Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A novel approach to collaborative testing in a crowdsourcing environment

Yuan-Hsin Tung[a,b,*], Shian-Shyong Tseng[a,c,*]

[a] *Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, ROC*
[b] *Telecommunication Laboratory, Chunghwa Telecom Co., Ltd., Taiwan, ROC*
[c] *Department of Applied Informatics and Multimedia, Asia University, Taiwan, ROC*

ARTICLE INFO

Article history:
Received 7 April 2012
Received in revised form 1 January 2013
Accepted 21 March 2013
Available online 6 April 2013

Keywords:
Crowdsourcing
Cloud computing
Software testing
Collaborative testing
Integer linear programming

ABSTRACT

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.

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

* Corresponding authors at: Telecommunication Laboratory, Chunghwa Telecom Co., Ltd., Taiwan, ROC. Tel.: +886 928327053.
E-mail addresses: yhdong@cht.com.tw (Y.-H. Tung), sstseng@asia.edu.tw (S.-S. Tseng).

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. Riungu 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 be 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 potency. 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

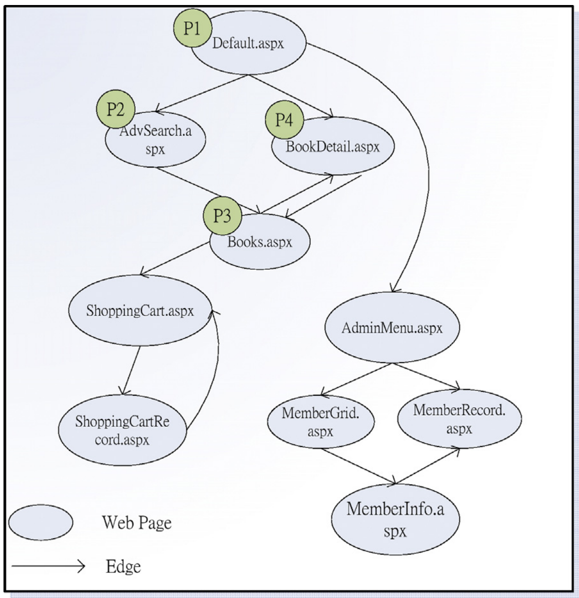


Fig. 1. An excerption of dependence graph for the Web application, BookStore.

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, execution time for each test case, and collaborative testing trustworthiness. 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 execution 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:

G	dependence graph; a dependence graph for a Web application is a directed graph, $G = (P, E)$;
P	set of nodes in the dependence graph;
E	set of edges in the dependence graph;
p_i	Web page, $p_i \in P$;
e_{ij}	a direct link from p_i to p_j, where $e_{ij} = (p_i, p_j)$, $e_{ij} \in E$, and $\forall\, p_i, p_j \in P$
M_j^k	input attribute-value pairs; $M_j^k = \langle (att_j^k(1), val_j^k(1)), (att_j^k(2), val_2^k(2)), \ldots \rangle$, where $att_j^k(x)$ is the input attribute and $val_j^k(x)$ is the input attribute value;
TC$_j$	test case TC$_j$ is a test path with input attribute-value pairs $TC_j = \langle (p_{j1}, M_j^1), (p_{j2}, M_j^2), (p_{j3}, M_j^3) \ldots \rangle, \forall p_j \in P$.

Fig. 1 shows the dependence graph that describes the control flow of the target Web application for collaborative testing by analyzing its program structures.

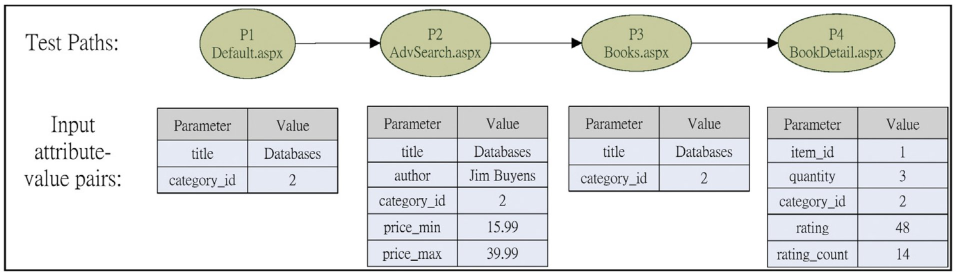
To illustrate this definition, a test case of Web application BookStore (Open Source Web Applications, 2011), (the open-source Web project) is shown in Fig. 2. In this application, testers select “Advanced Search” on page “Default.aspx” to connect to “AdvSearch.aspx” and search for books by typing “Databases” in the title field. Testers are then directed to “Books.aspx,” which displays the search results. Testers select a book and are then directed to “BookDetail.aspx,” showing their selection details. The test case is collected from each tester session log and represented as:

