Ant Colony System With Sorting-Based Local Search for Coverage-Based Test Case Prioritization

Chengyu Lu, Jinghui Zhong, Yinxing Xue, Liang Feng, and Jun Zhang, *Fellow, IEEE*

***Abstract*—Test case prioritization (TCP) is a popular regression testing technique in software engineering field. The task of TCP is to schedule the execution order of test cases so that certain objective (e.g., code coverage) can be achieved quickly. In this article, we propose an efficient ant colony system framework for the TCP problem, with the aim of maximizing the code coverage as soon as possible. In the proposed framework, an effective heuristic function is proposed to guide the ants to construct solutions based on additional statement coverage among remaining test cases. Besides, a sorting-based local search mechanism is proposed to further accelerate the convergence speed of the algorithm. Experimental results on different benchmark problems, and a real-world application, have shown that the proposed framework can outperform several state-of-the-art methods, in terms of solution quality and search efficiency.**

***Index Terms*—Ant colony system (ACS), regression testing, statement coverage, test case prioritization (TCP).**

I. Introduction

REGRESSION testing is an important software testing process in software engineering [1]–[5]. The goal of regression testing is to ensure that any new modifications of the software should not adversely affect the existing functions, and there is no new fault hidden in the inserted code. To achieve this goal, regression testing is often performed throughout the entire software development and maintenance life cycle. However, as the software becomes large, the number of the testing cases will increase over time, and the cost of performing regression testing could become more and more expensive. For example, as reported by Elbaum *et al.* [6], it takes seven weeks to execute the entire test suite for one of their industrial products. How to reduce the regression testing time is still a challenging research topic in the software engineering community.

Over the past decades, various methods have been proposed to reduce the computational cost of regression testing [3], [7]–[9]. In particular, the coverage-based test case prioritization (TCP) is one of the most effective and important approaches. The key idea of coverage-based TCP is to schedule the execution order of test cases and let sorted test cases with higher priorities to be run earlier. The goal is to maximize the code coverage as soon as possible [10]–[15].

Code coverage, as a prioritization goal, possesses many advantages of its own. First, since it can never be foreseen before the testing activities that how many faults are hidden in the code, as well as which test cases are able to expose them, the fault exposure ability, though being one of the most common criteria, cannot be adopted directly to prioritize test cases. Code coverage, on the other hand, serves as a substitutive goal and has been proven to be effective when approximating the rate of fault detection [3], [9], [16]–[20]. Besides, code coverage itself is also an irreplaceable concern such as being an avionics standard [21] and an independent measurement reflecting on a test suite's quality [22]. Moreover, Chen *et al.* [23] also found that optimizing the code coverage can calibrate the reliability estimation of a software, which predicts the probability that a program can survive an environment within a time interval.

The coverage-based TCP is an NP-hard optimization problem [3]. Existing methods for the problem include deterministic approaches [9], [20], [24], [25] and stochastic searching methods [11]–[15]. Among others, additional greedy strategy [3] is one of the most effective deterministic approaches. However, greedy-based approaches can only find near-optimal solutions. Stochastic searching methods, on the other hand, possess the advantages of strong global searching ability and model independence. Currently, stochastic searching methods for coverage-TCP problems include genetic algorithm (GA)-based approaches [7], [11]–[14] and ant colony optimization (ACO)-based approaches [15], [26]–[28]. GA-based techniques search blindly with random genetic operators (i.e., mutation, crossover, and selection), and the searching direction is driven only by fitness. ACO-based techniques, on the other hand, are capable of utilizing both problem-specific knowledge (i.e., heuristic function) and historical search information (i.e., pheromone value) to guide the search. ACO-based techniques have been

Manuscript received November 12, 2018; revised March 26, 2019 and June 6, 2019; accepted July 14, 2019. Date of publication August 9, 2019; date of current version August 31, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61602181 and Grant 61876025, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X183, in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003, and in part by the Guangdong-Hong Kong Joint Innovation Platform under Grant 2018B050502006. The work of Y. Xue was also supported by the CAS Pioneer Hundred Talents Program. Associate Editor: T. H. Tse. *(Corresponding author: Jinghui Zhong.)*

C. Lu and J. Zhong are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: chengyulu_cs@outlook.com; jinghuizhong@gmail.com).

Y. Xue is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: yxxue@ustc.edu.cn).

L. Feng is with the School of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: liangf@cqu.edu.cn).

J. Zhang is with the Victoria University, Melbourne VIC 8001, Australia (e-mail: junzhang@ieee.org).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2930358

proven to perform better than GA-based techniques such as NSGA-II [29] for coverage-based TCP problems [15]. In the literature, several ACOs have been proposed to solve coverage-based TCP problems [15], [26], [27]. Particularly, Bian *et al.* [15] have recently proposed a multiobjective ACO framework, which optimizes the effective execution time (EET) and the rate of statement coverage simultaneously. In designing their ACO, a genetic biological concept, which is referred to as Epistasis, was borrowed to represent a specific segment of coverage-based TCP solutions, and a special pheromone updating strategy, which utilizes the Epistasis segment, was proposed to enhance the history-best solution. They empirically confirmed the effectiveness of narrowing down the search space and accelerating the convergence through the Epistasis-based pheromone updating strategy. However, existing ACO-based methods for coverage-based TCP problems focus on utilizing pheromone to guide the search, while ignoring the heuristic function. The heuristic function is a key component of ACO, which is designed based on the specific problem to accelerate the search. Thus, existing ACOs for coverage-based TCP problems are not effective enough to utilize domain knowledge to improve the search efficiency.

In this article, to efficiently solve the coverage-based TCP problem, we propose an enhanced ACO framework, which is capable of utilizing both domain-specific knowledge and historical search information to guide the search. We design an additional coverage-based heuristic (ACB-Heuristic) function to assist the search of the proposed ACO framework. Also, we discover and mathematically prove an important property of coverage-based TCP problems and further design a new local search mechanism based on the above discovery. Generally, the major contributions of the article are as follows.

1) First, an ACB-Heuristic function is proposed to help artificial ants choose their next states according to problem-specific domain knowledge, i.e., the number of not-yet-covered statements in the remaining test cases. This heuristic incorporates the searching efficacy of the state-of-the-art additional greedy strategy with the global searching ability of the ACO methodology, so that the proposed framework can avoid the drawback of both the existing deterministic and stochastic techniques.
2) Second, we propose a sorting-based local search (SB-LS) mechanism to fine-tune the solutions constructed by artificial ants. Based on a key property of the solutions of coverage-based TCP problems, this local search mechanism is able to improve both the solution quality and the search efficiency. Furthermore, this local search is general enough to be applied to improve the quality of solutions generated by other methods for coverage-based TCP problems.
3) In addition, we conduct comprehensive experiments to evaluate the proposed ACO framework. Several state-of-the-art approaches are adopted for comparison. A number of benchmark problems and practical problems are utilized for testing. The results have demonstrated the high efficacy and generality of the proposed ACO.

The rest of this article is organized as follows. Section II presents the preliminaries of the coverage-based TCP problem and the ACO methodology and reviews some related works. Section III gives the proposed ACO framework. Section IV provides the experiments. Section V discusses the threats to validity of the study. Finally, Section VI concludes this article.

TABLE I
TEST SUITE OF FIVE TEST CASES COVERING A PROGRAM WITH NINE STATEMENTS

test case	statements								
	1	2	3	4	5	6	7	8	9
A	x				x				
B	x				x	x	x		
C	x	x	x	x	x	x	x		
D					x				
E							x	x	x

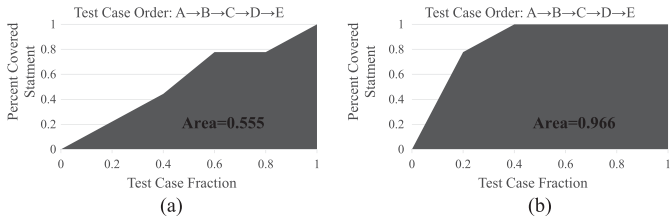


Fig. 1. Example illustrating the APSC measure. (a) Pri.1 (b) Pri.2.

II. PRELIMINARIES

A. Problem Definitions

The TCP problem in regression testing can be mathematically expressed as follows [3].

Definition II.1. Given a test suite (T), the set of permutations of T (PT), and a function from PT to real numbers (f), the task for TCP problem is to find $T^{'} \in PT$ such that $(\forall T^{''})(T^{''} \in PT)(T^{''} \neq T^{'})[f(T^{'}) \geq f(T^{''})]$.

In the above definition, PT stands for the set of all possible orderings of T and f is a function that maps an order to a value.

The rate of statement coverage measures how quickly the statements in the source program can be covered by a sequence of prioritized test cases. The average percentage of statement coverage (APSC) is one of the most widely used metrics for rate of statement coverage [10], which is computed as follows:

$$\text{APSC}\left(T^{'}\right) = 1 - \frac{TS_1 + TS_2 + \cdots + TS_m}{mn} + \frac{1}{2n} \quad (1)$$

where $T^{'}$ is a candidate permutation of T, m is the number of statements in the tested program, n is the number of test cases in T, and TS_i is the sequential number of the test case that first covers statement i (e.g., $TS_2 = 5$ means that the first test case in $T^{'}$ that covered statement 2 is the fifth test case). The value of the APSC metric varies from 0 to 1. A higher value implies a faster statement coverage rate. The letters m and n are specifically referred to as the number of statements and the number of test cases, respectively.

Consider an illustrating program with nine statements and a test suite of five test cases with covered statements listed in Table 1. From Fig. 1(a) and (b), the APSC values of the two solutions correspond to the shaded areas under the curves,

which increase in ascending order. As can be observed, it is clear that the prioritized suite Pri.2 with an ordering of $C{\rightarrow}E{\rightarrow}A{\rightarrow}B{\rightarrow}D$ leads to the earliest coverage of the most statements, possessing the highest APSC value of 0.966.

B. Ant Colony Optimization

ACO is a class of metaheuristic algorithms inspired by the foraging activities of natural ants. For years, ACO algorithms have been applied to a wide range of combinatorial optimization problems [30]–[34] and demonstrated robust and versatile optimization capabilities.

One of the most effective ACO algorithms is the ant colony system (ACS) [35]. In the ACS, a problem-dependent heuristic evaluation function (η), which serves as a measurement on the quality of candidate components, has been added to the partially built solution. Artificial ants decide the next vertex to transfer to utilizing a probabilistic state transition rule, which is called the pseudorandom-proportional (PRP) rule [35]. This rule is expressed as follows:

$$s = \begin{cases} \underset{u \in J_k(r)}{\arg\max} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\}, & \text{if } q \le q_0 \\ & \text{(exploitation)} \\ S, & \text{otherwise} \\ & \text{(exploration)} \end{cases} \tag{2}$$

where r is the vertex an artificial ant k currently locates on. s is the next vertex k is going to visit. $J_k(r)$ is the set of vertices connected to r but has never been visited by k. η is the heuristic function. τ returns the pheromone amount of edge (r, s). β is a parameter determining the relative importance between τ and η. q is a random number uniformly distributed in $[0, 1]$. q_0 is a threshold between 0 and 1 determining the relative importance of exploitation versus exploration. S is a random variable selected according to the probability distribution called the random-proportional rule given as follows:

$$P_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]\cdot[\eta(r,s)]^{\beta}}{\sum\limits_{u \in J_k(r)} [\tau(r,u)]\cdot[\eta(r,u)]^{\beta}}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

by which edges with greater η and τ are more favored by the artificial ants.

A pheromone updating rule indicates how the pheromone amount on the graph changes with respect to time, and it usually takes into account the evaporation as well as the reinforcement of the trails. The ACS introduces two pheromone updating rules, which are the local pheromone updating (LPU) rule and the global pheromone updating (GPU) rule. The role of the LPU rule in the ACS is to shuffle the tours [35] by reducing the pheromone amount of edges that are frequently crawled by ants. The purpose of the GPU rule is to evaporate a certain amount of pheromone on all edges, while reinforcing the pheromone amount of the path indicated by the history-best solution T_H so that the algorithm can eventually converge to it.

The LPU rule is performed by all ants after each transition. Supposing that r is the vertex, where an ant is located on before transition, and s is the vertex this ant transfers to, the LPU rule can be expressed as follows:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \tag{4}$$

where $\rho \in (0, 1]$ is the pheromone decay coefficient. τ_0 is the initial pheromone amount in the graph, which is usually smaller than $\tau(r, s)$. By evaporating some pheromone, edges that are usually crawled on by ants become less desirable. In essence, the LPU rule encourages ants to explore vertices that are fewer visited to broaden the searching region.

