds. Effectively using crowd resources to support collaborative testing is an interesting and challenging topic. This paper defines the collaborative testing problem in a crowd environment as an NP-Complete job assignment problem and formulates it as an integer linear programming (ILP) problem. Although package tools can be used to obtain the optimal solution to an ILP problem, computational complexity makes these tools unsuitable for solving large-scale problems. This study uses a greedy approach with four heuristic strategies to solve the problem. This is called the crowdsourcing-based collaborative testing approach. This approach includes two phases, training phase and testing phase. The training phase transforms the original problem into an ILP problem. The testing phase solves the ILP using heuristic strategies. A prototype system, called the Collaborative Testing System (COTS), is also implemented. The experiment results show that the proposed heuristic algorithms produce good quality approximate solutions in an acceptable timeframe.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

As Web applications continue to proliferate, ensuring that they are high quality and reliable is critical (Andrews et al., 2005; Ricca and Tonella, 2001, 2006; Homma et al., 2011). Low reliability software can negatively affect businesses, consumers, and governments as they increasingly depend on the Internet for daily operation. The labor and resource-intensive nature of software testing makes producing reliable software difficult. Many approaches (Bertolino, 2007; Weyers et al., 2011; Souza et al., 2007; Whitehead, 2007; Held and Blochinger, 2009; Abdullah et al., 2009; Shukla and Redmiles, 1996) have recently been proposed to address collaborative testing in the software engineering domain, such as Web application testing, open-source testing, and game beta testing, but they have only focused on workflow design, testing assistance, testing process improvement, and generating bug reports. The Internet is an extensive source of experienced human resources. Howe (Crowdsourcing Wikipedia, 2011; Howe, 2006) proposed the term crowdsourcing—a combination of the words crowd and outsourcing. Crowdsourcing leverages large groups of people or communities on the Internet to solve problems. Amazon Mechanical Turk (Amazon Mechanical Turk, 2011; Amazon Mechanical Turk Wikipedia, 2011) uses this concept to achieve business goals through mass collaboration enabled by Web 2.0 technologies. Crowdsourcing collaborative testing can be used for functional test, user acceptance test, contract acceptance test, user experience test, and beta test. Effectively and efficiently using crowd resources from the Internet to assess software functionalities in limited timeframes and sharing test results with others is important for collaborative testing.

Web applications typically involve complex, multi-tiered, heterogeneous architectures consisting of Web sites, applications, database servers, and clients. Targeted Web application functionalities should be assessed by several test cases for collaborative testing. Therefore, the appropriate assignment of test cases to corresponding testers by accounting for individual and assembled test results should be considered. This study explores how collaborative testing can be effectively conducted in a crowdsourcing environment (Lucca and Fasolino, 2006; Miao et al., 2008; Benedikt et al., 2002; Wang et al., 2008). Collaborative testing requires testers to verify software functions and examine software outputs. The process is inherently cooperative, requiring the coordinated efforts of many testers. However, because testers from a crowdsourcing environment may not be as experienced as professional testers, their experience, abilities, and degrees of involvement should be expressed in terms of various quantitative values. This study uses...
these values to create a trustworthiness indicator. Collaborative testing test case assignment should be based on trustworthiness in a crowdsourcing environment.

This paper defines the collaborative testing problem in a crowdsourcing environment as a job assignment problem and formulates it as an integer linear programming (ILP) problem. Although an ILP optimization package tool can be used to calculate an optimal solution, long execution times make it unsuitable for solving large-scale instances. Therefore, this study uses a greedy approach based on four proposed heuristic strategies, called the crowdsourcing-based collaborative testing approach. The proposed approach includes two phases, training phase and testing phase.

The training phase represents Web application systems, test cases, and testers on five matrices: (1) a test case page coverage matrix (A), (2) a test case execution time matrix (T), (3) a page threshold matrix (TH) (4) a tester trustworthiness matrix (W), and (5) a tester availability matrix (H). The ILP transformer algorithm transforms the matrices into an ILP formulation. In the testing phase, test cases are assigned to participating testers in a collaborative testing environment and testers are guided while performing tests, potentially reducing required testing effort. A prototype system, called the Collaborative Testing System (COTS), was implemented to perform collaborative testing. Three experiments were conducted to evaluate the performance of the approach. The experimental results show that although linear programming tool CPLEX (CPLEX, 2011) can optimally solve the ILP formulation for small-scale problems, the proposed heuristic algorithms always produce good quality approximate solutions within an acceptable time.

This paper is organized as follows: Section 2 discusses related studies; Section 3 describes the collaborative testing scenario and formulates the problem as an ILP; Section 4 explains the crowdsourcing-based collaborative testing approach; Section 5 describes the experimental design and results; and Section 6 provides a conclusion.

2. Related work

2.1. Crowdsourcing and Amazon Mechanical Turk

Crowdsourcing (Collective intelligence Wikipedia, 2011; Crowdsourcing Wikipedia, 2011; Howe, 2006), the act of leveraging mass collaboration to achieve business goals, has become popular on the Internet. Howe (Crowdsourcing Wikipedia, 2011; Howe, 2006) first used the term crowdsourcing—a combination of crowd and outsourcing—to describe the practice of solving complex problems and contributing relevant and novel ideas through an open request on the Internet. Amazon Mechanical Turk (Amazon Mechanical Turk, 2011; Amazon Mechanical Turk Wikipedia, 2011) is a crowdsourcing Web service that enables computer programmers (called Requesters) to co-ordinate human intelligence to perform tasks which computers cannot complete. Requesters set human intelligence tasks (HITs) and Workers (or Providers) can browse, select, and complete tasks for a monetary payment set by a Requester. A Requester can set Worker qualification requirements for a task and they can create tests to verify these qualifications. They can also accept or reject a task completed by a Worker, which affects the Worker's reputation. Workers can be located anywhere in the world. TopCoder (TopCoder, 2011) proposed another crowdsourcing business model to administer software programming contests. Clients pay TopCoder to design software applications based on their requirements. TopCoder then creates a contest for community members to develop software and uses this contest-based system with a rating mechanism to evaluate programmer performance relative to other participants. For example, if a new coder beats an established coder, the new coder is ranked higher. This paper uses the variable trustworthiness to reflect tester performance in a crowdsourcing test environment by referring to crowdsourcing Web sites, such as Amazon Mechanical Turk and TopCoder.