TC$_1$=<(Default.aspx, (category_id=‘2’, title=‘Databases’)), (AdvSearch.aspx, (title=‘Databases’, author=‘Jim Buyens’, category_id=‘2’, price_min=‘15.99’, price_max=‘39.99’)), (Books.aspx, (category_id=‘2’, title=‘Databases’)), (BookDetail.aspx, (item_id=‘1’, category_id=‘2’, quantity=‘3’, rating=‘48’, rating_count=‘14’))>

Fig. 3 shows 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 constraints.

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

i	number of test cases;
j	number of testers;
k	number of Web pages;
x_{ij}	$x_{ij} = 1$ means that the ith test case is assigned to jth tester, otherwise, $x_{ij} = 0$;
a_{ik}	$a_{ik} = 1$ means that the ith test case covers the kth page, otherwise, $a_{ik} = 0$;
A	test case page coverage matrix, $A = [a_{ik}]$;
t_{ij}	time taken for the jth tester to execute the ith test case;



Parameter	Value
title	Databases
category_id	2

Parameter	Value
title	Databases
author	Jim Buyens
category_id	2
price_min	15.99
price_max	39.99

Parameter	Value
title	Databases
category_id	2

Parameter	Value
item_id	1
quantity	3
category_id	2
rating	48
rating_count	14

Fig. 2. The example of test case in Web application, BookStore.

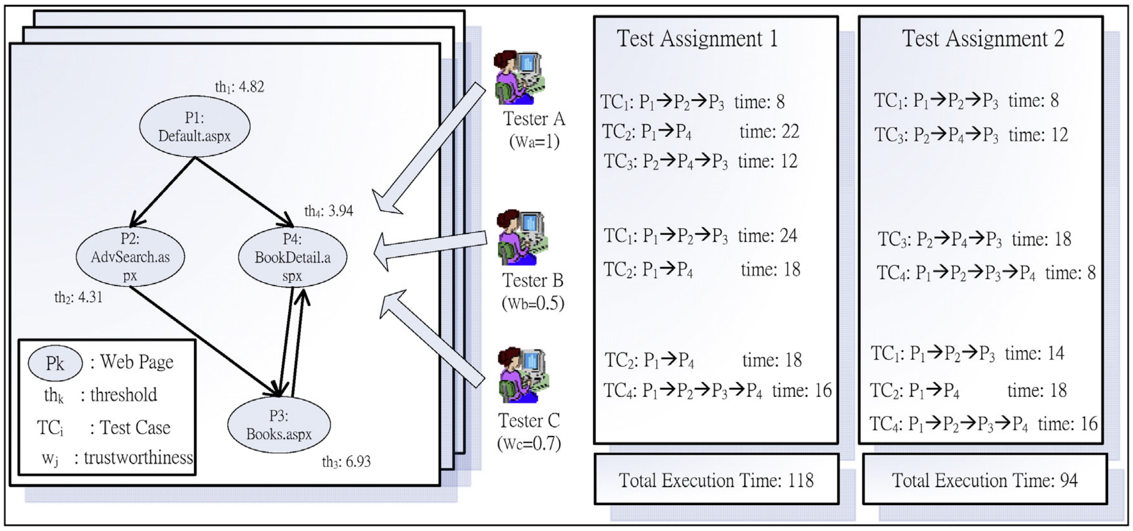


Fig. 3. An example of collaborative testing with dependence graph, test cases, and tester profiles on the crowdsourcing environment.

T	test case execution time matrix, $T=[t_{ij}]$;
w_j	trustworthiness of the jth tester;
W	tester trustworthiness matrix, $W=[w_j]$;
LoC_k	lines of code for the kth Web page, page complexity evaluation;
th_k	threshold of the kth Web page, $th_k=\frac{\mathrm{LoC}_k}{\sum_m \mathrm{LoC}_m}$;
TH	page threshold matrix, $TH=[th_k]$;
h_j	the jth tester's available time;
H	tester availability matrix, $H=[h_j]$.