The GPU rule is performed once in every iteration, after all ants have finished their tours. It can be defined as follows:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \begin{cases} \Delta\tau, & \text{if } (r,s) \in T_H \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where $\alpha \in (0, 1)$ is the pheromone decay parameter indicating the rate of evaporation, T_H is the history-best solution, $\Delta\tau$ is a problem-specific variable, which is usually related to T_H. By this means, the trail indicated by T_H will become more and more prominent. In the meanwhile, the PRP rule allows ants to explore on the neighborhood of this trail instead of simply repeating the history-best solution. In essence, GPU and PRP rules combine to constantly improve the history-best solution and gradually converge the ant colony on this solution.

C. Related Works

TCP is an active research topic in the regression testing field. Despite various prioritization goals [7], [9], [36]–[38], boosting the rate of fault detection is one of the most prevalent objectives [25]. However, in TCP problems, the locations of faults are unknown before test execution. Code coverage (e.g., statement coverage) is one of the most widely used approximations for fault detection [9], [15], [24], [39]. Studies such as [10], [20], and [40] emphasized the importance of the coverage of a test suite to the suite's effectiveness in detecting faults.

Existing techniques for coverage-based TCP can be classified into two categories [41]. The first category is deterministic methods. The key idea is to use greedy heuristic. An additional greedy strategy [3], [4] is one of the greedy-based techniques with respect to fault detection. Li *et al.* [10] further extended it to utilize code coverage [10] and proved that the additional greedy strategy is the most efficient among a total of five algorithms. Taking statement coverage as an example, the additional greedy algorithm considers the number of uncovered statements. Hence, the priority of candidate test cases depends on the number of statements covered by them but not by the partially constructed solution. By far, there have been many additional greedy strategy variants developed [9], [20], [24], and the additional greedy strategy is still one of the most effective techniques for coverage-based TCP problems [25]. However, since coverage-based TCP problems are NP-hard optimization problems, deterministic approaches can only find local optimal solutions.

The second category of coverage-based TCP problems is stochastic searching. This class of approaches possesses strong global searching ability. Compared to deterministic approaches, stochastic searching approaches have the potential to avoid local optima. Two major stochastic-searching-based approaches are GA and ACO. It is shown that in recent years, both GA and ACO have been getting more and more popular for solving TCP problems [41].

Li *et al.* [10] implemented a GA variant and suggested that the automated discovery capability of the GA is encouraging. Konssard and Ramingwong [11] modified the GA based on total coverage information of test cases, and the average percentage of condition coverage was measured. Experimental results indicated that the GA could outperform other five approaches including the artificial bee colony optimization. Jun *et al.* [13] improved the GA to achieve faster convergence, and the average percentage of block coverage was evaluated. Besides single-objective GAs, NSGA-II [29] was also adopted to solve multiobjective TCP [14], test case selection [7], and test suite minimization [39] problems. The key idea of the GA is to use genetic operators to conduct searching, and the evolution of the GA is driven only by the fitness of the history-best individual. These make the GA a blind search, which means the domain knowledge of the problem is not utilized.

In 2010, Singh *et al.* [26] first applied ACO to solve the TCP problems. They demonstrated the strong and robust performance of ACO and obtained TCP solutions close to the global optimum. Suri and Singhal [27] proposed to use ACO to prioritize test cases under the goals of optimizing both the rate of fault detection and the EET simultaneously. In this method, artificial ants select test cases based on both the amount of pheromone and randomness. Furthermore, Bian *et al.* [15] proposed an Epistasis-based ACO (E-ACO) to solve the multiobjective TCP problem, which is to maximize the APSC while minimizing the EET in the meanwhile. In Epistasis ACO, a special pheromone updating rule is designed to enhance edges that are related to a so-called Epistasis test segment. Pheromone information in ACO provides artificial ants' searches with historical search experience, which helps them converge to the history-best solution. In the meanwhile, another component helps the ants explore around the history-best solution so that the history-best can be constantly improved, which is the heuristic function. Through a biased random selection mechanism, which utilizes heuristic function, artificial ants are able to search globally. However, the above three works ignored the design of heuristic functions, which prevented the ants from utilizing problem-specific local information to produce better solutions.

The difference between this article and existing works is that we utilize problem-specific knowledge to guide the search procedure. Here, the problem-specific knowledge means the actual coverage information of the test cases about those not-covered-yet statements, which is essentially the basic idea of the state-of-the-art additional greedy strategy. Both an ACB-Heuristic function and an SB-LS mechanism are designed based on this knowledge. Comprehensive experiments are conducted to compare the efficiency of the proposed method and several state-of-the-art methods and to investigate the effectiveness of the above two components.

III. Proposed Method

In this section, we present a coverage-based ACS (CB-ACS) for TCP problems. First, we present the algorithm framework in Section III-A. Second, the ACB-Heuristic function is presented in Section III-B. Third, the global best tour (GBT) tree aided pheromone updating rule is presented in Section III-C. Finally, the SB-LS mechanism is presented in Section III-D.

A. General Framework

The TCP problem with an objective to maximize the rate of statement coverage of a test suite can be converted to an optimization problem as follows.

Problem:

$$\text{minimize } \sum_{i=1}^{n} i \cdot \Phi(t_i)$$

$$\text{subject to } \sum_{i=1}^{n} \Phi(t_i) = m$$

where t_i is the ith test case in the ordered test suite and $\Phi(t_i)$ returns the additional coverage obtained by executing t_i.

One key issue of applying ACS is to properly map the TCP problem to a construction graph representation. Considering a test suite $T = \{t_1, t_2, \ldots, t_n\}$, in the proposed method, all the test cases in T are regarded as vertices on a graph $G = \{V, E\}$, where V is the set of vertices and E is the set of edges. $V = \{v_{-1}, v_1, v_2, \ldots, v_n\}$ contains $n + 1$ elements in total, with one virtual vertex v_{-1} that does not symbolize any test case and n real vertices mapping to the n test cases.

In coverage-based TCP problems, the first test case usually has a large influence on the final performance of the whole test case order. Therefore, the first test case should be treated equally as all other test cases, which means that it would be better to let the artificial ants themselves choose their own starting vertices. Hence, v_{-1} serves as a virtual starting vertex for all the ants before they start their tours. Correspondingly, the edge set $E = \{E_v, E_r\}$ consists of two parts: the subset of the virtual edges $E_v = \{e(-1, 1), e(-1, 2), \ldots, e(-1, n)\}$ containing edges from v_{-1} to all other real vertices, and the subset of the common real edges E_r.

Apart from that, each vertex except v_{-1} contains its own covered statement set $Sv_i = \{s_{i1}, s_{i2}, \ldots, s_{ij}\}(1 \le i \le n)$, where j is the number of elements in Sv_i and s_{i1}, s_{i2},... are the sequential number of the covered statements, taking value from 1 to m. Fig. 2 illustrates an example of the proposed graph representation.

In the proposed representation, any vertex (both virtual and real) cannot be visited twice by the same ant during one iteration. The sequential order of vertices in an artificial ant's tour is a feasible solution. Given that the TCP techniques do not abandon test cases, theoretically, a solution is built only after the ant has traveled along all the vertices on the graph. In many cases, however, the full statement coverage is satisfied ahead of the accomplishment of the ant's traversal, and the subsequent test cases will have no effect on the rate of statement coverage. In order to save computational time, ants are allowed to stop wherever they obtained full statement coverage, and the subsequent test cases are added at the end of the partial solution in ascending order. In the following of this article, all the tours mentioned are referred to as the partial solution from the virtual starting point v_{-1} to the test case in which statement coverage is fulfilled.

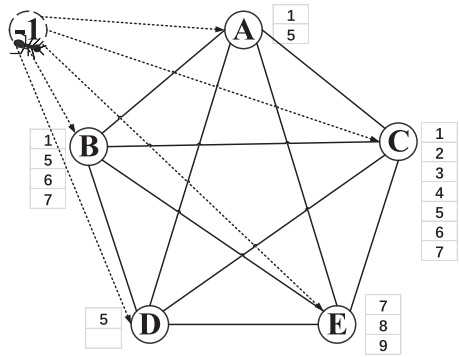


Fig. 2. Example of the conversion from the test suite in Table I to the corresponding graph. The undirected and solid lines are common real edges, while the directed and one-way dash lines are virtual edges.

The determination of the first history-best solution T_{H_0} is essential since it decides the value of τ_0, which is used both as the initial pheromone amount on G and the pheromone increment of the LPU rule in (4). T_{H_0} is determined through a *conditional random push* (CRP) heuristic. Every time, a random test case that has not been selected is added to the back of the partially constructed solution if this test case is able to cover some additional statements. This process repeats until the test case queue obtains 100% statement coverage. Finally, τ_0 is set to be the APSC value of T_{H_0}.

The whole CB-ACS framework is illustrated in Fig. 3 and briefly described as follows. First, the n unordered test cases are converted to a graph G with one virtual starting vertex v_{-1} and n real vertices. The CRP heuristic is applied to find the first history-best solution T_{H_0}. After that, an amount of τ_0 pheromone is dropped on every common real and virtual edge in G, with $\tau_0 = \text{APSC}\,(T_{H_0})$, and all the artificial ants are initialized at v_{-1}.

Artificial ants keep transferring to unvisited vertices simultaneously. In every transition, ants calculate the ACB-Heuristic value for every vertex that has not been moved to during this iteration. Utilizing the PRP rule in (2), ants attempt to move to vertices both with a larger additional statement number and belonging to the history-best solution, but with a biased probability to avoid the trap of local optima. According to the LPU rule, all ants deposit pheromone onto G every time they make a transition.

Eventually, all artificial ants terminate at some vertices, where full statement coverage is obtained. After that, the SB-LS mechanism is applied to fine-tune the solutions they construct. Then, the fitness values of these solutions are calculated. The iteration-best fitness value is compared with the history-best one and the GBT tree is updated. At last, the GPU rule utilizing the proposed GBT tree is applied to update the pheromone amount on G. Here, a single iteration is accomplished, and this process is repeated κ times, which is determined as follows:

$$(1-\alpha)^{\kappa} \times \tau_0 = \epsilon \tag{6}$$

where ϵ is a small value close to zero, e.g., 10^{-10}. Equation (6) indicates that the searching process should be terminated

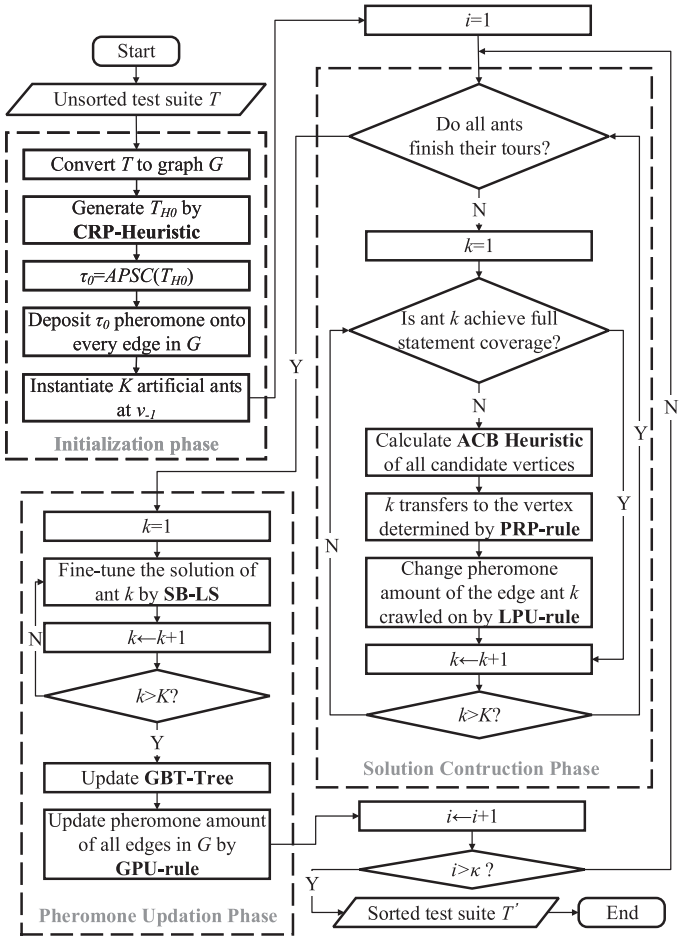


Fig. 3. Flowchart of the proposed CB-ACS framework.

before the pheromone level of those edges that had never been reinforced becomes too weak. Since κ can only take integer values, (6) can be deduced as

$$\kappa = \left\lceil \log_{1-\alpha} \frac{\epsilon}{\tau_0} \right\rceil. \tag{7}$$

After κ iterations, the algorithm stops, and the final history-best solution is regarded as the final solution.