Crowdsourcing can create an interesting dynamic when testing in a cloud environment (Crowdsourcing Wikipedia, 2011; Howe, 2006; Buyya et al., 2009). Thus, vendors of cloud testing services must find methods of constantly encouraging testers to compete effectively. Riengu et al. (2010) investigated how crowdsourcing supports cloud testing and different crowdsourcing models that could be practical for cloud testing. Testing service provider uTest (Utest, 2011) proposed the crowdsourcing Test-as-a-Service model for providing testing services to customers using several heterogeneous, resourceful, and skilled testers. Another crowdsourcing model uses a community of users or an interest group to test specific software, for example Amazon Mechanical Turk, TopCoder, and AppStori (AppStori, 2012). Feedback from the community is then used to improve the tested application. Incorporating crowdsourcing and collaborative testing in the cloud raises social issues such as trust and communication. Hence, research should investigate the role and impact of trust in collaborative testing activities in the cloud.

2.2. Collaborative testing

In software engineering, software testing often involves substantial collaboration among testers, developers, and users. A wide range of communication and collaboration technologies are used to coordinate project work. Many studies (West et al., 2011; Shahriar and Zulkernine, 2011; Held and Blochinger, 2009) have focused on collaborative testing. However, most proposed collaborative testing tools only focus on testing processes and generating bug reports. No tools support collaborative testing in a crowdsourcing environment and account for practical communication requirements and testing resource constraints. In Tsai et al. (2004) and Bai et al. (2007a,b), collaborative verification and validation (CVV) of an application requires contributions from many parties in a collaborative manner. The CVV framework publishes and ranks test cases based on their potential. The most potent test cases are first used for testing new software to reduce testing efforts. Whitehead (Whitehead, 2007) classified software engineering collaboration tools into four categories: model-based collaboration tools, processes support tools, awareness tools, and collaboration infrastructure. The main collaborative tool used to manage the interface between testers and developers is the bug tracking tool (Shukla and Redmiles, 1996) which assists with generating and storing or recording an initial error report, prioritization, adding follow-on comments and error data, linking similar reports, and assigning a developer to repair the software. Once a bug is fixed, it is recorded in the bug tracking system. Software inspection involves many engineers reviewing a specific software artifact. Therefore, software inspection tools (Macdonald and Miller, 1999) are historically collaborative. Based on tool features, Hedberg (Hedberg, 2004) divided software inspection history into four phases: early tools, distributed tools, asynchronous tools, and Web-based tools. Early tools were designed to support engineers in face-to-face meetings, while distributed tools allowed engineers to participate remotely in inspection meetings. Asynchronous tools meant that inspection participants did not have to meet simultaneously and Web-based tools supported inspection processes on the Web.

3. Collaborative testing problem definition

This section introduces the proposed approach to support collaborative testing for Web applications. To simplify discussion of
the collaborative testing formulation, this study assumes that the
dependence graph, test cases, and tester profiles were designed
by test engineers and the tester profiles contain tester availability,
exection time for each test case, and collaborative testing trust-
worthiness. These assumptions are based on Web user behavior
evaluations.

Collaborative testing was defined, with assumptions, as a job
assignment problem using the motivation example. The problem
was then modeled as an ILP formulation with resource constraints.
The job assignment problem was reduced to an NP-Complete
problem (Garey and Johnson, 1979). The proposed crowdsourcing-
based approach for collaborative testing was then introduced.

3.1. Motivation example

To conveniently collect test cases for Web application testing,
test engineers produced test cases using a user-session based
approach (Liu et al., 2000; Benedikt et al., 2002; Elbaum et al.,
2005; Deng et al., 2004), where each test case was a set of execu-
tion paths and input attribute-value pairs. The Web application
can be represented as a dependence graph. The notations used for
the dependence graph and test case are as follows:

*Fig. 1. An excerpt of dependence graph for the Web application, BookStore.*

\[ T_{C_1} = \langle \text{Default.aspx}, \ (\text{category_id}=2, \ \text{title}='\text{Databases}') \rangle, \ (\text{AdvSearch.aspx}, \ (\text{title}='\text{Databases}', \ \text{author}='\text{Jim Buyens}', \ \text{category_id}=2, \ \text{price_min}=15.99, \ \text{price_max}=39.99)' \rangle, \ (\text{Books.aspx}, \ (\text{category_id}=2, \ \text{title}='\text{Databases}') \rangle, \ (\text{BookDetail.aspx}, \ (\text{item_id}=1, \ \text{category_id}=2, \ \text{quantity}=3, \ \text{rating}=48, \ \text{rating_count}=14') \rangle \]

*Fig. 3. A collaborative testing example in a crowdsourcing
environment. The Web application with four Web pages and
four test cases was extracted from BookStore (Open Source
Web Applications, 2011) and three online testers participated in the
collaborative test without interacting. With no coordination, testers
usually execute specific popular application functions. Fig. 3 uses
the threshold and trustworthiness variables to describe specific
situations in a crowdsourcing test. The threshold variable is used as
a testing criterion for testing a target based on page complexity.
A higher threshold means that a testing target is more complex and
requires more testing effort. The trustworthiness variable reflects
a tester’s experiences, abilities, and degrees of involvement by
using a crowdsourcing Web site rating mechanism (AppStori, 2012;
TopCoder, 2011). Higher trustworthiness reflects more reliable
testing results. In this scenario, the collaborative testing problem
is transformed into a job assignment problem with resource con-
straints.*