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 (TC_1) in Fig. 3 covers three pages, P_1, P_2, and P_3, and is written as ($P_1\to P_2\to P_3$). 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 TC_1 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:

TC_1: Tester A executed the case in 8 min.

TC_2: Tester B executed the case in 18 min.

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: $P_1\to P_2\to P_3\to P_4$, Tester B: $P_1\to P_2\to P_3$, or Tester C: $P_1\to P_4$.

Table 1
Test case page coverage matrix, $A=[a_{ik}]$.

	P_1	P_2	P_3	P_4
TC_1	1	1	1	0
TC_2	1	0	0	1
TC_3	0	1	1	1
TC_4	1	1	1	1

Table 2
Test case execution time matrix, $T=[t_{ij}]$.

	Tester A	Tester B	Tester C
TC_1	8	24	14
TC_2	22	18	18
TC_3	12	18	24
TC_4	16	8	16

Table 3
Page threshold matrix, $TH=[th_k]$.

	P_1	P_2	P_3	P_4	Summary
Page Complexity (Lines of Code, LoC_k)	533	476	765	435	2209
Test threshold, th_k	4.82	4.31	6.93	3.94	20

Table 4
Tester trustworthiness matrix, $W=[w_j]$.

	Tester A	Tester B	Tester C
Experiences of Participation	3000	1500	2100
Trustworthy, w_j	1	0.5	0.7

Table 5
Tester availability matrix, $H=[h_j]$.

h_i	Tester A	Tester B	Tester C
Availability	48	24	48

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. $\sum_i \sum_j a_{ik} x_{ij} \geq 1$ for each page, [CC1: Coverage Constraint 1]where $a_{ik} = 1$, if the ith test case covers the kth page, otherwise $a_{ik} = 0$.

2. $\sum_k a_{ik} \geq 1$ for each test case. [CC2: Coverage Constraint 2]

3. $0 \leq \sum_i x_{ij} t_{ij} \leq h_j$ for each tester, [*TAC*: Tester Available Constraint]

where $t_{ij} > 0$ and $h_j > 0$.

4. $\sum_i \sum_j a_{ik} x_{ij} w_j \leq th_k$ 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 ith test case is assigned to the jth tester, otherwise $x_{ij} = 0$. Variable t_{ij} reflects the testing time of the jth tester for the ith 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 ith test case covers the kth Web 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 ith test case for the jth 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 jth 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

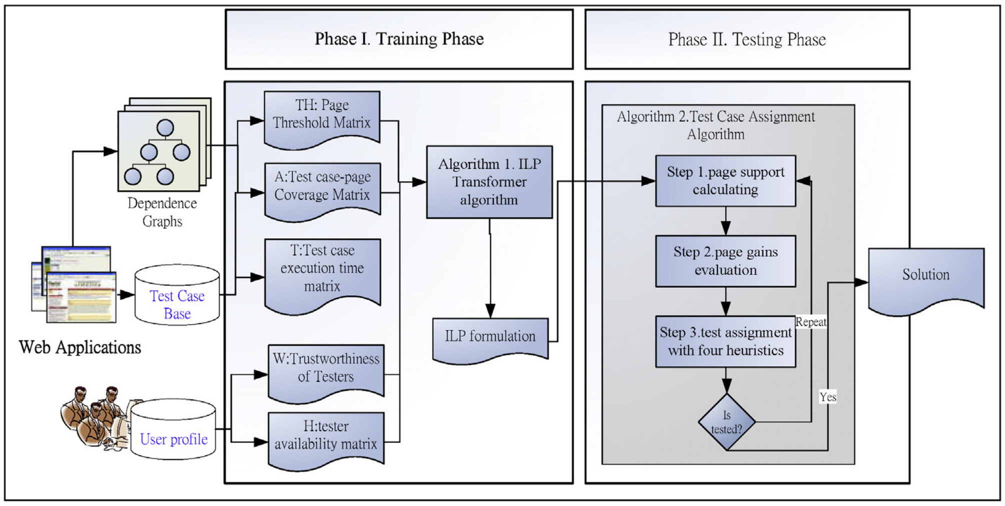

Phase I. Training Phase
Phase II. Testing Phase
Dependence Graphs
Web Applications
Test Case Base
User profile
TH: Page Threshold Matrix
A:Test case-page Coverage Matrix
T:Test case execution time matrix
W:Trustworthiness of Testers
H:tester availability matrix
Algorithm 1. ILP Transformer algorithm
ILP formulation
Algorithm 2.Test Case Assignment Algorithm
Step 1.page support calculating
Step 2.page gains evaluation
Step 3.test assignment with four heuristics
Is tested?
Repeat
Yes
Solution