B. ACB-Heuristic Function

The rationale behind the heuristic function in the proposed framework is to utilize the additional coverage information in the additional greedy strategy to improve the rate of statement coverage stepwise. Since the rate of statement coverage is affected only by the moment when statements are first covered, it is those statements which have not been covered yet that we should try to concern and optimize. The additional coverage is the exact measure of the uncovered statement number a test case can cover. Previous empirical studies [3], [10], [20] have also shown that the additional coverage is effective to guide the search for solutions with better rates of statement coverage. Keeping the above in mind, we adopt the additional coverage as the heuristic information in the proposed framework.

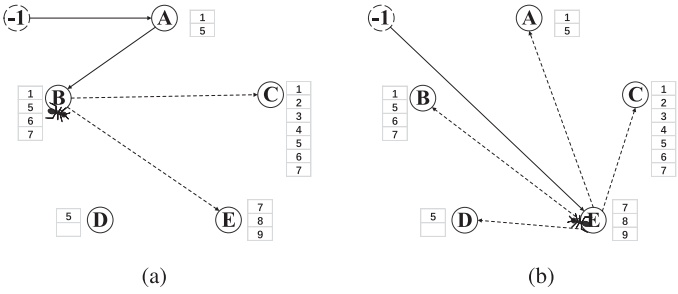


Fig. 4. Examples of two ants' angles of view based on their different partially built tours. The graph corresponds to the test suite in Table I. (a) Ant 1. (b) Ant 2.

Specifically, an artificial ant k in the CB-ACS maintains a covered statement set S_{a_k} of its own, in which the statements covered by itself are recorded. After every transition, S_{a_k} is updated by

$$S_{a_k} \leftarrow S_{a_k} \bigcup (S_{v_s} \setminus S_{a_k}) \tag{8}$$

where S_{v_s} is the covered statement set of the selected vertex s and $\setminus$ computes the difference set of S_{v_s} and S_{a_k}.

The corresponding heuristic function is defined by

$$\eta_k(s) = \left\| (S \setminus S_{a_k}) \bigcap S_{v_s} \right\| \tag{9}$$

where S is the set of all statements and $\|\cdot\|$ returns the number of elements in the input set. $\eta_k(s)$ evaluates how expensive does ant k transfer to s. If any candidate vertex s contains some statements that have never been covered by k during this iteration, k will consider it worthwhile to transfer to s. Otherwise, if S_{a_k} includes all the elements in S_{v_s}, $\eta_k(s)$ becomes zero and k will not move to s. Moreover, a vertex where k will cover more additional statements yields a larger heuristic, while vertices that only provide fewer additional statements for k yield smaller heuristics on the contrary.

Here, the cost measure of an edge from vertex r to s (i.e., the distance) can be deduced from $\eta_k(s)$ as in

$$\delta_k(r,s) = \begin{cases} \dfrac{1}{\eta_k(s)}, & \eta_k(s) \neq 0 \\ \infty, & \text{otherwise} \end{cases}. \tag{10}$$

The distance $\delta_k(r,s)$ of an edge (r,s) is not always fixed. In the CB-ACS, every artificial ant views the Graph G from its perspective, which is based on what and how many statements it has already covered. Therefore, one G may have different shapes in different ants' eyes, with the same vertex set, but different edge connection and distances. Any edge that is considered existed by one ant may become invisible to another ant. Also, any two vertices that were considered connected by an ant may become disconnected after it has further covered a few other statements later.

Fig. 4 illustrates a pair of examples demonstrating this mechanism. The solid lines mark the edges that have been crawled on by the ant, while the dash lines mark edges that are visible but have not been crawled on. Both ants 1 and 2 begin their tours from v_{-1}. Ant 1 in Fig. 4(a) has already visited v_A and

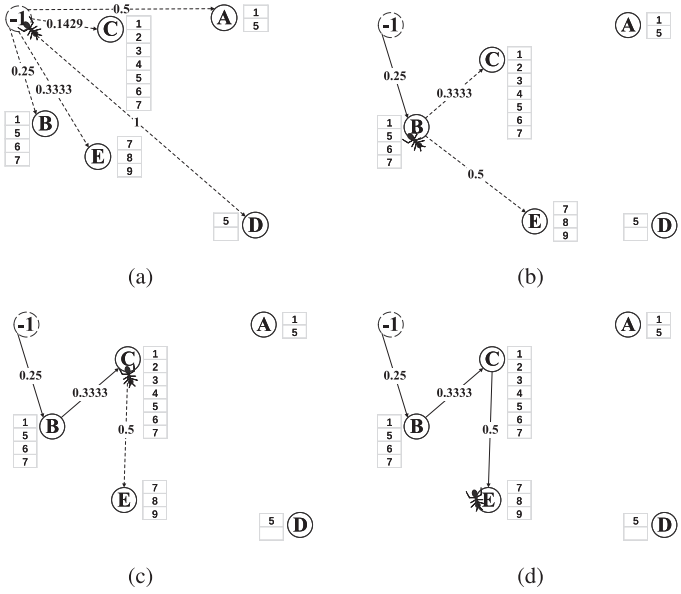


Fig. 5. Entire possible tour of an ant. (a) v_{-1}. (b) $v_{-1} \rightarrow v_B$. (c) $v_{-1} \rightarrow v_B \rightarrow v_C$. (d) $v_{-1} \rightarrow v_B \rightarrow v_C \rightarrow v_E$.

v_B successively and covered statements 1 and 5–7. Located at v_B, therefore, ant 1 considers v_C and v_E reachable because it will be able to earn three and two additional statements there (statements 2–4 for v_C and statements 8 and 9 for v_E). On the contrary, v_D is regarded as unreachable by ant 1, given that the only element in S_{v_D} has already been covered by it. As for ant 2, it has only visited v_E and covered statements 7–9. Hence, all other four vertices are connected and worth transferring to from its perspective.

Fig. 5 illustrates an example of the complete solution construction process. In Fig. 5(a), an ant locates at v_{-1}, and all the other five real vertices are connected to v_{-1}. In Fig. 5(b), the ant selects v_B, and in the meanwhile, the graph G changes in its eyes, with both v_A and v_D turning out to be two isolated vertices and v_C and v_E becoming connected to v_B. In Fig. 5(c), the ant transfers to the nearest vertex which is v_C. Still the full statement coverage condition is not satisfied, so the ant continues to move. In Fig. 5(d), it eventually stopped at the only vertex that remains connected, which is v_E. So far, the full statement coverage constraint is satisfied, and the ant stops.

To summarize, the artificial ants in the CB-ACS are required to remember the following information during their tours: 1) T_k, the ordered sequence of the visited vertices (i.e., the partially built solution); and 2) S_{a_k}, its covered statement set. In the meanwhile, they need to keep updating the connection situation of the graph in their eyes during traveling.

C. Pheromone and GBT Tree

In the traditional ACS, only the history-best solution is kept to guide the ants' later searches. However, there are times when more than one tour yield the same APSC value. In the CB-ACS, all such tours are regarded as *equivalent*, and the ones that yield the greatest historically best APSC values are recorded by the

Algorithm 1: $addBranch$: Adding a Tour to Γ_H.

Input: $T_I^{'}$, length of $T_I^{'}$ L, Γ_H
Output: updated Γ_H
1: $i \leftarrow 1$, $curNode \leftarrow \Gamma_H.root\,()$
2: **while** $i < L$ and $T_I^{'}[i] \in curNode.ChildSet()$ **do**
3: $curNode \leftarrow curNode.Child(T_I^{'}[i])$
4: $i \leftarrow i + 1$
5: **end while**
6: **while** $i < L$ **do**
7: $curNode.addChild\left(T_I^{'}[i]\right)$
8: $curNode \leftarrow curNode.Child\left(T_I^{'}[i]\right)$
9: **end while**

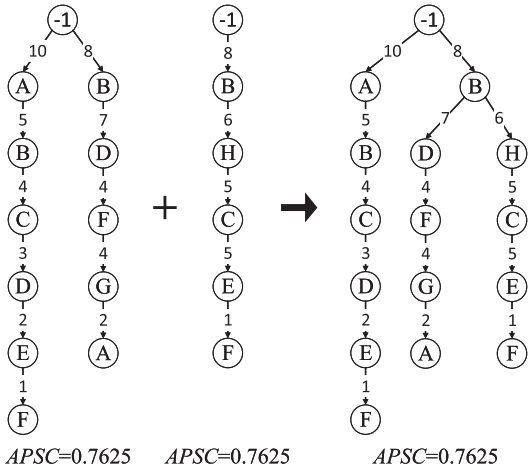


Fig. 6. Example of the $addBranch$ operation of a GBT tree with its corresponding test suite illustrated in Table II.

algorithm. Furthermore, tours that share the exactly same test case orders are referred to as *strictly equivalent*.

In every iteration, after all ants have built their tours, all the iteration-best equivalent tours are selected as the iteration-best tour set $\{T_I\}$, and compared with the history-best tour set $\{T_H\}$. Generally, four base cases may occur.

1) APSC $(\{T_I\}) >$ APSC $(\{T_H\})$.
2) APSC $(\{T_I\}) <$ APSC $(\{T_H\})$.
3) APSC $(\{T_I\}) =$ APSC $(\{T_H\})$, and $\exists T_I^{'} \in \{T_I\}$, $T_I^{'} \notin \{T_H\}$.
4) APSC $(\{T_I\}) =$ APSC $(\{T_H\})$, and $\forall T_I^{'} \in \{T_I\}$, $T_I^{'} \in \{T_H\}$.

In case 1, the original $\{T_H\}$ is cleared and $\{T_I\}$ is assigned to $\{T_H\}$ to become the new history-best tour set, which is expressed in (11). In case 3, the original $\{T_H\}$ absorbs all the tours in $\{T_I\}$ that are not strictly equivalent to any tours in itself, which is expressed in (12). In cases 2 and 4, the original $\{T_H\}$ remains as before

$$\{T_H\} \leftarrow \{T_I\} \tag{11}$$

$$\{T_H\} \leftarrow \{T_I\} \bigcup \{T_H\}. \tag{12}$$

For convenience, the set $\{T_H\}$ is represented in a tree structure, which is referred to as the GBT tree Γ_H. The root of Γ_H is the virtual test case v_{-1}, and each top-down path from the root to a leaf node represents a tour T. A single tour is also considered to be a GBT tree, despite its linear structure, and in most cases, a GBT tree starts expanding itself from a single tour.

The GBT tree is updated during every iteration. In base case 1, the entire GBT tree will be deleted and the new tour becomes the new GBT tree. In base cases 2 and 4, the tree remains the same. In case 3, the new equivalent solution $T_I^{'}$ is added to Γ_H as a new branch by Algorithm 1.

In Algorithm 1, $T_I^{'}[i]$ represents the ith element in tour $T_I^{'}$ (e.g., $T_I^{'}[0]$ is the virtual starting vertex v_{-1}); $curNode.ChildSet()$ returns the children set of the current tree node; $curNode.Child(T_I[i])$ returns the child node of the current tree node with the index $T_I^{'}[i]$; $curNode.addChild(T_I^{'}[i])$ creates a new child node of the current tree node with the index $T_I^{'}[i]$.

To better illustrate the procedure of the algorithm, consider the example in Fig. 6. The number on an edge indicates the additional coverage by including the next tree node (i.e., the next test case). Originally, the tree has already had two branches representing two different solutions both yielding an APSC of 0.7625. Then, a new equivalent solution is found and is added to the GBT tree. Since the front sequence $v_{-1} \xrightarrow{8} B$ of the new solution has already existed in the tree, it is eliminated, and the subsequent part of the solution starting from v_H is connected to the tree.