The notations used for the collaborative testing formulation are as follows:

\[
\begin{align*}
\text{G} & \quad \text{dependence graph; a dependence graph for a Web application is a} \\
\text{P} & \quad \text{set of nodes in the dependence graph;} \\
\text{E} & \quad \text{set of edges in the dependence graph;} \\
\text{p_i} & \quad \text{Web page, p_i \in P;} \\
\text{e_{ij}} & \quad \text{a direct link from p_i to p_j, where e_{ij} = (p_i, p_j), e_{ij} \in E, and } V p_i, p_j \in P \\
\text{M_i^{\text{attr}}} & \quad \text{input attribute-value pairs; } \\
\text{M_i^{\text{attr}}} & = \langle (\text{attr}^{\text{attr}}_1, \ \text{val}^{\text{attr}}_1), (\text{attr}^{\text{attr}}_2, \ \text{val}^{\text{attr}}_2) \ldots \rangle, \text{where attr}^{\text{attr}}_i(x) \text{ is the} \\
\text{input attribute and val}^{\text{attr}}_i(x) \text{ is the input attribute value; } \\
\text{TC_j} & \quad \text{test case TC_j is a test path with input attribute-value pairs} \\
\text{TC_j} & = \langle (p_{j1}, \ M_1^{\text{attr}}), (p_{j2}, \ M_2^{\text{attr}}), \ldots, p_{jk}, M_k^{\text{attr}} \ldots \rangle, \forall p_j \in P. \\
\end{align*}
\]
The five matrices of collaborative testing problem were then formulated in the training phase. Table 1 shows a test case page coverage matrix of the relationships between test cases and pages. For example, test case 1 (TC1) in Fig. 3 covers three pages, P1, P2, and P3, and is written as \( (P1 \rightarrow P2 \rightarrow P3) \). Table 1 shows the test case page coverage matrix based on the set of test cases. Three testers participated in the test and the test case execution time for each tester was estimated by adding the execution time of the corresponding pages. Tester A executed TC1 in 8 min. Table 2 shows the test case execution time matrix which summarizes the tester execution times.

Two factors that may affect collaborative testing were also considered: page complexity and tester trustworthiness. Because different pages may have different complexities, the threshold was defined as the testing criterion by referring to page complexity. In the page threshold matrix in Table 3, the test threshold was defined with lines of code which are complexity indicators (Albrecht and Gaffney, 1983; Low and Jeffery, 1990). More complex pages require more testing. Because inexperienced and even malicious testers are likely to participate in testing, users must decide whether to trust the testing results. To make this decision, a tester trustworthiness matrix was created based on tester prior participation, as shown in Table 4. In Table 5, the tester availability matrix shows the time that testers can dedicate to testing.

This study aims to find the minimal-time test case combination that covers the whole Web application and achieves the testing criteria. When testing began, Testers A, B, and C executed pages individually. To complete testing, testers must be coordinated to conduct the test cases and work quickly. The test case executions for the case solutions are as follows:

- TC1: Tester A executed the case in 8 min.
- TC2: Tester B executed the case in 18 min.
- TC3: To achieve collaborative testing with minimal total testing time, test cases must be appropriately assigned to testers. The Web application testing structures were described by constructing a dependence graph. Page coverage was calculated during testing using the dependence graph. Crowd testers were then guided by test case assignment and they then performed test cases on their own, that is, Tester A: \( P1 \rightarrow P2 \rightarrow P3 \rightarrow P4 \), Tester B: \( P1 \rightarrow P2 \rightarrow P3 \), or Tester C: \( P1 \rightarrow P4 \).

<table>
<thead>
<tr>
<th>TC1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tester</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>8</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>TC2</td>
<td>24</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>TC3</td>
<td>12</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>TC4</td>
<td>16</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tester</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>48</td>
<td>24</td>
<td>48</td>
</tr>
</tbody>
</table>
This is how test cases were assigned to testers and how testers were guided to the required regions to achieve testing and minimize total testing time. This is a job assignment problem with resource constraints.

3.2. Problem definition: collaborative testing job assignment

This section defines the proposed job assignment problem using a mathematical formula. Based on the matrices, the job assignment problem was formulated as an ILP problem. The job assignment problem was used to calculate a sub-optimal representative set (Garey and Johnson, 1979). Calculating the optimal representative set is a minimal representative set problem, or an NP-Complete problem. The ILP formulation was developed as follows:

Objective function [OBJ] is

\[
\min \sum_{i} \sum_{j} x_{ij} t_{ij},
\]

where \(x_{ij}\) is binary for test case \(i = 1, \ldots, n\), and tester \(j = 1, \ldots, m\). Subject to [CONSTRAINTS]:

1. \(\sum_{i} a_{ik} x_{ij} \geq 1\) for each page. [CC1: Coverage Constraint]
2. \(\sum_{k} a_{ik} \geq 1\) for each test case. [CC2: Coverage Constraint 2]
3. \(0 \leq \sum_{j} x_{ij} t_{ij} \leq h_{j}\) for each tester. [TAC: Tester Available Constraint]
   where \(t_{ij} > 0\) and \(h_{j} > 0\).
4. \(\sum_{j} a_{ik} x_{ij} w_{j} \leq th_{k}\) for each tested page. [TC: Trustworthiness Constraint]
   where \(0 < w_{j} \leq 1\) and \(th_{k} > 0\).
5. \(x_{ij}\) is binary. [BC: Binary Constraint]