Fig. 4. Architecture of crowdsourcing-based collaborative testing approach.

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 $x_{ij} = 1$ in the objective function, this means that the ith test case is assigned to the jth tester. The number of variables in the objective function is (i^*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_{ij} is the Binary Constraint (BC). There are: $(k+i)$ constraints for CC1 and CC2, j constraints for TAC, k constraints for TC, and i^*j constraints for BC. The proposed model consists of $(ij+i+j+2k)$ 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, $W = [w_j]$, tester availability matrix, $H = [h_j]$.
Output: collaborative testing in the ILP formulation, ILP = {OBJ, CONSTRAINTS}
Step 1. According to matrices A and T, let objective function $OBJ = \Sigma_i \Sigma_j x_{ij} t_{ij}$
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 $x_{ij}$.
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 t' 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 t' from candidate test case set T'. Calculate TCS(H)
for candidate test case t' and identify test case t' with maximum TCS(H).
Step 3.3. Assign test case t' 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.

$$\text{Page support vector}: U = \{u_k | \text{page support for} \quad k\text{th Web page}\},$$
$$\text{where page support}: u_k = \sum_i \sum_j a_{ik} x_{ij} w_j \quad . \tag{1}$$

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.

$$\text{Page gain}: Eval_k = \frac{(th_k - u_k)}{th_k}, \tag{2}$$

where th_k is the threshold of the kth page and u_k is the page support of the kth 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.

$$\begin{aligned}&\text{Heuristic 1}\,(H1): \text{maximal coverage heuristic}\\ &TCS(H1) = \max \textstyle\sum_i a_{ik},\end{aligned} \tag{3.1}$$

where $a_{ik} = 1$ means that the ith test case contains the kth page.

$$\begin{aligned}&\text{Heuristic}\quad 2\,(H2): \text{maximal}\quad \text{time}\quad \text{heuristic}\\ &TCS(H1) = \min t_{ij}\end{aligned} \tag{3.2}$$

where t_{ij} refers to the execution time of the ith test case by the jth tester.

$$\begin{aligned}&\text{Heuristic}\quad 3\,(H3): \text{maximal-trustworthiness}\quad \text{heuristic}\\ &TCS(H3) = \max w_j,\end{aligned} \tag{3.3}$$

where w_j represents the trustworthiness of tester j.

$$\begin{aligned}&\text{Heuristic}\quad 4\,(H4): \text{compound}\quad \text{heuristic}\\ &TCS(H4) = \min\left(\frac{T_{ij}}{\max \sum_i a_{ik}}\right)\end{aligned} \tag{3.4}$$

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)––$O(m \times n)$.

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 $O(n \times \log(n))$, 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

$T = O(m \times n) + 2 \times O(n) + \max(O(n^2), \quad O(n \times \log(n), 2 \times O(n^2)) = O(n^2)$.

Example:

Table 6 shows an example illustrating the collaborative testing approach processes. The top of the table shows page thresholds

Table 6
An example of results of test case assignment algorithm with heuristic *H2*.

Threshold (th_k)	Assignment	Page1 2.41		Page2 2.15		Page3 3.47		Page4 1.97		Test case suggestion
	Iteration	u_k	$Eval_k$	u_k	$Eval_k$	u_k	$Eval_k$	u_k	$Eval_k$	
Page Support (u_k) and Page Gain ($Eval_k$)	Initial	0	1	0	1	0	1	0	1	TC_3
	$r=1$	0	1	0.7	0.67	0.7	0.8	0.7	0.64	TC_4
	$r=2$	0.7	0.71	1.4	0.35	1.4	0.6	1.4	0.29	TC_1
	$r=3$	1.2	0.5	1.9	0.12	1.9	0.45	1.4	0.29	TC_2
	$r=4$	1.7	0.29	1.9	0.12	1.9	0.45	1.9	0.03	TC_1
	$r=5$	2.7	0	2.9	0	2.9	0.16	1.9	0.03	TC_3
	$r=6$	2.7	0	3.9	0	3.9	0	2.9	0	–

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 kth page. The maximal $Eval_k$ value means that the kth 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, TC_3, TC_4, TC_1, TC_2, TC_1, TC_3.