Theoretically, any tour that is equivalent to but not in the history-best tour set $\{T_H\}$ should be added to the GBT tree. However, in practice, a large number of interequivalent solutions are identical in most parts of their sequences, but only different in a few segments at the back (e.g., $-1 \xrightarrow{5} A \xrightarrow{4} B \xrightarrow{3} C \xrightarrow{2} D \xrightarrow{1} E \xrightarrow{1} F \xrightarrow{1} G \xrightarrow{1} H$ and $-1 \xrightarrow{5} A \xrightarrow{4} B \xrightarrow{3} C \xrightarrow{2} D \xrightarrow{1} E \xrightarrow{1} H \xrightarrow{1} G \xrightarrow{1} F$). Also, these solutions are, in fact, of tiny influence on guiding the ants' searches; on the contrary, they may mislead the ants. Therefore, when it comes to algorithm implementation, we restrict the $addBranch$ operation to be permitted only when the additional coverage of the first different test case in the new solution is higher than $m * 1\%$.

To apply the GBT tree to the pheromone update, the problem-specific pheromone increment $\Delta\tau$ in (5) is defined as follows:

$$\Delta\tau = APSC_h \cdot \gamma_r, (r, s) \in \Gamma_H \tag{13}$$

where r is an internal node in Γ_H, s is one of its children nodes, APSC_h is the fitness value of the history-best solutions, and γ_r is the number of leaf nodes in all the subbranches starting from r. In the proposed method, one edge may be updated more than once because the same (r, s) might appear in different positions of the tree.

D. Local Search

We have proven that in coverage-based TCP problems, any optimal solution always satisfies the following condition (the detailed proof is given in Appendix).

Algorithm 2: Sorting-Based Local Search.

Input: $T^{'}$, a constructed tour; L, length of $T^{'}$;
T, the original test suite.
Output: sorted $T^{''}$

```
Initialize Φ_0 ← ∅
for i = 1 → L − 1 do
    // T'[0] is the virtual starting test case v_{−1}
    for j = L − 1 → i do
        if ||T'_{[j−1]}.Φ|| ≤ ||T'_{[j]}.Φ|| then
            Φ_0 ← (T'_{[j]}.Ψ ⋂ T'_{[j−1]}.Φ)
            T'_{[j]}.Φ ← (T'[j].Φ ⋃ Φ_0)
            T'_{[j−1]}.Φ ← (T'_{[j−1]}.Φ \ Φ_0)
            Swap(T'[j − 1], T'[j])
            // Swap() swaps the value of two variables
        end if
    end for
end for
```

Theorem III.1. Given a permutation $T^{'}$ of a test suite T and assumed that $T^{'}$ is the optimal prioritization solution, then

$$\forall t_i, t_j \in T^{'}, i \neq j,\ \mathrm{Pri}_{T^{'}}(t_i) > \mathrm{Pri}_{T^{'}}(t_j) \Rightarrow \|t_i.\Phi\| > \|t_j.\Phi\|,$$

where $\mathrm{Pri}_{T^{'}}(t_k)$ is the priority of t_k in $T^{'}$ (e.g., $\mathrm{Pri}_{T^{'}}(t_i) > \mathrm{Pri}_{T^{'}}(t_j)$ means t_i is in front of t_j in $T^{'}$), $t_k.\Phi$ is the additional statement set of t_k in $T^{'}$, and $\|\cdot\|$ count the number of elements in the input set.

Theorem III.1 is a sufficient condition for any TCP solution to be optimal, under the goal of rate of code coverage, and it inspires us to design a new local search strategy, which is referred to as the *SB-LS*.

In the CB-ACS, during every iteration when an ant accomplishes tour construction, the solution is sorted in descending order in terms of additional statement coverage. The detailed implementation is demonstrated in Algorithm 2.

In Algorithm 2, $T^{'}_{[k]}$ is the kth test case in solution $T^{'}$, and $T^{'}_{[k]}.\Psi$ is its total covering statement set. SB-LS is of high computational cost. The time complexity in the worst case is $O\left(mL^2\right)$, with m being the total number of statements in the tested program. In practice, however, with the help of the ACB-Heuristic function, in most cases, two adjacent test cases $T^{'}_{j-1}$ and $T^{'}_{j}$ of the solutions satisfy the condition of $\left\|T^{'}_{j-1}.\Phi\right\| > \left\|T^{'}_{j}.\Phi\right\|$, which means that the set operations inside the **if** block in Algorithm 2 seldom take place, and the computation time can, thus, be reduced significantly. Despite its low frequency, we will show in the next section that such local search is of great significance in the optimization activity.

To more clearly illustrate such a mechanism, consider the example demonstrated in Fig. 7. Originally, an artificial ant constructs a solution of $-1 \xrightarrow{10} A \xrightarrow{5} B \xrightarrow{2} E \xrightarrow{2} F \xrightarrow{3} C \xrightarrow{3} H$, of which the test suite is listed in Table II. The number over the arrow in the solution indicates the additional covering statement number obtained by the pointed test case. We discover the additional covering statement number of test case C is not larger than that of H. According to SB-LS, we swap these two cases. It is because if there is no intersection between the additional covering statement sets Φ of C and H, swapping will have no negative effect on the solution. On the contrary, if the intersection does exist, swapping can indeed improve the solution. In this scenario, the additional covering statement sets of C and H are $\Phi_C = \{19, 20, 21\}$ and $\Phi_H = \{16, 17, 18\}$, respectively, and the total covering statement set of H is $\Psi_H = \{7, 16, 17, 18, 19, 20\}$, which means the intersection is $\Phi_0 = \Psi_H \bigcap \Phi_C = \{19, 20\}$. Therefore, after swapping, the number of additional statement covered by H increases to 5 ($\Phi_H \bigcup \Phi_0 = \{16, 17, 18, 19, 20\}$), while that by C reduces to 1 ($\Phi_C \setminus \Phi_0 = \{21\}$). Next, test case H rises again, since 5 is still larger than the number of additional statement by the test case in front of H, which is 2 by F. Similar processes repeat until every pair of two adjacent test cases t_i and t_j with $\mathrm{Pri}_{T^{'}}(t_i) > \mathrm{Pri}_{T^{'}}(t_i)$ satisfies $\Phi_{t_i} \geq \Phi_{t_j}$. In summary, the whole sorting process is essentially the bubble sort technique incorporating with three set operations during every swapping.

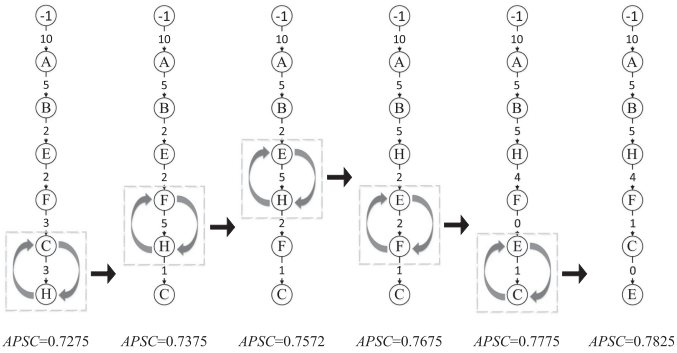


Fig. 7. Example of how the SB-LS mechanism works. The demonstrated solutions correspond to the test suite example in Table II.

IV. Experimental Studies

This section describes the detailed experiments for the coverage-based TCP problems using the CB-ACS.

A. Research Questions

The following three research questions motivated the experiments.

- ***RQ***1. How does the proposed CB-ACS compare with other existing prioritization approaches in terms of APSC?
- ***RQ***2. How do the two components (i.e., the ACB-Heuristic and the SB-LS mechanism) contribute to the final performance of the CB-ACS?
- ***RQ***3. Is the proposed framework truly generic for the coverage-based TCP problems?

Two groups of experiments are designed to evaluate the CB-ACS based on the above questions. The first group is conducted using a set of small- and middle-size problems to answer both

TABLE II
TEST SUITE OF EIGHT TEST CASES COVERING A PROGRAM WITH 25 STATEMENTS

test case	Statements																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	x	x	x	x	x	x	x	x	x	x															
B								x	x	x	x	x	x	x	x										
C					x														x	x	x	x			
D	x	x	x													x	x	x	x						
E	x		x	x																			x	x	
F						x																x	x	x	x
G						x	x													x	x				
H							x									x	x	x	x	x					

TABLE III
TESTED PROGRAMS AND TEST SUITES

Program	Number of statements	Lines of code	Test pool size	Average size of small suites	Average size of large suites
Print_tokens	402	726	4130	16	318
Print_tokens2	483	570	4115	12	388
Replace	516	564	5542	19	353
Space	2748	3815	13585	155	1894

TABLE IV
EXPERIMENTAL RESULTS: AVERAGE APSC (*100%)

P	Test suite	AG	GA	ME-ACO	CB-ACS
Rep.	S	93.9699-	94.0235-	93.1736-	**94.0459**
	L	99.7573-	99.6087-	99.6494-	**99.7621**
Pri.2	S	92.5324-	92.5769-	92.2757-	**92.5836**
	L	99.8199-	99.7082-	99.7852-	**99.8202**
Pri.	S	94.6878-	94.7060-	94.3130-	**94.7131**
	L	99.8375-	99.6781-	99.7905-	**99.8385**
Space	S	95.1268-	94.8025-	92.3504-	**95.2355**
	L	99.8790-	99.3050-	98.8975-	**99.8812**

RQ1 and ***RQ2***. Furthermore, in order to investigate the generality problem in ***RQ3***, the second group of experiments is conducted.

B. Comparison and Components Studies

In this section, experiments are conducted using a set of tested programs that have been widely used in TCP communities [3], [10], [25]. The experiments are separated into two parts. First, the performance of the proposed method is validated by comparing the proposed CB-ACS with several state-of-the-art approaches. After that, the effects of both ACB-Heuristic and SB-LS are studied through the component analysis.

1) Experimental Settings: There are totally four tested programs studied in the experiments, which are as shown in Table III. The first three programs are from an infrastructure referred to as the "Siemens" programs, which have been widely used as benchmarks in coverage-based TCP problems [10], [15], [25]. The last one was designed for the European Space Agency, which is significantly larger than the "Siemens" programs and has been widely used as a practical TCP problem [7], [10], [15]. All these tested programs are available from the Software-Infrastructure-Repository (SIR).[1]

Usually, there are some lines of source code in the source program that can never be covered, e.g., function headers in function definitions and variable declarations. In our experiments, only lines that can be executed by at least one test case in the test pool are considered as statements, and the total number of them is taken as the total statement number m.

The test pools contain all available test cases. To run the experiments, test suites were assembled into two sizes: the small suites and the large suites, following the same approach by Li *et al.* [10]. For each of the programs, 100 small test suites and 100 large test suites were generated.

2) Comparison Studies: The proposed CB-ACS is compared with three other algorithms. The first algorithm is the additional greedy strategies proposed in [10], which has been said to be the most effective deterministic technique [25]. The second one is a stochastic searching approach designed based on the GA [10], and the last one is a recently published ACO-based approach [15]. It should be noted that, to our knowledge, there is no available ACO framework specifically designed for the *coverage-based* TCP problems. Hence, we modified the E-ACO [15] to form an ME-ACO as a competitor. The E-ACO is a recently published multiobjective ACO framework, which has been shown effective for TCP problems. We modified the E-ACO based on [42] and [43] to make it suitable for solving the single-objective *coverage-based* TCP problem. Interested readers can refer to [10] and [15] for details on the setups of the compared algorithms. The parameters of the CB-ACS are set as follows: $\beta = 2$, $q_0 = 0.9$, $\alpha = \rho = 0.1$, and $\epsilon = 10^{-10}$, and the number of ants is 16.

Since the additional greedy strategy is a deterministic approach, every test suite is applied to the tested program once, and the statistical result is the average APSC of 100 test suites of the same tested program under the same size. On the other hand, due to the stochastic searching nature of GA-based and ACO-based approaches, each test suite is applied to the other three algorithms for ten times, and the average results are used for comparison.

The comparison results in terms of the average APSC values (100%) are listed in Table IV, where $+$, $-$, and $*$ indicate that the competitor is significantly better than, worse than, and similar to the CB-ACS under the Wilcoxon test with the significance level $\alpha = 0.05$, respectively.