According to this ILP formulation, the objective function is used to calculate the total collaborative testing execution time. To calculate the optimal execution time, the objective function minimizes the total execution time with variables \(x_{ij}\) and \(t_{ij}\). If binary variable \(x_{ij} = 1\), the \(i\)th test case is assigned to the \(j\)th tester, otherwise \(x_{ij} = 0\). Variable \(t_{ij}\) reflects the testing time of the \(j\)th tester for the \(i\)th test case. To model the collaborative testing characteristics, the coverage, availability, and trustworthiness constraints were constructed as Constraints 1–4. Table 1 shows that \(a_{ik} = 1\) if the \(i\)th test case covers the \(k\)th page. Constraint 1 ensures that all Web pages are covered by test cases and Constraint 2 ensures that each test case covers at least one page. Coverage measurements are important for software testing. Constraint 3 ensures that tester resources are limited to prevent work overloading by referring to Table 2, \(T = \{t_{ij}\}\), and Table 5, \(H = \{h_{j}\}\). In Table 2, variable \(t_{ij}\) is the execution time of the \(i\)th test case for the \(j\)th tester. In Table 5, \(h_{j}\) represents tester work time. Constraint 4 models trustworthiness and testing support by considering tester trustworthiness from Table 4, \(W = \{w_{j}\}\). Each tester gains a trustworthy weighting when he or she executes testing work. Threshold value \(th_{k}\) in Table 3 is a Web page testing criterion and \(w_{j}\) represents the trustworthiness of the \(j\)th tester. A more complex page with higher threshold \(th_{k}\) requires more testing. The summary of \(a_{ik} x_{ij} w_{j}\) means that the assigned test cases must cover testing threshold \(th_{k}\) of each page. If covered, the page is marked as tested, if not, the page is assigned to another tester. In Constraint 5, variable \(x_{ij}\) can only equal 0 or 1.

4. Crowdsourcing-based approach for collaborative testing

4.1. Overview

Fig. 4 shows that the proposed crowdsourcing-based approach consists of two main phases. In the training phase, the ILP transformer was constructed to transform the collaborative testing problem into an ILP formulation. The required settings include a dependence graph, test cases, and tester profiles. A leverage dependence graph (Jeffrey and Gupta, 2005; Leon et al., 2005) was used to represent the Web applications. Test case bases were converted into a coverage matrix to represent the relationship between Web applications and test cases. A tester profile was created for each tester based upon his or her participation. In the testing phase, participating testers conducted tests. The test case assignment algorithm was designed based on the greedy approach for assigning appropriate test cases to individual testers. This approach continues to assign test cases to testers until all pages are finished.

4.2. Training phase

In the training phase, five matrices based on the dependence graph, test case base, and tester profiles were used to represent the job assignment problem. Algorithm 1, the ILP transformer algorithm, was proposed to convert the original problem into an ILP
problem according to the problem definition in Section 3.2. Algorithm 1 consists of three steps. In Step 1, the objective function calculates the minimal time required to execute the test cases using the available testers. If variable \( x_j = 1 \) in the objective function, this means that the \( j \)th test case is assigned to the \( j \)th tester. The number of variables in the objective function is \( \sum_j \). In Step 2, Coverage Constraint 1 (CC1) and Coverage Constraint 2 (CC2) were constructed by considering the test case page coverage matrix \( A \). The Tester Available Constraint (TAC) and Trustworthiness Constraint (TC) were constructed by considering the test case execution time matrix \( T \) and tester availability matrix \( H \). The binary constraint for variable \( x_j \) is the Binary Constraint (BC). There are: \( \sum_i \) constraints for CC1 and CC2, \( j \) constraints for TAC, \( k \) constraints for TC, and \( i \cdot j \) constraints for BC. The proposed model consists of \((i \cdot j + i + 2k)\) constraints. Step 3 returns the ILP formulation.

**Algorithm 1. ILP transformer algorithm**

**Input:** Test case page coverage matrix, \( A = [a_{ik}] \), test case execution time matrix, \( T = [t_{ij}] \), page threshold matrix, \( TH = [th_k] \), tester trustworthiness matrix, \( W = [w_k] \), tester availability matrix, \( H = [h_k] \).

**Output:** Collaborative testing in the ILP formulation, \( ILP = \{OBJ, CONSTRAINTS\} \).

**Step 1.** According to matrices \( A \) and \( T \), let objective function \( OBJ = \sum_i \sum_j x_{ij} t_{ij} \) to minimize the test case combination execution time.

**Step 2.** Generate constraints.

**Step 2.1.** Referring to test case page coverage matrix, \( A \), add CC1 and CC2 to \( CONSTRAINTS \).

**Step 2.2.** Referring to test case execution time matrix, \( T \), and tester availability matrix, \( H \), add TAC to \( CONSTRAINTS \).

**Step 2.3.** Referring to matrices, \( A, W, \) and \( S \), add TC to \( CONSTRAINTS \).

**Step 2.4.** Add BC to \( CONSTRAINTS \) for variable \( x_j \).

**Step 3.** ILP formulation, \( ILP = \{OBJ, CONSTRAINTS\} \).

### 4.3. Testing phase

The test case assignment algorithm was proposed in the testing phase and linear programming tool ILOG CPLEX was used to evaluate algorithm performance. The algorithm calculates the representative set of test cases with the minimal execution time and satisfies the objective function and the constraints. The algorithm identifies one candidate test case in each iteration. As shown in Algorithm 2, the proposed algorithm consists of three steps.

**Algorithm 2. Test case assignment algorithm**

**Input:** Collaborative testing in the ILP formulation, \( ILP = \{OBJ, CONSTRAINTS\} \); proposed heuristic strategy, \( H \).