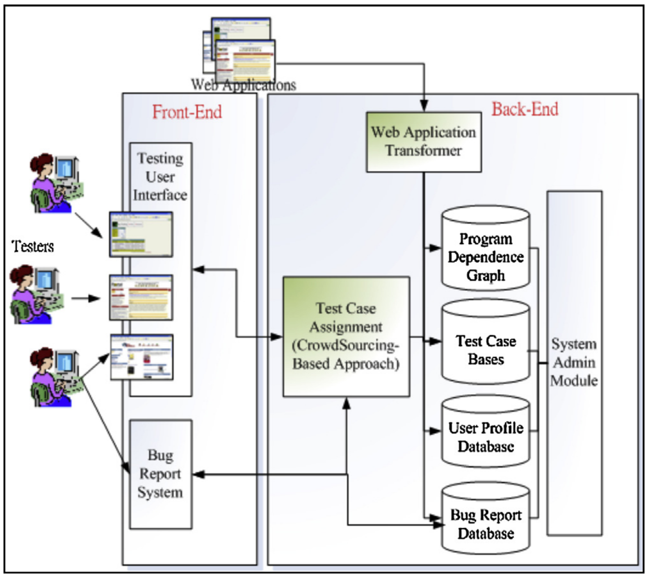


Fig. 5. System architecture of COTS.

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

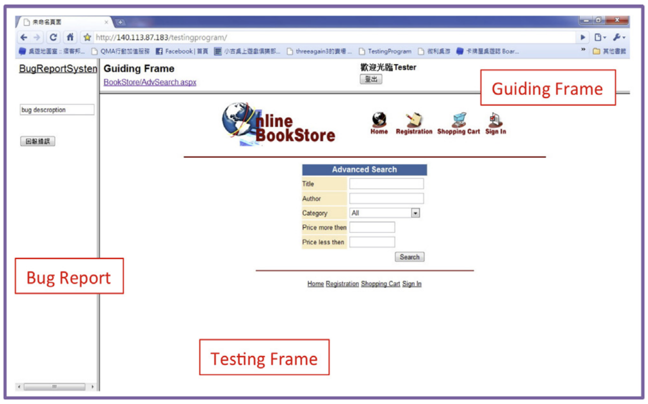
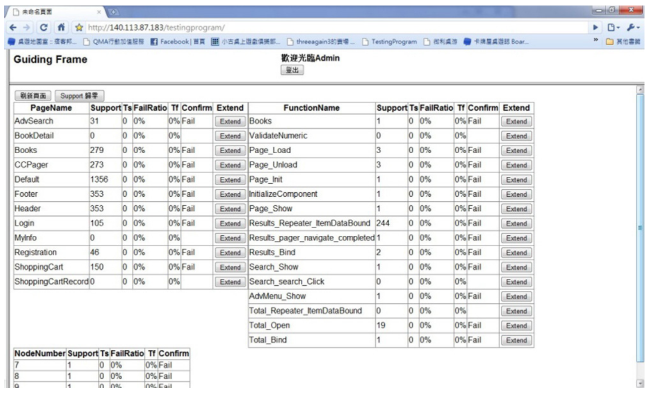


Fig. 6. Testing user interface of COTS.