It can be observed that the APSCs obtained by all approaches in large test suites are higher than those in the small suites. This is because a larger number of test cases (i.e., the value of n) leads to a larger denominator in (1), and thus, the final APSC gets closer to 1. For small test suites of small programs, GA is the

[1][Online]. Available: https://sir.unl.edu/portal/index.php

TABLE V
ORDERINGS FOUND BY DIFFERENT APPROACHES OF SMALL TEST SUITE NO. 40 OF REPLACE

Alg	Solution segment	APSC*100%
AG	-1 $\xrightarrow{166}$ 4964 $\xrightarrow{13}$ 2869 $\xrightarrow{6}$ 5391 $\xrightarrow{5}$ 863 $\xrightarrow{5}$ 4048 $\xrightarrow{3}$ 272 $\xrightarrow{3}$ 1064 $\xrightarrow{2}$ 1294 $\xrightarrow{1}$ 4915 $\xrightarrow{1}$ 920 $\xrightarrow{1}$ 3177 $\xrightarrow{1}$ 5195 $\xrightarrow{1}$ 1797	92.1575
GA	-1 $\xrightarrow{166}$ 4964 $\xrightarrow{13}$ 272 $\xrightarrow{10}$ 863 $\xrightarrow{5}$ 4048 $\xrightarrow{4}$ 5195 $\xrightarrow{4}$ 1064 $\xrightarrow{2}$ 1294 $\xrightarrow{1}$ 3177 $\xrightarrow{1}$ 1797 $\xrightarrow{1}$ 4915 $\xrightarrow{1}$ 920	92.9688
ME-ACO	-1 $\xrightarrow{153}$ 863 $\xrightarrow{33}$ 272 $\xrightarrow{7}$ 4048 $\xrightarrow{4}$ 5195 $\xrightarrow{4}$ 1064 $\xrightarrow{1}$ 920 $\xrightarrow{1}$ 4964 $\xrightarrow{0}$ 2869 $\xrightarrow{2}$ 1294 $\xrightarrow{1}$ 1797 $\xrightarrow{0}$ 3255 $\xrightarrow{1}$ 3177 $\xrightarrow{0}$ 1819 $\xrightarrow{0}$ 5391 $\xrightarrow{1}$ 4915	92.7885
CB-ACS	-1 $\xrightarrow{153}$ 863 $\xrightarrow{33}$ 272 $\xrightarrow{7}$ 4048 $\xrightarrow{4}$ 1064 $\xrightarrow{4}$ 920 $\xrightarrow{2}$ 1294 $\xrightarrow{1}$ 1819 $\xrightarrow{1}$ 3177 $\xrightarrow{1}$ 4915 $\xrightarrow{1}$ 5195 $\xrightarrow{1}$ 1797	93.1190

TABLE VI
ORDERINGS FOUND BY DIFFERENT APPROACHES OF LARGE TEST SUITE NO. 0 OF REPLACE

Alg	Solution segment	APSC*100%
AG	-1 $\xrightarrow{184}$ 2370 $\xrightarrow{7}$ 5371 $\xrightarrow{4}$ 223 $\xrightarrow{3}$ 3536 $\xrightarrow{2}$ 436 $\xrightarrow{2}$ 169 $\xrightarrow{2}$ 1288 $\xrightarrow{1}$ 4068 $\xrightarrow{1}$ 2823 $\xrightarrow{1}$ 3177 $\xrightarrow{1}$ 5053	99.6913
GA	-1 $\xrightarrow{139}$ 2805 $\xrightarrow{34}$ 1949 $\xrightarrow{11}$ 4893 $\xrightarrow{5}$ 3968 $\xrightarrow{4}$ 1100 $\xrightarrow{0}$ 1420 $\xrightarrow{4}$ 4321 $\xrightarrow{1}$ 5091 $\xrightarrow{2}$ 327 $\xrightarrow{0}$ 147 $\xrightarrow{3}$ 223 $\xrightarrow{0}$ 3357 $\xrightarrow{1}$ 3728 $\xrightarrow{0}$ 4720 $\xrightarrow{0}$ 2552 $\xrightarrow{0}$ 4749 $\xrightarrow{0}$ 4160 $\xrightarrow{1}$ 5265 $\xrightarrow{0}$ 1679 $\xrightarrow{0}$ 3059 $\xrightarrow{1}$ 2819 $\xrightarrow{0}$ 3583 $\xrightarrow{0}$ 3416 $\xrightarrow{0}$ 2123 $\xrightarrow{0}$ 1194 $\xrightarrow{0}$ 3582 $\xrightarrow{0}$ 3110 $\xrightarrow{0}$ 2757 $\xrightarrow{0}$ 2657 $\xrightarrow{0}$ 1999 $\xrightarrow{0}$ 5532 $\xrightarrow{2}$ 1288	99.3906
ME-ACO	-1 $\xrightarrow{184}$ 2370 $\xrightarrow{2}$ 1288 $\xrightarrow{3}$ 5484 $\xrightarrow{2}$ 273 $\xrightarrow{4}$ 5053 $\xrightarrow{1}$ 826 $\xrightarrow{0}$ 1204 $\xrightarrow{4}$ 647 $\xrightarrow{0}$ 4773 $\xrightarrow{0}$ 4810 $\xrightarrow{0}$ 758 $\xrightarrow{2}$ 920 $\xrightarrow{1}$ 4068 $\xrightarrow{1}$ 3029 $\xrightarrow{0}$ 3651 $\xrightarrow{3}$ 5396 $\xrightarrow{1}$ 2823	99.5578
CB-ACS	-1 $\xrightarrow{184}$ 2370 $\xrightarrow{7}$ 5371 $\xrightarrow{4}$ 5053 $\xrightarrow{3}$ 4321 $\xrightarrow{3}$ 436 $\xrightarrow{2}$ 1288 $\xrightarrow{2}$ 2150 $\xrightarrow{1}$ 454 $\xrightarrow{1}$ 805 $\xrightarrow{1}$ 2823	99.7009

second best, slightly outperforming the widely used additional greedy (AG) technique. But when it comes to large suites of small programs, or to the large program *space*, the bottleneck of the GA is revealed: It performed worse than both CB-ACS and AG in *space*, and even worse than ME-ACO in large suites of small programs. AG performed worse than both the GA and the CB-ACS in small suites of small programs, but obtained the second best performance in all other cases. This suggests that although in small problems the greedy strategy might not be a good choice, in large problems (e.g., large suites of small programs or large programs), this deterministic approach is still capable of producing promising solutions. ME-ACO obtained the worst performance in both small suites of small programs and the large program *space*, while outperformed the GA in large suites of small programs. The proposed CB-ACS always performed significantly better than the other three approaches on all tested programs.

To better understand the differences among the performance of these four techniques, we illustrate two examples of their corresponding built solutions in Tables V and VI. Numbers pointed by the right arrows are the indices of test cases in the original test pool, while numbers over the arrows are the corresponding additionally covering statement numbers of the right-hand-side test cases. Although only CB-ACS introduces the virtual starting test case mechanism, we still mark it at the front of every solution for convenience and consistency. The solution segments in the two tables are from the first test case of the original solution to the last one, where full statement coverage is obtained. It can be observed that test case No. 4964 and No. 2370 are the two cases with the largest covering statement number in Tables V and VI, since they are the first test cases in the solution produced by AG. In Table V, both ME-ACO and CB-ACS suggested a different choice, which is No. 863. Although being not able to cover as many statements as No. 4964, No. 863 allows a larger further additional coverage in the next two steps and finally leads to higher APSCs. This proves that being less greedy in some cases does benefit to the overall situation, and that the ACO frameworks are capable of jumping out of local optima.

The GA has quite different performance in Tables V and VI. In Table V, the GA performed the second best, while in Table VI, it became the worst one. Owning to the stochastic searching nature, the GA is capable of exploring a large portion of the whole search space when the problem scale is small, which eventually leads to a promising APSC value. Compared to the GA, ACO-based techniques seem to be more effective in producing better solutions. In Table V, the solutions of ME-ACO and CB-ACS are equivalent in the first five test cases (i.e., their additional covering statement numbers are 153, 33, 7, 4, and 4 respectively), which means artificial ants in the two ACO frameworks knew how to avoid local optima. In CB-ACS, the difference is the additional coverage of the subsequent test cases that are strictly in descending order owing to the ACB-Heuristic function as well as the SB-LS mechanism, while in ME-ACO, artificial ants selected some useless test cases (e.g., $\xrightarrow{0}$ 2869 in Table V) and failed to bring forward test cases with relatively large additional coverage (e.g., $\xrightarrow{2}$ 1294). Overall, the above results indicate that the proposed CB-ACS has the strongest global search ability.

3) Component Analyses of the CB-ACS: In the CB-ACS, we design two key components, which are the ACB-Heuristic and the SB-LS. To investigate the effectiveness of them, two simplified CB-ACS variants are generated for comparison, which are CB-ACS/H and CB-ACS/L. Specifically, to understand the effectiveness of the heuristic function, CB-ACS/H is created by replacing the ACB-Heuristic with the following function:

$$\eta(r, s) = |S_s \setminus S_r| . \tag{14}$$

To evaluate the effectiveness of the local search mechanism, we remove it from the CB-ACS framework to produce CB-ACS/L. The other settings of the three algorithms are identical. Without loss of generality, the component analyses are conducted by testing CB-ACS and its two variants on ten small test suites

TABLE VII
COMPONENT ANALYSES RESULTS UNDER *space*: AVERAGE APSC (*100%)

TS	CB-ACS	CB-ACS/H	CB-ACS/L	AG
0	**95.0897**	94.7671-	95.0436-	94.7756-
1	**94.8930**	94.6624-	94.8801-	94.8604-
2	**95.2568**	95.0121-	95.2330-	95.1561-
3	**95.5014**	95.2680-	95.4410-	95.4108-
4	**95.3117**	95.1905-	95.2951-	95.1831-
5	**94.8204**	94.6174-	94.8041-	94.8026-
6	**95.3732**	95.1206-	95.3119-	95.2740-
7	**94.9478**	94.7276-	94.9326-	94.7324-
8	**95.3256**	95.0275-	95.2694-	95.2589-
9	**95.0094**	94.8567-	95.0077-	94.9959-
AVG	**95.1529**	94.9450-	95.1218-	95.0450-

from the *space* program (i.e., No. 0 to No. 9). The experiments are repeated ten times. The average APSC values (100%) are demonstrated in Table VII. Moreover, ten graphs about the evolutionary curves of the best fitness values are drawn in Fig. 8.

It can be observed that CB-ACS/H performed the worst, achieving the lowest APSC values in all the ten experimented test suites. Such dissatisfaction results from that the heuristic function in CB-ACS/H considers only the additional coverage between two adjacent test cases r and s, which is always a fixed value. This may mislead the ants' searches. For example, if an edge (r, s) yields a larger heuristic value, the test case s on the other end of the edge possesses more additional statements and is more likely to be transferred to. However, the truth might be the opposite, since it is possible that many additional statements covered by s may have already been covered by the ant. On the other hand, the proposed heuristic function of CB-ACS can avoid this drawback. By considering the statements covered by the candidate vertices and the ants themselves, the proposed heuristic provides the actual number of additional statements, which are covered by the candidate vertices, but not covered by the ants. Hence, the proposed heuristic is more effective to guide the ants to search better solutions. The results of CB-ACS and CB-ACS/H in Table VII and Fig. 8 have demonstrated the effectiveness of the proposed heuristic.

The results in Table VII show that CB-ACS/L performed significantly worse than the CB-ACS. Reflected from the ten fitness curves in Fig. 8, CB-ACS/L required more iterations to achieve the same APSC values as CB-ACS. Notice that the proposed SB-LS serves as a fine-tuning approach. By iteratively swapping every pair of two adjacent test cases in the solution if the former covers no more additional statement than the latter, and updating their corresponding additional covering statement set after every swap, SB-LS is able to discover solutions, which might have a potential to outperform the history-best solutions after a few swapping operations. To summarize, the results in Table VII and Fig. 8 show that the proposed local search is effective to improve the search efficacy.

To summarize, among the two components, the ACB-Heuristic function is the most powerful. Without the proposed heuristic, the proposed CB-ACS would perform worse than the state-of-the-art additional greedy strategy. The SB-LS mechanism diversifies the search and accelerates the converge speed.

C. Investigating the Generality of the CB-ACS

In this section, whether the proposed CB-ACS is generic enough for practical use is investigated.

Generality is an important issue when verifying the value-in-use of any proposed techniques. Thanedar *et al.* [44] stressed the applicability problem of optimization algorithms to a wide range of complex scenarios. They defined five necessary considerations, which are robustness, function form irrelevance, accuracy, ease of use, and efficiency. In this article, the generality refers to that the observations found in the experiments can be reflective of other situations, and that the proposed framework can perform as well on projects other than those it was evaluated on [45], which is close to the applicability above.

In the community of software engineering, Nagappan *et al.* [45] regarded generality as one of the many goals for research. They highlighted the importance of diversity as one aspect of generality, which can be expressed as a sufficiently large sampling to contain members from different subgroups of the whole population of subjects. Basili *et al.* [46] remarked on the difficulties of drawing conclusions that are general enough from empirical studies in software engineering, due to a large number of relevant context variables.