**Output:** Test case assignments, \( x_{ij} \); objective function, \( OF \).

**Initialize:** Set all variable \( x_{ij} = 0 \), \( OF = 0 \).

**Step 1.** Calculate page support \( u_k \) for each page \( k \).

**Step 2.** Identify target page \( k \) with the maximal page gain.

**Step 2.1.** Calculate page gain \( Eval_k \) for each page \( k \) with page support \( u_k \) and trustworthiness \( th_k \).

**Step 2.2.** Identify target page \( k \) with maximal page gain \( Eval_k \).

**Step 3.** Identify test case \( r \) for target page \( k \) based on \( TCS(H) \).

**Step 3.1.** Use proposed heuristic strategy \( H \) from \( H1 \) to \( H4 \) for test case selection.

**Step 3.2.** Select test case \( r \) from candidate test case set \( T \). Calculate \( TCS(H) \) for candidate test case \( r \) and identify test case \( r \) with maximum \( TCS(H) \).

**Step 3.3.** Assign test case \( r \) to tester \( j \), set \( x_{ij} = 1 \), and update OF based on OBJ.

**Step 3.4.** If all page supports, \( u_k \), are bigger than threshold \( th_k \), EXIT; otherwise, GOTO Step 1.

In Step 1, the page support was used to monitor all test cases performed for each page. In Eq. (1), the page support was calculated by referring to the test case page coverage matrix and tester trustworthiness.

**Page support vector:** \( U = [u_k] \) page support for \( k \)th Web page.

**Page support:** \( u_k = \sum_j a_{ij} x_{ij} w_j \). (1)

In Step 2, page gain, \( Eval_k \), was calculated with page support \( u_k \) and trustworthiness \( th_k \) for each node in Eq. (2). The node with maximal \( Eval_k \) is the target page for testing. Once the target page is selected, the candidate test case set is produced, where each test case contains the target page.

**Page gain:** \( Eval_k = \frac{(th_k - u_k)}{th_k} \). (2)

Where \( th_k \) is the threshold of the \( k \)th page and \( u_k \) is the page support of the \( k \)th page.

In Step 3 the most essential test case was selected from the set of candidate test cases. Four heuristic strategies can be selected in the proposed algorithm. The greedy algorithm is based on four heuristic strategies for test case selection. \( TCS(H) \), defined in Eqs. (3.1)–(3.4). Eq. (3.1), \( TCS(H1) \), selects the test case with maximal coverage and Eq. (3.2), \( TCS(H2) \), selects the test case with the minimal execution time. Eq. (3.3), \( TCS(H3) \), selects the test case with maximal tester trustworthiness and Eq. (3.4), \( TCS(H4) \), is a compound heuristic, consisting of minimal execution time and maximal coverage.

**Heuristic 1 (H1):** maximal coverage heuristic \( TCS(H1) = \max \sum_k n_{ik} \). (3.1)

Where \( n_{ik} = 1 \) means that the \( i \)th test case contains the \( k \)th page.

**Heuristic 2 (H2):** maximal time heuristic \( TCS(H1) = \min t_{ij} \) (3.2)

Where \( t_{ij} \) refers to the execution time of the \( i \)th test case by the \( j \)th tester.

**Heuristic 3 (H3):** maximal-trustworthiness heuristic \( TCS(H3) = \max w_j \), (3.3)

Where \( w_j \) represents the trustworthiness of tester \( j \).

**Heuristic 4 (H4):** compound heuristic \( TCS(H4) = \min \left( \frac{T_{ij}}{\max \sum_k a_{ik}} \right) \) (3.4)

The selected test case is used to update the current page support. Testing is complete once the testing criterion is met, otherwise proceed to step 1.

The crowdsourcing-based collaborative testing approach contains two algorithms. Algorithm 1 is an ILP transformer with linear time complexity to transform the proposed matrices into an ILP formulation. Therefore, the time complexity of Algorithm 2, which dominates the complexity of the computation of the proposed approach, must be analyzed.

In Step 1, all \( a_{ik} x_{ij} w_j \) are summarized as page support \( u_k \) for each page. Page support time complexity is calculated by multiplying the number of test cases \( m \) by the number of pages \( n \). In Step 2, the page gain calculation, \( Eval_k \), requires subtraction and division. In Step 2.1, the time complexity of all page gain calculations is \( O(n) \). Step 2.2 uses time complexity \( O(n) \) to identify the maximal page gain. The total time complexity of Step 2 is \( 2O(n) \).

In Step 3, the greedy algorithm with four heuristic strategies is based on a selection sort algorithm. Therefore, the average time complexity of the selection sort algorithm is \( O(n \log(n)) \), and the time complexity of the worst case is no worse than \( O(n^2) \). In Eq. (3.4), the compound heuristic is a min-max function and time complexity is \( 2O(n^2) \). Combining Steps 1–3, Algorithm 2 is represented by

\[ T = O(m \times n) + 2 \times O(n) + \max(2O(n^2), 2 \times O(n^2)) = O(n^2). \]

**Example:**

Table 6 shows an example illustrating the collaborative testing approach processes. The top of the table shows page thresholds...
and the bottom shows the proposed algorithm processes. According to the proposed algorithm, in Step 1 the page support vector, \( U(r) = [u_k] \), is calculated for each page using Eq. (1). Test case 1 is assigned to Tester A in iteration \( r = 1 \) and the page support with vector \( U \) is updated from \( U(0) = [0, 0, 0, 0] \) to \( U(1) = [0, 0.7, 0.7, 0.7] \). In Step 2, the target page is selected by calculating the page gain, \( Eval_k \), for each page. As shown in Eq. (2), page support \( u_k \) represents the testing situation of the \( k \)th page. The maximal \( Eval_k \) value means that the \( k \)th page is the target page in the Web application. The \( Eval_k \) calculation selected Page 3, with a maximal value of 0.557. In Step 3, a test case is selected based on the algorithm test case selection strategy, TCS. In this case, the collaborative testing algorithm was performed with test case selection strategy H2 (the minimal time heuristic), as shown in Eq. (3.2). The test case containing Page 3 was given to testers and testing was performed iteratively until page \( Eval_k \) was \( < 0 \). The proposed test case was assigned to testers and testing was performed iteratively until testing of all Web application pages was complete. The proposed algorithm identifies candidate test cases in each iteration. The test case combination is suggested by the sequence, TC3, TC4, TC1, TC2, TC1, TC3.