Guiding Frame

歡迎光臨Admin

PageName	Support	Ts	FailRatio	Tf	Confirm	Extend
AdvSearch	31	0	0%	0%	Fail	Extend
BookDetail	0	0	0%	0%		Extend
Books	279	0	0%	0%	Fail	Extend
CCPager	273	0	0%	0%	Fail	Extend
Default	1356	0	0%	0%	Fail	Extend
Footer	353	0	0%	0%	Fail	Extend
Header	353	0	0%	0%	Fail	Extend
Login	105	0	0%	0%	Fail	Extend
MyInfo	0	0	0%	0%		Extend
Registration	46	0	0%	0%	Fail	Extend
ShoppingCart	150	0	0%	0%	Fail	Extend
ShoppingCartRecord	0	0	0%	0%		Extend

FunctionName	Support	Ts	FailRatio	Tf	Confirm	Extend
Books	1	0	0%	0%	Fail	Extend
ValidateNumeric	0	0	0%	0%		Extend
Page_Load	3	0	0%	0%	Fail	Extend
Page_Unload	3	0	0%	0%	Fail	Extend
Page_Init	1	0	0%	0%	Fail	Extend
InitializeComponent	1	0	0%	0%	Fail	Extend
Page_Show	1	0	0%	0%	Fail	Extend
Results_Repeater_ItemDataBound	244	0	0%	0%	Fail	Extend
Results_pager_navigate_completed	1	0	0%	0%	Fail	Extend
Results_Bind	2	0	0%	0%	Fail	Extend
Search_Show	1	0	0%	0%	Fail	Extend
Search_search_Click	0	0	0%	0%		Extend
AdvMenu_Show	1	0	0%	0%	Fail	Extend
Total_Repeater_ItemDataBound	0	0	0%	0%		Extend
Total_Open	19	0	0%	0%	Fail	Extend
Total_Bind	1	0	0%	0%	Fail	Extend

NodeNumber	Support	Ts	FailRatio	Tf	Confirm
7	1	0	0%	0%	Fail
8	1	0	0%	0%	Fail
9	1	0	0%	0%	Fail

Fig. 7. System Admin of COTS for monitoring the collaborative testing.

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 constructed for collaborative testing. Testing engineers selected the test cases according to the features of Web application BookStore. As shown in Table 1, *i* 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 subcolumns 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.

	Test case (i)	Tester (j)	Variable ($i \times j$)	Execution time (s)	Objective optimal
ex1-1	35	15	525	282.75	2705
ex1-2	40	18	720	530.79	2355
ex1-3	45	21	945	752.28	2275
ex1-4	50	24	1200	707.25	2195
ex1-5	55	27	1485	2991.73	2150
ex1-6	60	30	1800	2651.95	2100
ex1-7	65	33	2145	2851.46	2095
Average	50	24	1260	1538.32	2267.86

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

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

	Test Case (i)	Tester (j)	Variable ($i \times j$)	ILP		Heuristic algorithm			
				Execution Time (s.)	Objective Optimal	Obj.H1	Obj.H2	Obj.H3	Obj.H4
ex1-1	35	15	525	282.75	2705	4265	3225	3805	3315
ex1-2	40	18	720	530.79	2355	4295	2690	3755	2745
ex1-3	45	21	945	752.28	2275	4125	2595	4240	2670
ex1-4	50	24	1200	707.25	2195	3755	2470	3475	2470
ex1-5	55	27	1485	2991.73	2150	3360	2235	3250	2395
ex1-6	60	30	1800	2651.95	2100	3270	2190	3240	2185
ex1-7	65	33	2145	2851.46	2095	3230	2170	3000	2105
Avg.	50	24	1260	1538.32	2267.86	3757.14	2510.71	3537.86	2555

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

	Test Case (i)	Tester (j)	Variable ($i \times j$)	ILP		Heuristic algorithm			
				Execution Time (s.)	Objective Optimal	Obj.H1	Obj.H2	Obj.H3	Obj.H4
ex2-1	30	20	600	2.46	2485	3140	3520	4760	3705
ex2-2	35	20	700	5.48	2395	3930	3385	4100	3505
ex2-3	40	20	800	23.53	2300	3945	2770	3570	2810
ex2-4	45	20	900	40.99	2270	3930	2920	3960	2780
ex2-5	50	20	1000	19.33	2225	3795	2675	3870	2725
ex2-6	55	20	1100	80.11	2340	3375	2680	3740	2675
ex2-7	60	20	1200	84.37	2150	3300	2660	3930	2700
ex2-8	65	20	1300	1001.86	2240	3480	2535	3925	2605
ex2-9	70	20	1400	2599.27	2130	3390	2545	4265	2585
Avg.	50	20	1000	428.6	2281.67	3587.22	2854.44	4013.33	2898.89