In the fields of optimization and evolutionary computation, generality is also of great significance. Miller [47] suggested that engineers and hard scientists demonstrate the generality of models for important processes usually by applying them to various practical problems. Tanev *et al.* [48] defined generality in their context as the deteriorating extent of the agents' performance when faced with unknown situations. Bibai *et al.* [49] stressed the problems for any evolutionary algorithms to attain good performance in other problems as in the ones they have been designed and fine-tuned for.

To quantitatively assess generality as well as to better clarify its concept for the proposed framework, we detailed it into four different dimensions, which are listed as follows.

1) *Scalability* [50]–[52]: A generic framework should be adaptable not only to benchmark problems, but also to real-world programs with larger scale.
2) *Granularity* [10], [20], [53]: A generic framework should be generalized to multiple levels of code granularity (e.g., statement, branch, function).
3) *Efficiency* [52], [54], [55]: A generic framework should have acceptable computational time cost to be applied to real-word applications.
4) *Effectiveness* [3], [56], [57]: A generic framework should also have promising fault revealing capability in addition to code coverage capability.

The rationale behind this division is to match each of these requirements to the generality factors mentioned above. Among them, the scalability corresponds to the demand of a diverse sampling in [45]. Both the granularity and the effectiveness concern with the good performance beyond the intended problem [49]. Also, the efficiency is consistent with the inherent requirement of applicability in [44]. By this means, the generality is expected to be examined reasonably and comprehensively.

1) Experimental Design: To investigate the effectiveness of the proposed framework on large-scale problems, a new problem

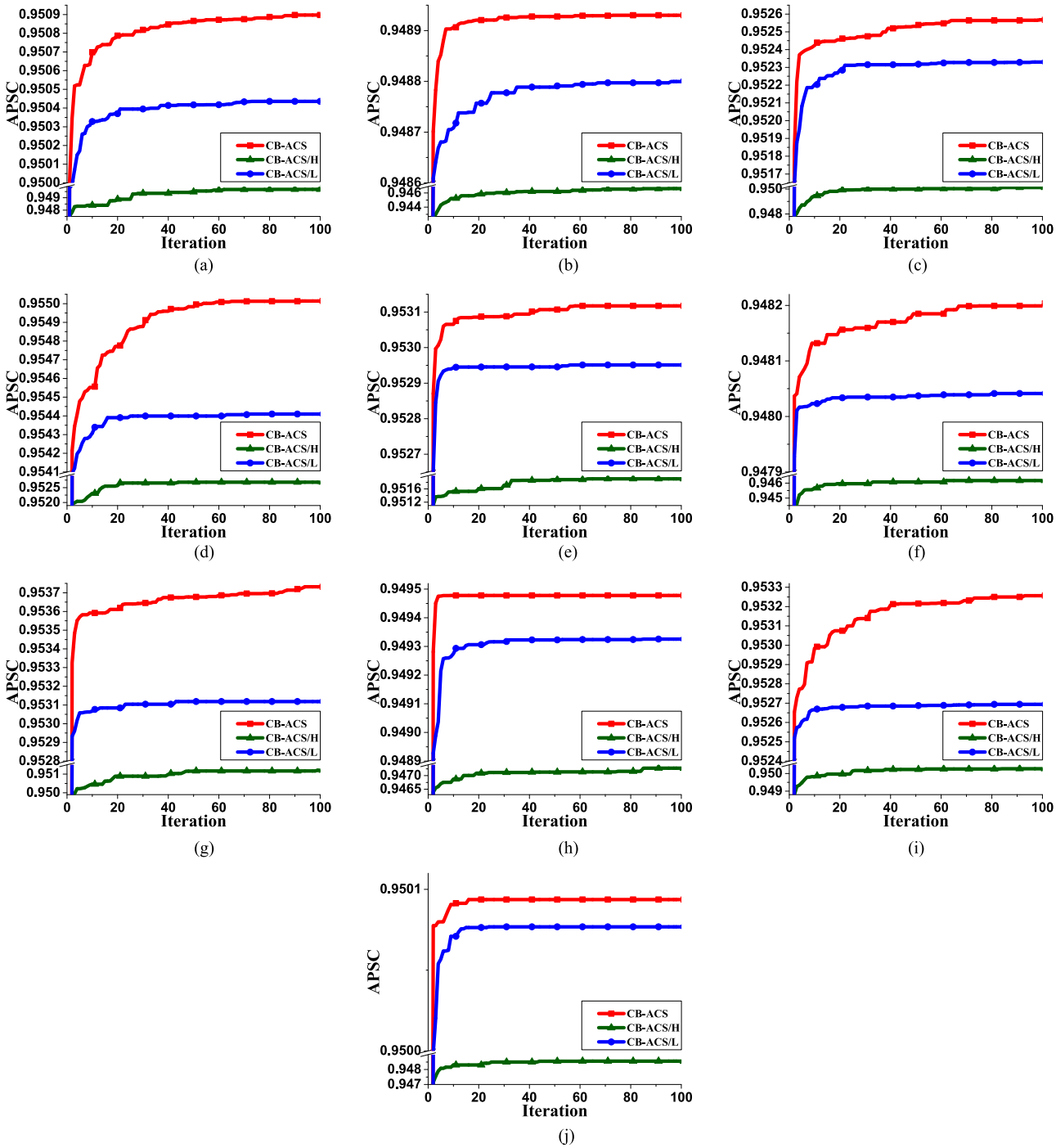


Fig. 8. Fitness curves of the history-best solutions of CB-ACS, CB-ACS/H, and CB-ACS/L. (a) Test suite 0. (b) Test suite 1. (c) Test suite 2. (d) Test suite 3. (e) Test suite 4. (f) Test suite 5. (g) Test suite 6. (h) Test suite 7. (i) Test suite 8. (j) Test suite 9.

set consisting of practical programs is needed. In this experiment, five programs adopted as the experimental subjects are reported in Table VIII. Integrated in the Defects4J framework [58], they are all open-source java projects and have been widely used in the TCP research [50], [51], [59], [60]. Different from the SIR subjects used in Section IV-B, where the faults are seeded by the authors of the infrastructures (i.e., the faults are inserted intentionally), faults in the Defects4J programs are real (i.e., they

TABLE VIII
INFORMATION OF THE TESTED PROGRAMS

Program	# stmts	# brans	# funcs	# lines	# TCs	# faults
Chart	58688	13912	4563	96382	356	26
Closure	76174	18400	7573	90697	221	101
Lang	22650	6490	1477	21787	113	39
Math	73870	15294	4916	84323	385	7
Time	17590	3520	2171	27801	123	27

are inserted by the developers unintentionally). By instincts, the second kind of faults should be more consistent with manufacturing. The detailed setups of acquiring the corresponding artifacts (i.e., the coverage traces, the test cases, and the fault statistics) accord with [51]. Due to the different organization of the Defects4J subjects, no division of test cases into test suites was available in this experiment. Instead, all the test cases were combined to one single test suite, and the studied approaches were each applied ten times to capture typical performance. The other approaches adopted for comparison are the same as those in Section IV-B with identical setups.

In order to validate the generalization of the proposed framework for other code granularity levels, and to measure the effectiveness of the CB-ACS in terms of fault exposure, new metrics are required, which are introduced as follows.

1) *Average percentage faults detected (APFD):* This measures how fast a prioritized test suite reveals the faults.
2) *Average percentage of branch coverage (APBC):* This measures how fast a prioritized test suite covers the branches (i.e., decisions).
3) *Average percentage of function coverage (APFC):* This measures how fast a prioritized test suite covers the functions.

Normally, for the code-based criterion (i.e., APSC, APBC, and APFC), the coverage information of test cases can be determined as soon as the program instrumentation. However, APFD evaluates the rate of fault detection, which requires prior knowledge of the faults every test case that is able to expose. Since TCP occurs in advance of the testing activities, the APFD metric cannot be adopted as a fitness function to guide the search of evolutionary algorithms, but serves as a measurement of how well the coverage criteria under different levels code granularity had approximated the fault exposure.

The branch coverage and function coverage are similar in the form to the statement coverage. Functions refer to all the user-defined functions invoked during the program execution. Branches refer to different code execution path generated by logical statements such as *if* and *while*, and switched by boolean conditions.

The definitions of APBC, APFC, and APFD are also similar to APSC, expect that TS_i in (1) is replaced by TB_i, TFU_i and TFA_i respectively, representing the index of the first test case covering branch (function) i or detecting fault i. For instance, the APFD metric is defined as

$$\text{APFD}\left(T^{'}\right) = 1 - \frac{\sum_{i=1}^{m} TFA_i}{mn} + \frac{1}{2n}. \tag{15}$$

While the unselected test cases after full code coverage are added to the back of the solution in previous sections, they are no longer trivial when regarding the rate of fault detection. This is because a hidden fault cannot always be triggered by executing the corresponding statement as few as once, and there might be some faults only detectable by some specific test cases, no matter how few additional code coverage they possess.

Among the four studied methods, CB-ACS, AG, and ME-ACO suffer from these unselected test cases. This is because the first two terminate themselves when full coverage is obtained, while the last one utilizes the Epistasis mechanism to prioritize some of the test cases. In order to compare the fault exposure abilities fairly, the additional greedy strategy is applied to iteratively sort the unselected test cases, until no remaining test cases left.

TABLE IX
AVERAGE APXC (*100%) UNDER DIFFERENT LEVELS OF CODE GRANULARITY

program	AG	GA	ME-ACO	CB-ACS
APBC				
Chart	90.1299-	85.7210-	83.0015-	**90.1397**
Closure	**96.3564***	94.6026-	93.6052-	**96.3564**
Lang	**88.1136***	86.9940-	84.7674-	**88.1136**
Math	91.0770-	86.6214-	83.5438-	**91.1409**
Time	91.5574-	90.6859-	89.0068-	**91.6524**
APFC				
Chart	88.4112-	84.0122-	82.1289-	**88.4254**
Closure	97.2074*	95.6541-	94.3475-	**97.2085**
Lang	81.3698-	80.4112-	74.4116-	**81.4036**
Math	89.7230-	85.1298-	81.2600-	**89.9296**
Time	90.4416*	89.5638-	86.7251-	**90.4496**
APSC				
Chart	88.4147-	83.7865-	82.5734-	**88.4211**
Closure	**95.2826***	93.3726-	92.2254-	**95.2826**
Lang	**85.0090***	84.0487-	80.2875-	**85.0090**
Math	89.8883-	85.4097-	82.6040-	**89.9295**
Time	**89.4131***	88.4930-	86.4247-	**89.4131**

To determine the efficiency of the proposed CB-ACS, the time cost for running the algorithms is recorded. Evolutionary algorithms possess strong capability of parallel computation, and the efficiency comparison only makes sense when all the approaches achieve their maximal processing abilities. Therefore, the other three approaches expect AG were all implemented in a multithread manner in order to fully utilize the computation resource of the CPU.

Without loss of generality, the average time cost is calculated over ten times of repetitions, and all the experiments were performed on Intel Core i7-8700 with fixed 3.20 GHz, six cores, 12-thread CPU, 16-GB RAM, running Windows 10 64-bit operating system.

2) Scalability Results: The comparison results in terms of the average APBC, APFC, and APSC values (100%) are listed in Table IX, where $+$, $-$, and $*$ indicate that the competitors are significantly better, worse, and similar to CB-ACS, respectively, under the Wilcoxon test with the significance level $\alpha = 0.05$.

Different from the results in Table IV, where the "Siemens" programs were studied and all the APSC values are as high as 90% or above, the objective values obtained by all the methods in this experiment reduced by a significant margin. In practice, a single test case is rarely able to cover a large number of structural elements due to the growth of the programs' sizes. In the meanwhile, any small increase in the objective APxC(here x = S/B/F) values means larger advancement in the coverage moment of structural elements when compared with the benchmark cases. The above two reasons make it important to prioritize test cases wisely when solving problems from the real-world applications.