5. Implementation and experiment

This study implements the prototype COTS to support collaborative testing using the proposed crowdsourcing-based approach. To evaluate the performance of the proposed approach, a series of computational simulations were designed and conducted. The experiments consisted of three parts: (1) converting the collaborative testing problem to the ILP formulation; (2) comparing the heuristic algorithm and the ILP formulation; and (3) investigating collaborative testing using the proposed approach and the COTS.

5.1. System implementation

The COTS is a Web-based application that was developed using ASP.NET and MS SQL Server. This study applies the proposed approach to support test case assignment of collaborative testing using the COTS. The COTS also collects and analyzes bug reports for collaborative testing. Fig. 5 shows that the system contains five main modules: the Testing User Interface, Bug Report System, Test Case Assignment, Web Application Transformer, and System Admin modules. As shown in Fig. 6, the Testing User Interface and Bug Report System modules provide interactive interfaces to testers. The Bug Report System module allows testers to report any bugs or glitches. The COTS records tester-reported bugs and stores the information in the Bug Report Database. The Test Case Assignment module uses the proposed approach to provide testing objectives to testers. Fig. 7 shows that the System Admin module provides various administrative functions for system administrators to monitor testing. To evaluate the proposed algorithm performance, test engineers performed collaborative testing on the COTS in various experiments.

5.2. Experiments and results

5.2.1. Experiment 1: converting the collaborative testing problem to the ILP formulation

Experiment 1 determines the minimal-time test case combination using the ILP formulation. The ILP formulation was used to identify optimal collaborative testing solutions. It was implemented using the linear programming tool, ILOG CPLEX Interactive Optimizer 11.2.1, on a Linux server with Intel Core

![Fig. 5. System architecture of COTS.](image1.png)

![Fig. 6. Testing user interface of COTS.](image2.png)
2 extreme 2.66 GHz CPUs and 4 GB RAM. The Web application, BookStore (Open Source Web Applications, 2011), from Web site www.gotocode.com was used. The application was developed in the ASP.NET language. To simulate a real testing environment, several combinations of test cases and testers were conducted for collaborative testing. Testing engineers selected the test cases according to the features of Web application BookStore. As shown in Table 1, t represents test cases and j represents testers. The parameters for tester profiles, user trustworthiness weights, participating testers, and testing thresholds were configured based on the preliminary settings in Section 3.

The experiment results suggest that the execution times increase sharply as the number of variables \((i'j')\) increases. For example, when variable \((i'j')\) is 1800, the average execution time is more than 2651 s. And when variable \((i'j')\) is 2145, the average execution time is 2851 s. To verify the correlation between number of variables and execution time, the R-squared value (coefficient of determination, \(R^2\)) (Pearson, 1987) for the two indicators was calculated. When the coefficient is greater than 0.75, the correlation between the two indicators is regarded as high and one of the indicators is eliminated based on the next criterion. An R-squared value of 0.8035 reflects a significant positive correlation between the number of variables and execution time. When the number of variables increases, the execution time increases. In this case, the CPLEX failed to render results when the number of variables was more than 2800. Table 7.

5.2.2. Experiment 2: comparisons between the heuristic algorithm and ILP formulation

Experiment 2 compares the performance of the proposed algorithm with that of the CPLEX ILP formulation. Collaborative testing experiments designed by experienced test engineers were conducted on the target Web application, BookStore. Tables 8–10 show the experiment results with test case and tester configurations.

The solutions to the corresponding ILP formulation were used as a baseline for comparisons. In Tables 8–10, the first three columns show the configuration of test cases, testers, and number of variables. The ILP column shows the two execution results, execution time and optimal objective. The four heuristic algorithm sub-columns summarize the computational objectives of the proposed algorithm and heuristic strategies.

In Table 8, the number of test cases increases with the number of testers. In Table 9, the number of testers is constant and the number of test cases increases. In Table 10, the number of test cases is set at 70 and the number of testers increases in increments of 5.

The following observations are based on the numerical results.

(1) Tables 8–10 show that the ILP formulation execution time increases rapidly as the number of variables increase, demonstrating that the problem quickly becomes more complex when the number of variables increases. When the number of variables is large, the corresponding execution time becomes prohibitively long. However, the proposed approach is almost unaffected by increasing problem complexity. The proposed algorithm achieved averages of approximately 90% (ILP over Heuristic H2 in Table 8), 79% (ILP over Heuristic H2 in Table 9), and 89.5% (ILP over Heuristic H2 in Table 10) of the optimal solution. This shows that the proposed approach is suitable for collaborative testing in a real-time environment, without requiring complex computations.

(2) Because the proposed approach uses a greedy algorithm, this ensures that a feasible solution is obtained within an acceptable time. This is reflected by the experiment results. However, because the collaborative testing problem is NP-Complete, the CPLEX ILP formulation execution time increases significantly as the number of variables increases. In Table 10, when the number of variables exceeds 2800, the solution cannot be found in 7200 s.

(3) The experiment results show that the objective decreases as the number of test cases and testers increases. In Tables 8–10, the objective optimal column suggests that when there are few test cases and testers, the number of solution combination choices is relatively small. This results in higher cost solutions.