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

	Test Case (i)	Tester (j)	Variable ($i \times j$)	ILP		Heuristic algorithm			
				Execution time (s.)	Objective optimal	Obj.H1	Obj.H2	Obj.H3	Obj.H4
ex3-1	70	15	1050	402	2245	3515	2670	4990	2760
ex3-2	70	20	1400	682	2045	3515	2585	4660	2485
ex3-3	70	25	1750	970	2035	3250	2245	3930	2325
ex3-4	70	30	2100	1837	1990	3220	2255	3905	2345
ex3-5	70	35	2450	2671	2125	3310	2275	4160	2340
ex3-6	70	40	2800	7200[a]	2110[b]	3195	1980	3120	2000
Avg.	70	27.5	1925	2293.67	2091.67	3334.17	2335	4127.5	2375.83

[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.

Group	Crowd tester	Execution times (min)	Avg. execution time (min)	Numbers of bug reports	Defect coverage
Random Group	85	16,740	196.94	97	100%
Guided Group	59	8983	152.25	55	100%

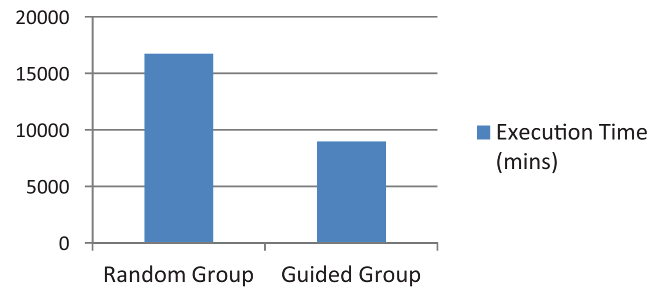


Fig. 8. Total execution time of guide group and random group in experiment 3.

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

Abdullah, R., Lakulu, M., Ibrahim, H., Selamat, M.H., Nor, M.Z.M., 2009. The challenges of open source software development with collaborative environment. In: International Conference on Computer Technology and Development.

Albrecht, A.J., Gaffney Jr., J.E., 1983. Software function, source lines of code, and development effort prediction: a software science validation. IEEE Transaction on Software Engineering SE-9 (6), 639–648.

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

Amazon_Mechanical_Turk Wikipedia, http://en.wikipedia.org/wiki/Amazon_Mechanical_Turk (last accessed 01.12.11.).

Andrews, A.A., Offutt, J., Alexander, R.T., 2005. Testing web applications by modeling with FSMs. Software Systems and Modeling, 326–345, July.

"AppStori," http://www.appstori.com/ (last accessed 01.09.12.).

Bai, X., Cao, Z., Chen, Y., 2007a. Design of a trustworthy service broker and dependence-based progressive group testing. International Journal of Simulation and Process Modelling 3 (1), 66–79.

Bai, X., Wang, Y., Dai, G., Tsai, W.-T., Chen, Y., 2007b. A framework for contract-based collaborative verification and validation of web services. Component-Based Software Engineering, vol. 4608. Springer, Berlin/Heidelberg 258–273.

Benedikt, M., Freire, J., Godefroid, P., 2002. VeriWeb:automatically testing dynamic web sites. In: Proceedings of 11th International World Wide Web Conference, Honolulu, HI, USA, pp. 654–668, May.

Bertolino, A., 2007. Software testing research: achievements, challenges, dreams. In: Briand, A., Wolf, L. (Eds.), Future of Software Engineering. IEEE-CS Press, IEEE Computer Society, Washington, DC, USA.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT Platforms vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer System 25, 599–616.

Collective_intelligence Wikipedia, http://en.wikipedia.org/wiki/Collective_intelligence (last accessed 01.12.11.).

CPLEX, IBM ILOG CPLEX Optimizer, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, (last accessed 01.12.11.).

Crowdsourcing Wikipedia, http://en.wikipedia.org/wiki/Crowdsourcing (last accessed 01.12.11.).

Deng, Y., Frankl, P., Wang, J., 2004. Testing web database applications. SIGSOFT Software Engineering Notes 29 (5), 1–10.

Elbaum, S., Rothermel, G., Karre, S., Fisher II, M., 2005. Leveraging User Session Data to Support Web Application Testing. IEEE Transaction on Software Engineering 31 (3), 187–202, March.