All the APxC values obtained by CB-ACS are significantly larger than those by the other two EA approaches (i.e., GA and ME-ACO). Lacking the help of the problem-specific coverage information, the performance gap between the GA and the

TABLE X
AVERAGE RUNNING TIME (S) UNDER DIFFERENT LEVELS OF CODE GRANULARITY

P	AG	GA	ME-ACO	CB-ACS
APBC				
Chart	0.139	34.5	8.89	55.1
Closure	0.189	29.2	5.37	50.8
Lang	0.0503	4.80	0.65	2.98
Math	0.126	40.1	7.07	37.9
Time	0.0590	2.99	1.14	8.92
APFC				
Chart	0.0991	11.7	5.66	21.9
Closure	0.136	12.6	4.08	24.5
Lang	0.0460	1.15	0.604	1.15
Math	0.0931	13.5	6.40	15.6
Time	0.0574	2.12	0.97	7.04
APSC				
Chart	0.418	143	14.4	273
Closure	0.812	118	22.7	379
Lang	0.0662	17.1	1.16	16.0
Math	0.480	174	17.2	277
Time	0.192	15.8	5.67	73.0

TABLE XI
AVERAGE APXC (*100%) OF THE THREE EAS WITH RESPECT TO THE SAME RUNNING TIME LIMIT

program	GA		ME-ACO		CB-ACS
	APxC	gens	APxC	iters	APxC
APBC					
Chart	87.3515	486	83.2814	1077	**90.1397**
Closure	95.3204	530	93.5431	459	**96.3564**
Lang	84.0892	121	85.8431	11952	**88.1136**
Math	86.2888	275	83.8047	778	**91.1409**
Time	91.3939	968	89.0794	364	**91.6524**
APFC					
Chart	85.9857	538	83.2965	15238	**88.4254**
Closure	96.5740	603	94.5828	7317	**97.2085**
Lang	80.2015	260	77.0063	202479	**81.4036**
Math	85.4628	336	83.2487	14341	**89.9296**
Time	90.3641	1142	87.3887	7248	**90.4496**
APSC					
Chart	86.1717	571	82.7854	1369	**88.4211**
Closure	94.8263	983	92.3852	642	**95.2826**
Lang	83.6529	281	80.9244	13917	**85.0090**
Math	87.0776	509	82.6670	892	**89.9295**
Time	89.3565	1563	86.4925	524	**89.4131**

CB-ACS is widened compared to that in the benchmarks. In both designed based on the ACO framework, ME-ACO performed the worst one among the four methods, while the latter performed the best. These results emphasize the importance of domain knowledge and the proper design of the heuristic function especially for large-scale real-world problems.

Although attaining identical APxC values in five out of 15 cases, the CB-ACS still performed better than AG in eight cases among the others. This indicates that the CB-ACS is able to prevent the nearsightedness of AG and find better solutions.

3) Granularity Results: The results in Table IX also exhibit strong adaptation of the CB-ACS for coverage-based TCP problems under different levels of code granularity. This is because the additional coverage information can be applied to different forms of code entities. This granularity independence characteristic provides the heuristic function with great flexibility. Meanwhile, Theorem III.1 is also satisfied with respect to both branch and function coverage, making it possible to apply the effective SB-LS mechanism to all the code entities. To summarize, the proposed CB-ACS can be generalized to different levels of code granularity with promising effectiveness.

4) Efficiency Results: Table X records the time cost of the four algorithms under 15 cases. Being a deterministic approach, which constructs solutions by being greedy stepwise, AG is the fastest, obtaining the computational time less than 1 s in all the cases. Due to the iteration process and a number of individuals (GA) or agents (ACO) to maintain, the other three EA-based approaches are significantly slower. Among them, the CB-ACS is slower than ME-ACO in all the cases, while slower than GA in 11 cases. It can be concluded that the CB-ACS possesses no advantage with respect to computational time.

The time cost of the CB-ACS mainly results from the calculation of the heuristic function. ME-ACO adopts the total coverage of test cases, which can be predetermined as the heuristic information. The CB-ACS, on the other hand, requires $O(m)$ time to determine the heuristic value of a single test case for one artificial ant. Therefore, in order to make one transition, an ant would have to take $O(mn)$ time to evaluate the additional coverage information of all the candidates, while ME-ACO costs only $O(n)$ time. In addition, the SB-LS mechanism is a unique component of the CB-ACS, which takes $O\left(mL^2\right)$ time to fine-tune a single solution.

It is worth mentioning that in the algorithm implementation, a feasible optimization has been made to reduce the time complexity of the CB-ACS. For every artificial ant, a copy of the graph Gc is maintained in its data structure. Every time the ant makes a transition, the covered statements in the statement set S_v of the unvisited test cases are removed. And the additional coverage information is regarded as the remaining size of S_v. By this means, the time complexity of an entire solution construction in one iteration can be reduced to $O\left(mL^2\right)$ from $O(mnL)$, with $L < n$ in most cases.

To investigate whether CB-ACS can still outperform the other two EAs given the same computation time, a fixed-time comparison is conducted. Instead of restricting their maximal iteration times, both the GA and the CB-ACS are allowed to search within the time interval taken by the CB-ACS. The results are demonstrated in Table XI. Compared to Table IX, there are improvements in the performance of both the two competitors. However, they are still worse than the proposed CB-ACS. These results indicate that the proposed heuristic function and the local search mechanism are effective to improve the search efficiency.

The CB-ACS sacrifices a few computational time in the pursuit of better effectiveness and remains high searching efficiency. In view of the large amount of time for performing regression testings (e.g., seven weeks as reported by [6]), the hundreds of seconds consumed by the CB-ACS are still in the acceptable range.

5) Effectiveness Results: Table XII demonstrates the fault exposure results corresponding to Table IX. Both CB-ACS and AG achieved the highest in four out of 15 cases, while ME-ACO achieved the highest in five out of 15 cases. Based on the APFD values, the CB-ACS performed better than AG in eight cases,

TABLE XII
AVERAGE APFD (*100%) AND THE CORRESPONDING VARIANCE

P	code entity	AG	GA	ME-ACO	CB-ACS
Chart	bran	73.2069	68.9989	61.4930	**73.2849**
	func	**68.4290**	65.6284	62.2589	68.3608
	stmt	**71.2784**	68.0347	61.3499	69.8137
Closure	bran	46.6077	51.5988	**53.9513**	50.2637
	func	50.7825	48.4756	**51.4071**	50.1487
	stmt	43.4731	47.3943	**51.1854**	43.2911
Lang	bran	58.2634	**58.3054**	51.9347	58.2657
	func	56.7716	57.2098	51.1958	**57.2424**
	stmt	**60.9441**	58.2261	54.5524	60.8928
Math	bran	**75.3162**	72.8609	63.2738	74.8409
	func	69.4010	67.7083	53.6644	**70.5692**
	stmt	69.9212	62.7865	59.0625	**70.6101**
Time	bran	57.0886	**59.9150**	58.4487	58.8767
	func	52.9903	51.9429	**55.0334**	53.0540
	stmt	54.6903	53.7917	**55.6861**	54.5993

performed better than GA in ten cases, and performed better than ME-ACO in ten cases. These results demonstrate that the proposed CB-ACS is capable of achieving very promising performance in terms of APFD.

V. THREATS TO VALIDITY

In this section, some possible threats to the validity of the experiments are discussed.

Internal validity concerns irrelevant factors that could have influenced the observed experimental results. First, the other two EAs were originally intended for different studied programs (i.e., "Siemens" programs for GA, and three larger SIR programs for ME-ACO). However, we followed the same parameter instructions when applying them to other programs (e.g., the Defects4J programs to both GA and ME-ACO). The optimal settings of their parameters were, thus, not guaranteed, which might have affected the comparison. Second, the inconsistency of the time assessment might have affected the validity of efficiency. To mitigate this threat, we have: 1) implemented all the methods in the same programming language; 2) reduced the iteration layers in nested loops in algorithm implementations to avoid the irrelevant growth of time complexity; and 3) conducted the experiments in the same machine.

External validity concerns the generalization of the results beyond the experimental environment. To eliminate the effects of programming languages of the studied programs to their test cases' coverage, we have adopted programs written in different languages (i.e., C and Java). However, the validity for programs in other languages (e.g., Python and C#) cannot be guaranteed. Also, the assembled test suites in Section IV-B and the untreated test suites in Section IV-C might have suggested specific distribution of test cases, which is not generic and practical enough.

Construct validity concerns the correctness of the measurement metrics under specific study purposes. In this article, they are mainly related to the choices of metrics when verifying the effectiveness of the proposed TCP technique. To this end, we have adopted the rates of code coverage under three different levels of granularity, the time cost, and the rate of fault detected to measure the effectiveness comprehensively. Moreover, the APSC, APBC, APFC, and APFD metrics we adopted have been widely accepted in the literature [10], [15], [52], [56], [57], and the confidence of the observed results can, thus, be guaranteed.

VI. CONCLUSION

This article proposed a new ACS framework to solve the TCP problem with an objective to maximize the rate of statement coverage. To enhance the searching ability of the artificial ants, an ACB-Heuristic function was defined. An effective SB-LS mechanism was also proposed to fine-tune solutions found by artificial ants. Experiments on benchmark problems confirmed that the proposed CB-ACS can achieve faster rate of statement coverage than other state-of-the-art techniques. Experiments on real-world programs confirmed that the proposed CB-ACS possesses promising generality.

As for future works, one promising direction will be to extend the proposed framework to solve multiobjective TCP problems. Attempts have been made to apply multiobjective EAs to TCP problems [7], [15], [61], [62], [63]. Contrasting criteria such as the code coverage, the EET of test cases, the fault history information for version specific regression testing, and the severity of the faults can be jointly considered in a multiobjective manner. Different heuristic functions can be designed, each regarding an independent objective. Then, the different heuristics are hybridized (e.g., by multiplication) to guide the algorithm to search for a set of nondominated solutions that have different tradeoffs among the objectives.

APPENDIX
PROOF OF THEOREM III.1

Proof. This proof is given by contradiction.

Suppose the following.

1) The optimal prioritization solution for T under the goal of maximizing its rate of statement coverage is T_{opt}.
2) $t_i (0 \leq i < n)$ is the ith test case in T_{opt}.
3) S_i is the set of covered statement of t_i.
4) C_i is the set of additional covering statement by executing t_i in $T^{'}$.
5) c_i is the additional statement gain of t_i, i.e., $c_i = |C_i|$.

Assume that $T_{\text{opt}} = -1 \xrightarrow{c_0} t_0 \xrightarrow{c_1} t_1 \xrightarrow{c_2} t_2 \xrightarrow{c_3} \cdots \xrightarrow{c_k} t_k \xrightarrow{c_{k+1}} t_{k+1} \xrightarrow{c_{k+2}} \cdots \xrightarrow{c_{n-1}} t_{n-1}$, where $0 < c_k < c_{k+1}$.

And the fitness value of T_{opt} is

$$\text{APSC}\left(T_{\text{opt}}\right) = 1 - \frac{\sum_{i=0}^{n-1}(i+1)\,c_i}{m \times n} + \frac{1}{2n}.$$

If t_k is swapped with t_{k+1} to produce $T^{'}_{\text{opt}}$, i.e., $T^{'}_{\text{opt}} = -1 \xrightarrow{c_0} t_0 \xrightarrow{c_1} t_1 \xrightarrow{c_2} t_2 \xrightarrow{c_3} \cdots \xrightarrow{c^{'}_k} t_{k+1} \xrightarrow{c^{'}_{k+1}} t_k \xrightarrow{c_{k+2}} \cdots \xrightarrow{c_{n-1}} t_{n-1}$, and

$$\text{APSC}\left(T^{'}_{\text{opt}}\right) = 1 - \frac{\sum_{i=0}^{n-1}(i+1)\,c^{'}_i}{m \times n} + \frac{1}{2n}$$

then according to the assumption, we have

$$
\begin{aligned}
&.\text{APSC}(T_{\text{opt}}) \geq \text{APSC}(T_{\text{opt}}^{'}) \\
&\Rightarrow \text{APSC}\left(T_{\text{opt}}\right) - \text{APSC}\left(T_{\text{opt}}^{'}\right) \geq 0 \\
&\because (\forall 0 \leq i < n, i \neq k, k+1)\left[c_i = c_i^{'}\right] \\
&\Rightarrow \frac{\sum_{l=k}^{k+1}(l+1)\left(c_l^{'} - c_l\right)}{m \times n} \geq 0. \qquad (*)
\end{aligned}
$$

However, we have

$$
\begin{aligned}
c_k &= |C_k|, c_{k+1} = |C_{k+1}| \\
c_k^{'} &= |C_{k+1} \cup (S_{k+1} \cap C_k)| \\
c_{k+1}^{'} &= |C_k \setminus (S_{k+1} \cap C_k)|.
\end{aligned}
$$

Suppose that $x = |S_{k+1} \cap C_k| \geq 0$

$$
\begin{aligned}
&\because (\forall 0 \leq i, j < n, i \neq j)\,[C_i \cap C_j = \emptyset] \\
&\therefore c_k^{'} = c_{k+1} + x, c_{k+1}^{'} = c_k - x \qquad (**) \\
&\therefore c_k^{'} \geq c_{k+1} > c_k \geq c_{k+1}^{'}.
\end{aligned}
$$

Substituting (**) into the left-hand side of (*), we obtain

$$
\frac{c_k - c_{k+1} - x}{m \times n} < 0
$$

which violates the assumption, and Theorem III.1 is proved. ■

REFERENCES

[1] E. Rogstad and L. C. Briand, "Clustering deviations for black box regression testing of database applications," *IEEE Trans. Rel.*, vol. 65, no. 1, pp. 4–18, Mar. 2016.