(4) The experiments propose four heuristic strategies to solve the ILP formulation. Heuristics 2 and 4 perform better than the other two heuristics. The average objectives of Heuristic 2 are 2510.71, 2854.44, and 2335, and the average objectives of Heuristic 4 are 2555, 2898.89, and 2375.83. The two heuristics include costs in the equations. Heuristics 1 and 3 may perform better in the long term when learning curves are considered, because tester capability increases with experience.

5.2.3. Experiment 3: collaborative testing with the proposed approach in the COTS

Experiment 3 evaluates the performance of the proposed approach and the prototype COTS. Crowdsourcing Web sites TopCoder and Amazon Mechanical Turk were consulted when designing the crowdsourcing test experiment. Collaborative testing was performed using Web application Bookstore and Internet crowd testers. Experienced test engineers embedded 10 bugs into the Web application. Crowd testers were then requested to participate in the test. Participating testers were divided into two groups, a guided group and a random group, based on page supports. A new tester registered by completing a questionnaire to estimate their trustworthiness. In Experiment 3, the COTS assigned test cases to testers according to heuristic H1 criteria in the guided group (minimum cost first strategy) and randomly assigned test cases to testers in the random group. The experiment ended when all page supports in BookStore reached their thresholds.

Over a 6-day testing period, 144 testers participated in the experiment. The COTS dispatched incoming crowd testers to the random or guided group, until each page support in the two groups exceeded its threshold. The guided group test ended 3 days before the random group. Fig. 8 shows that the guided group required less testing time than the random group. Total execution time comparisons between the two groups show that the testing effort reduction rate is almost 53%. This indicates that the approach improves testing effectiveness by guiding testers with test case assignment. Table 11 shows the bugs reported by participating testers. The groups reported 55 and 97 bugs and both groups of testers discovered all 10 defects. This shows that although both groups exhibit
Table 7
Execution results of ILP for different test case and tester combination in experiment 1.

<table>
<thead>
<tr>
<th>Test case (i)</th>
<th>Tester (j)</th>
<th>Variable (i × j)</th>
<th>Execution time (s)</th>
<th>Objective optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex1-1</td>
<td>35</td>
<td>15</td>
<td>525</td>
<td>282.75</td>
</tr>
<tr>
<td>ex1-2</td>
<td>40</td>
<td>18</td>
<td>720</td>
<td>530.79</td>
</tr>
<tr>
<td>ex1-3</td>
<td>45</td>
<td>21</td>
<td>945</td>
<td>752.28</td>
</tr>
<tr>
<td>ex1-4</td>
<td>50</td>
<td>24</td>
<td>1200</td>
<td>707.25</td>
</tr>
<tr>
<td>ex1-5</td>
<td>55</td>
<td>27</td>
<td>1485</td>
<td>2991.73</td>
</tr>
<tr>
<td>ex1-6</td>
<td>60</td>
<td>30</td>
<td>1800</td>
<td>2651.95</td>
</tr>
<tr>
<td>ex1-7</td>
<td>65</td>
<td>33</td>
<td>2145</td>
<td>2851.46</td>
</tr>
<tr>
<td>Average</td>
<td>50</td>
<td>24</td>
<td>1260</td>
<td>1538.32</td>
</tr>
</tbody>
</table>

R-squared (number of variables, execution time) = 0.8035.

Table 8
Execution results of different test case and tester combination in experiment 2.

<table>
<thead>
<tr>
<th>Test Case (i)</th>
<th>Tester (j)</th>
<th>Variable (i × j)</th>
<th>ILP</th>
<th>Heuristic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Execution Time (s.)</td>
</tr>
<tr>
<td>ex1-1</td>
<td>35</td>
<td>15</td>
<td>525</td>
<td>282.75</td>
</tr>
<tr>
<td>ex1-2</td>
<td>40</td>
<td>18</td>
<td>720</td>
<td>530.79</td>
</tr>
<tr>
<td>ex1-3</td>
<td>45</td>
<td>21</td>
<td>945</td>
<td>752.28</td>
</tr>
<tr>
<td>ex1-4</td>
<td>50</td>
<td>24</td>
<td>1200</td>
<td>707.25</td>
</tr>
<tr>
<td>ex1-5</td>
<td>55</td>
<td>27</td>
<td>1485</td>
<td>2991.73</td>
</tr>
<tr>
<td>ex1-6</td>
<td>60</td>
<td>30</td>
<td>1800</td>
<td>2651.95</td>
</tr>
<tr>
<td>ex1-7</td>
<td>65</td>
<td>33</td>
<td>2145</td>
<td>2851.46</td>
</tr>
<tr>
<td>Average</td>
<td>50</td>
<td>24</td>
<td>1260</td>
<td>1538.32</td>
</tr>
</tbody>
</table>

Table 9
Execution results of different numbers of test cases in experiment 2.

<table>
<thead>
<tr>
<th>Test Case (i)</th>
<th>Tester (j)</th>
<th>Variable (i × j)</th>
<th>ILP</th>
<th>Heuristic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Execution Time (s.)</td>
</tr>
<tr>
<td>ex2-1</td>
<td>30</td>
<td>20</td>
<td>600</td>
<td>2.46</td>
</tr>
<tr>
<td>ex2-2</td>
<td>35</td>
<td>20</td>
<td>700</td>
<td>5.48</td>
</tr>
<tr>
<td>ex2-3</td>
<td>40</td>
<td>20</td>
<td>800</td>
<td>23.53</td>
</tr>
<tr>
<td>ex2-4</td>
<td>45</td>
<td>20</td>
<td>900</td>
<td>40.99</td>
</tr>
<tr>
<td>ex2-5</td>
<td>50</td>
<td>20</td>
<td>1000</td>
<td>19.33</td>
</tr>
<tr>
<td>ex2-6</td>
<td>55</td>
<td>20</td>
<td>1100</td>
<td>80.11</td>
</tr>
<tr>
<td>ex2-7</td>
<td>60</td>
<td>20</td>
<td>1200</td>
<td>84.37</td>
</tr>
<tr>
<td>ex2-8</td>
<td>65</td>
<td>20</td>
<td>1300</td>
<td>1001.86</td>
</tr>
<tr>
<td>ex2-9</td>
<td>70</td>
<td>20</td>
<td>1400</td>
<td>2599.27</td>
</tr>
<tr>
<td>Avg.</td>
<td>50</td>
<td>20</td>
<td>1000</td>
<td>428.6</td>
</tr>
</tbody>
</table>

Table 10
Execution results of different numbers of testers in experiment 2.