Garey, M.R., Johnson, D.S., 1979. In: Klee, V. (Ed.), Computers and Intractability, a Guide to the Theory of NP-Completeness. Freeman, New York.

Hedberg, H., 2004. Introducing the next generation of software inspection tools. In: Product Focused Software Process Improvement (LNCS 3009), pp. 234–247.

Held, M., Blochinger, W., 2009. Structured collaborative workflow design. Future Generation Computer System 25, 638–653.

Homma, K., Izumi, S., Takahashi, K., Togashi, A., 2011. Modeling, verification and teesting of web applications using model checker. IEICE Transactions on Information & System E94-D (5), May.

Howe, J., 2006. The rise of crowdsourcing. Wired Magazine 14 (06), 17–23.

Jeffrey, D., Gupta, N., 2005. Test suite reduction with selective redundancy. In: Proceedings of the 21st IEEE International Conference Software Maintenance, pp. 549–558.

Leon, D., Masri, W., Podgurski, A., 2005. An empirical evaluation of test case filtering techniques based on exercising complex information flows. In: Proceedings of the 27th International Conference Software Engineering, pp. 412–421.

Liu, C.-H., Kung, D.C., Hsia, P., 2000. Object-based data flow testing of web applications. In: Proceedings of the First Asia-Pacific Conference Quality Software, pp. 7–16.

Low, G.G., Jeffery, D.R., 1990. Function points in the estimation and evaluation of the software process. IEEE Transaction on Software Engineering 16 (1), 64–71.

Lucca, G.A.D., Fasolino, A.R., 2006. Testing web-based applications: the state of the art and future trends. Information and Software Technology 48, 1172–1186.

Macdonald, F., Miller, J., 1999. A comparison of computer support systems for software inspection. Automated Software Engineering 6 (3), 291–313.

Miao, H., Qian, Z., Song, B., 2008. Towards automatically generating test paths for Web application Testing. In: International Symposium on Theoretical Aspects of Software Engineering, 2nd IFIP/IEEE.

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

Pearson, K., 1987. Mathematical contributions to the theory of evolution. on the law of ancestral heredity. Proceedings of the Royal Society of London 62, 386–412.

Ricca, F., Tonella, P., 2001. Analysis and Testing of Web Applications. In: Proceedings of the 23rd International Conference on Software Engineering, Toronto, ON, Canada, pp. 25–34.

Ricca, F., Tonella, P., 2006. Detecting anomaly and failure in web applications. IEEE MultiMedia 13 (2), 44–51.

Riungu, L.M., Taipale, F.O., Smolander, K., 2010. Research issues for software testing in the cloud. In: Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW 2010).

Shahriar, H., Zulkernine, M., 2011. Trustworthiness testing of phishing websites: a behavior model-based approach. Future Generation Computer System, http://dx.doi.org/10.1016/j.future.2011.02.001.

Shukla, S.V., Redmiles, D.F., 1996. Collaborative learning in a software bug-tracking scenario. In: Workshop on Approaches for Distributed Learning through Computer Supported Collaborative Learning, Boston, MA.

Souza, C.R., Quirk, S., Trainer, E., Redmiles, D.F., 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. In: Proceedings of the 2007 International ACM Conference on Supporting Group Work ACM New York, NY, USA.

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

Tsai, W.T., Chen, Y., Paul, R., Liao, N., Huang, H., 2004. Cooperative and group testing in verification of dynamic composite web services. In: Proceedings of 28th Annual Int. Computer Software and Applications Conf. (COMPSAC 2004), vol. 2, pp. 170–173.

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

Wang, M., Yuan, J., Miao, H., Tan, G., 2008. A static analysis approach for automatic generating test cases for Web applications. International Conference on Computer Science and Software Engineering.

West, A.G., Chang, J., Venkatasubramanian, K.K., Lee, I., 2011. Trust in collaborative web applications. Future Generation Computer System, http://dx.doi.org/10.1016/j.future.2011.02.007.

Weyers, B., Luther, W., Baloian, N., 2011. Interface creation and redesign techniques in collaborative learning scenarios. Future Generation Computer System 27, 127–138.

Whitehead, J., 2007. Collaboration in software engineering: a roadmap. Future of Software Engineering (FOSE'07).

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.