[2] Y. Le Traon, T. Jeron, J. Jezequel, and P. Morel, "Efficient object-oriented integration and regression testing," *IEEE Trans. Rel.*, vol. 49, no. 1, pp. 12–25, Mar. 2000.

[3] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[4] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[5] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, 2018.

[6] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," *SIGSOFT Softw. Eng. Notes*, vol. 25, pp. 102–112, Aug. 2000.

[7] A. Panichella, R. Oliveto, M. D. Penta, and A. D. Lucia, "Improving multi-objective test case selection by injecting diversity in genetic algorithms," *IEEE Trans. Softw. Eng.*, vol. 41, no. 4, pp. 358–383, Apr. 2015.

[8] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Trans. Softw. Eng.*, vol. 33, no. 2, pp. 108–123, Feb. 2007.

[9] J. A. Jones and M. J. Harrold, "Test suite reduction and prioritization for modified condition/decision coverage," *IEEE Trans. Softw. Eng.*, vol. 29, no. 3, pp. 195–209, Mar. 2003.

[10] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007.

[11] P. Konsaard and L. Ramingwong, "Total coverage based regression test case prioritization using genetic algorithm," in *Proc. 12th Int. Conf. Elect. Eng./Electron., Comput., Telecommun. Inf. Technol.*, Jun. 2015, pp. 1–6.

[12] G. S. Kaur Arvinder, "A genetic algorithm for regression test case prioritization using code coverage," *Int. J. Comput. Sci. Eng.*, vol. 3, no. 5, pp. 1839–1847, 2011.

[13] W. Jun, Z. Yan, and J. Chen, "Test case prioritization technique based on genetic algorithm," in *Proc. Int. Conf. Internet Comput. Inf. Serv.*, Sep. 2011, pp. 173–175.

[14] F. Yuan, Y. Bian, and Z. Li, "Epistatic genetic algorithm for test case prioritization," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2015, pp. 109–124.

[15] Y. Bian, Z. Li, R. Zhao, and D. Gong, "Epistasis based ACO for regression test case prioritization," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 1, no. 3, pp. 213–223, Jun. 2017.

[16] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proc. IEEE 16th Int. Conf. Softw. Eng.*, 1994, pp. 191–200.

[17] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set size and block coverage on the fault detection effectiveness," in *Proc. IEEE Int. Symp. Softw. Rel. Eng.*, Nov. 1994, pp. 230–238.

[18] T. Y. Chen, F. Kuo, H. Liu, and W. E. Wong, "Code coverage of adaptive random testing," *IEEE Trans. Rel.*, vol. 62, no. 1, pp. 226–237, Mar. 2013.

[19] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2009, pp. 233–244.

[20] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," in *Proc. 35th Int. Conf. Softw. Eng.*, May 2013, pp. 192–201.

[21] M. R. Elliott and P. Heller, "Object-oriented software considerations in airborne systems and equipment certification," in *Proc. ACM Int. Conf. Companion Object Oriented Program. Syst. Lang. Appl. Companion*, 2010, pp. 85–96.

[22] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, pp. 366–427, Dec. 1997.

[23] M.-H. Chen, M. R. Lyu, and W. E. Wong, "Effect of code coverage on software reliability measurement," *IEEE Trans. Rel.*, vol. 50, no. 2, pp. 165–170, Jun. 2001.

[24] S. Eghbali and L. Tahvildari, "Test case prioritization using lexicographical ordering," *IEEE Trans. Softw. Eng.*, vol. 42, no. 12, pp. 1178–1195, Dec. 2016.

[25] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, "To be optimal or not in test-case prioritization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 5, pp. 490–505, May 2016.

[26] Y. Singh, A. Kaur, and B. Suri, "Test case prioritization using ant colony optimization," *SIGSOFT Softw. Eng. Notes*, vol. 35, pp. 1–7, July 2010.

[27] B. Suri and S. Singhal, "Implementing ant colony optimization for test case selection and prioritization," *Int. J. Comput. Sci. Eng.*, vol. 3, no. 5, pp. 1924–1932, 2011.

[28] S. Kamna, Y. Singh, S. Dalal, and P. R. Srivastava, "Test case prioritization: An approach based on modified ant colony optimization," in *Proc. Int. Conf. Emerg. Res. Comput., Inf., Commun. Appl.*, 2016, pp. 213–223.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[30] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: Survey and new directions," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 33, no. 5, pp. 560–572, Sep. 2003.

[31] G. S. Tewolde and W. Sheng, "Robot path integration in manufacturing processes: Genetic algorithm versus ant colony optimization," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 38, no. 2, pp. 278–287, Mar. 2008.

[32] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 321–332, Aug. 2002.

[33] D. Martens, M. D. Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens, "Classification with ant colony optimization," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 651–665, Oct. 2007.

[34] C. Juang and C. Lu, "Ant colony optimization incorporated with fuzzy q-learning for reinforcement fuzzy control," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 3, pp. 597–608, May 2009.

[35] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[36] S. Sampath, R. Bryce, and A. M. Memon, "A uniform representation of hybrid criteria for regression testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1326–1344, Oct. 2013.

[37] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 733–752, Sep. 2006.

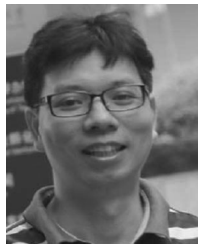
[38] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 593–617, Sep. 2010.

[39] A. Marchetto, G. Scanniello, and A. Susi, "Combining code and requirements coverage with execution cost for test suite reduction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 363–390, Apr. 2019.

[40] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. 18th Int. Symp. Softw. Testing Anal.*, 2009, pp. 57–68.
[41] O. Dahiya and K. Solanki, "A systematic literature study of regression test case prioritization approaches," *Int. J. Eng. Technol.*, vol. 7, no. 4, pp. 2184–2191, 2018.
[42] C. Gu, Z. Li, and R. Zhao, "ACO based test case prioritization and impact analysis of parameters," *J. Frontiers Comput. Sci. Technol.*, vol. 8, no. 12, pp. 1463–1473, 2014.
[43] X. Xing, Y. Shang, R. Zhao, and Z. Li, "Pheromone updating strategy of ant colony algorithm for multi-objective test case prioritization," *J. Comput. Appl.*, vol. 36, no. 9, pp. 2497–2502, 2016.
[44] P. B. Thanedar, J. S. Arora, G. Y. Li, and T. C. Lin, "Robustness, generality and efficiency of optimization algorithms for practical applications," *Struct. Optim.*, vol. 2, pp. 203–212, Dec. 1990.
[45] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proc. 9th Joint Meet. Found. Softw. Eng.*, 2013, pp. 466–476.
[46] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 456–473, Jul. 1999.
[47] T. C. Miller, "Two views of generality," *IEEE Trans. Syst., Man, Cybern.*, vol. 16, no. 3, pp. 450–453, May 1986.
[48] I. Tanev, M. Brzozowski, and K. Shimohara, "Evolution, generality and robustness of emerged surrounding behavior in continuous predators-prey pursuit problem," *Genetic Program. Evol. Mach.*, vol. 6, pp. 301–318, Sep. 2005.
[49] J. Bibai, P. Saveant, M. Schoenauer, and V. Vidal, "On the generality of parameter tuning in evolutionary planning," in *Proc. 12th Annu. Conf. Genetic Evol. Comput.*, 2010, pp. 241–248.
[50] Q. Luo, K. Moran, D. Poshyvanyk, and M. Di Penta, "Assessing test case prioritization on real faults and mutants," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2018, pp. 240–251.
[51] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "Fast approaches to scalable similarity-based test case prioritization," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng.*, May 2018, pp. 222–232.
[52] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "A test case prioritization genetic algorithm guided by the hypervolume indicator," *IEEE Trans. Softw. Eng.*, to be published, doi: 10.1109/TSE.2018.2868082.
[53] Q. Luo, K. Moran, and D. Poshyvanyk, "A large-scale empirical comparison of static and dynamic test case prioritization techniques," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 559–570.
[54] B. Jiang, W. K. Chan, and T. H. Tse, "PORA: Proportion-oriented randomized algorithm for test case prioritization," in *Proc. Int. Conf. Softw. Qual., Rel. Secur.*, Vancouver, BC, Canada, Aug. 3–5, 2015, pp. 131–140.
[55] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Softw. Syst. Model.*, vol. 18, no. 1, pp. 499–521, 2019.
[56] L. Mei, W. K. Chan, T. H. Tse, B. Jiang, and K. Zhai, "Preemptive regression testing of workflow-based web services," *IEEE Trans. Serv. Comput.*, vol. 8, no. 5, pp. 740–754, Sep./Oct. 2015.
[57] L. Mei *et al.*, "A subsumption hierarchy of test case prioritization for composite services," *IEEE Trans. Serv. Comput.*, vol. 8, no. 5, pp. 658–673, Sep./Oct. 2015.
[58] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proc. Int. Symp. Softw. Test. Anal.*, 2014, pp. 437–440.
[59] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng.*, Nov. 2015, pp. 58–68.
[60] A. Shi, T. Yung, A. Gyori, and D. Marinov, "Comparing and combining test-suite reduction and regression test selection," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 237–247.
[61] D. Silva, R. Rabelo, M. Campanh, P. S. Neto, P. A. Oliveira, and R. Britto, "A hybrid approach for test case prioritization and selection," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2016, pp. 4508–4515.
[62] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, "Employing rule mining and multi-objective search for dynamic test case prioritization," *J. Syst. Softw.*, vol. 153, pp. 86–104, Jul. 2019.
[63] S. Y. Shin, S. Nejati, M. Sabetzadeh, L. C. Briand, and F. Zimmer, "Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2018, pp. 49–60.

Chengyu Lu received the B.E. degree in computer science and engineering in 2018 from the South China University of Technology, Guangzhou, China, where he is currently working toward the M.E. degree in computer science.

His current research interests include evolutionary algorithms and swarm intelligence, and their applications in real-world problems.



Jinghui Zhong received the Ph.D. degree in information science and technology from Sun Yat-sen University, Guangzhou, China, in 2012.

He is currently an Associate Professor of Computer Science with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. From 2013 to 2016, he was a Postdoctoral Research Fellow with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include evolutionary computation, machine learning, and agent-based modeling.

Yinxing Xue received the B.E. and M.E. degrees in software engineering from Wuhan University, Wuhan, China, in 2005 and 2007, respectively, and the Ph.D. degree in computer science from the National University of Singapore, Singapore, in 2013.

He is currently a pre-tenure Research Professor with the University of Science and Technology of China, Hefei, China, where he has been a Research Professor with the School of Computer Science and Technology since 2018. His research interests include software program analysis, software engineering, and cyber security issues (e.g., malware detection, intrusion detection, and vulnerability detection).

Liang Feng received the Ph.D. degree in computer engineering from Nanyang Technological University, Singapore, in 2014.

He was a Postdoctoral Research Fellow with the Computational Intelligence Graduate Lab, Nanyang Technological University, Singapore. He is currently an Assistant Professor of Computer Science with the College of Computer Science, Chongqing University, Chongqing, China. His research interests include computational and artificial intelligence, memetic computing, big data optimization and learning, and transfer learning.

Jun Zhang (F'17) received the Ph.D. degree in electronic engineering from the City University of Hong Kong, Kowloon, Hong Kong, in 2002.

He is currently a Visiting Scholar with Victoria University, Melbourne, VIC, Australia. His current research interests include computational intelligence, cloud computing, high-performance computing, operations research, and power electronic circuits.

Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011, and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON CYBERNETICS, and IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.