<table>
<thead>
<tr>
<th>Test Case (i)</th>
<th>Tester (j)</th>
<th>Variable (i × j)</th>
<th>ILP</th>
<th>Heuristic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Execution time (s.)</td>
</tr>
<tr>
<td>ex3-1</td>
<td>70</td>
<td>15</td>
<td>1050</td>
<td>402</td>
</tr>
<tr>
<td>ex3-2</td>
<td>70</td>
<td>20</td>
<td>1400</td>
<td>682</td>
</tr>
<tr>
<td>ex3-3</td>
<td>70</td>
<td>25</td>
<td>1750</td>
<td>970</td>
</tr>
<tr>
<td>ex3-4</td>
<td>70</td>
<td>30</td>
<td>2100</td>
<td>1837</td>
</tr>
<tr>
<td>ex3-5</td>
<td>70</td>
<td>35</td>
<td>2450</td>
<td>2671</td>
</tr>
<tr>
<td>ex3-6</td>
<td>70</td>
<td>40</td>
<td>2800</td>
<td>7200^a</td>
</tr>
<tr>
<td>Avg.</td>
<td>70</td>
<td>27.5</td>
<td>1925</td>
<td>2293.67</td>
</tr>
</tbody>
</table>

^a In this experimental, we set the testing criteria as 7200 s.
^b Current best solution under time limit.

Table 11
Statistics of bug reports in experiment 3.

<table>
<thead>
<tr>
<th>Group</th>
<th>Crowd tester</th>
<th>Execution times (min)</th>
<th>Avg. execution time (min)</th>
<th>Numbers of bug reports</th>
<th>Defect coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Group</td>
<td>85</td>
<td>16,740</td>
<td>196.94</td>
<td>97</td>
<td>100%</td>
</tr>
<tr>
<td>Guided Group</td>
<td>59</td>
<td>8983</td>
<td>152.25</td>
<td>55</td>
<td>100%</td>
</tr>
</tbody>
</table>
Fig. 8. Total execution time of guide group and random group in experiment 3.

the same defect revealing ability, the guided group only required 56% of the testing effort of the random group.

6. Conclusion

This study models collaborative testing in a crowdsourcing environment as a job assignment problem and represents the problem using an ILP formulation. A greedy algorithm with four heuristic strategies is proposed to solve a large-scale collaborative testing problem. Based on the proposed algorithm, the COTS is implemented to evaluate the proposed model. The experiment results show that the approach is effective and performs well at testing and revealing defects. The experiment results also show that the average objective solution of the proposed algorithm is approximately 90% of the optimal solution. The approach can be applied in a real-time crowdsourcing environment and reduces testing efforts by almost 53%.

Crowdsourcing is a process where networked people collaborate to complete a cloud computing task. Different collaborative scenarios mean that various factors and constraints must be considered. Future studies should apply this approach to other phases of the software development life cycle, such as requirement development, software design, and code implementation, to accelerate system development in a crowdsourcing environment. Various factors, such as crowd expertise and knowledge, should also be incorporated into the collaborative testing approach.

Acknowledgments

This work was partially supported by the National Science Council of the Republic of China under grants NSC101–2511–S–468–007–MY3 and NSC101–2511–S–468–001.

References


Amazon Mechanical Turk, https://www.mturk.com/mturk/welcome (last accessed 01.12.11.).


"Appstore," http://www.appstore.com/ (last accessed 01.09.12.).


Open Source Web Applications with Source Code in ASP, JSP, PHP, Perl, ColdFusion, ASP.NET, etc., http://www.gotocode.com/ (last accessed 01.12.11.).


TopCoder, http://www.topcoder.com/ (last accessed 01.12.11.).


Utest, What we test, Available at: http://www.ustest.com/what-we-test (last accessed 01.12.11.).


Yuan-Hsin Tung is working toward the Ph.D. degree in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. He received a BS degree in Transportation Communication Management from the National Cheng-Kung University in Taiwan, and an MS in Information Management from the National Sun Yat-Sen University in Taiwan. Currently, he has been an Associate Researcher, since 2000, at the Test Center of Telecommunication Labs. of Chunghwa Telecom in Taoyuan, Taiwan. His main researches are cloud computing, knowledge engineering, data-mining technology, software testing, and software engineering.

Shian-Shyong Tseng received his Ph.D. degree in Computer Engineering from the National Chiao Tung University in 1984. He has been with the Department of Computer and Information Science at National Chiao Tung University since 1983. From 1988 to 1991, he was the director of the Computer Center National Chiao Tung University. From 1991 to 1992 and 1996 to 1998, he acted as the chairman of Department of Computer and Information Science. From 1992 to 1996, he was the Director of the Computer Center at Ministry of Education and the Chairman of Taiwan Academic Network (TANet) management committee. In December 1999, he founded Taiwan Network Information Center (TWNIC) and was the chairman of the board of directors of TWNIC from 2000 to 2005. He is now the Dean of College of Computer Science, Asia University. His current research interests include data mining, expert system, computer algorithm and Internet-based applications.