Swarm and Evolutionary Computation 20 (2015) 23–36

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Regular Paper

Adapting ant colony optimization to generate test data for software structural testing ☆

Chengying Mao [a,b,*], Lichuan Xiao [a], Xinxin Yu [a], Jinfu Chen [c]

[a] School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China
[b] The State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
[c] School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, Jiangsu, China

ARTICLE INFO

Article history:
Received 14 January 2014
Received in revised form
8 July 2014
Accepted 13 October 2014
Available online 28 October 2014

Keywords:
Test data generation
Meta-heuristic search
Ant colony optimization
Branch coverage
Fitness function
Experimental evaluation

ABSTRACT

In general, software testing has been viewed as an effective way to improve software quality and reliability. However, the quality of test data has a significant impact on the fault-revealing ability of software testing activity. Recently, search-based test data generation has been treated as an operational approach to settle this difficulty. In the paper, the basic ACO algorithm is reformed into discrete version so as to generate test data for structural testing. First, the technical roadmap of combining the adapted ACO algorithm and test process together is introduced. In order to improve algorithm's searching ability and generate more diverse test inputs, some strategies such as local transfer, global transfer and pheromone update are defined and applied. The coverage for program elements is a special optimization objective, so the customized fitness function is constructed in our approach through comprehensively considering the nesting level and predicate type of branch. To validate the effectiveness of our ACO-based test data generation method, eight well-known programs are utilized to perform the comparative analysis. The experimental results show that our approach outperforms the existing simulated annealing and genetic algorithm in the quality of test data and stability, and is comparable to particle swarm optimization-based method. In addition, the sensitivity analysis on algorithm parameters is also employed to recommend the reasonable parameter settings for practical applications.

1. Introduction

Software testing has been viewed as an important way to ensure software reliability and trustworthiness. Especially, automated software testing (AST) can greatly reduce manual labor, shorten testing time, and improve testing efficiency. In the past few decades, it has been taken into serious consideration both in the academic world and in industry. Generally speaking, test data automated generation is a key and challenging task in AST [2]. An appropriate test data set can effectively reduce testing execution time and increase fault exposure probability. Apart from the above two issues, coverage ratio is also a critical index to evaluate the quality of test data set. In functional testing (black-box testing), the index can be reflected by function-point coverage, combination pairs or equivalence class coverage. In structural testing (white-box testing), the coverage typically focuses on the construct elements of program code, such as statements, branches, definition-usage pairs and paths. Compared with functional testing, structural testing is more cost-effective to detect faults in program, thus has been widely applied and studied.

However, with the rapid growth of software size and complexity, the traditional structural testing methods like symbolic execution are hard to generate suitable test inputs in practice. Recently, search-based software testing (SBST) has attracted great attention and gained extensive popularity in software testing community [3,4]. The underlying idea of SBST is to adopt some meta-heuristic search (MHS) algorithms to generate test data with high coverage ratio and strong fault-revealing capability. In general, the structure-oriented test data generation is firstly converted into a coverage optimization problem. Subsequently, the search algorithm is used to produce test inputs, which can gradually gain upon the full coverage about program elements. Experimental results show that, SBST can outperform the traditional test data generation methods in several aspects such as cost, efficiency and fault-revealing ability [5]. Although SBST has exhibited its advantage on test data generation, there are still two challenges in this field: first, the cooperation between search process and testing process. That is, how to transform the classical search algorithm to fit the requirements of test data generation, and how execution information of program is effectively feedback to search process? Second, the fitness function also plays critical role for coverage effect and search efficiency.

☆This paper is an extended version of a conference paper that appeared as [1].

* Corresponding author at: School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China.
E-mail address: maochy@yeah.net (C. Mao).

http://dx.doi.org/10.1016/j.swevo.2014.10.003
2210-6502/

At present, meta-heuristic search algorithms including genetic algorithm (GA) [6], simulated annealing (SA) [7,8], tabu search (TS) [9] and particle swarm optimization (PSO) [10] have been widely used to assist software testing activities. GA, SA and PSO, in particular, become the three most well-known methods [11]. Ant colony optimization (ACO) [12–15], as one of MHS algorithms, has been extensively used to obtain the optimal solution in management and industrial field. However, the applications of ACO for generating test data have not been profoundly studied until today. In the paper, we mainly address the problem of ACO-based automated test data generation. The basic ACO is reformed to realize the searching behavior in the continuous intervals of program's input variables. At the same time, an interaction mechanism is designed to utilize the execution trace information to direct the optimization during the evolution process. More importantly, a fitness function comprehensively considering branch's nesting level and predicate type is constructed for branch coverage. In addition, the effectiveness of our ACO-based method has been validated by empirical studies. The main contributions can be summarized as follows.

- The basic ACO algorithm is transformed to a discrete version so as to handle the test data generation problem in structural testing. Some strategies such as local transfer, global transfer and pheromone update are used to improve the coverage capability of test suite.
- A fitness function for branch coverage is designed by fully considering the reachability of branch, which is measured by both nesting level and predicate type. The function can make sure that the branch with high reachable difficulty has more chance to be traversed by test inputs.
- Some well-known benchmark programs are utilized to validate the effectiveness of our ACO-based approach, and the comparative analysis on SA, GA and PSO is also performed. Experimental results show that ACO-based method is better than SA and GA, and is comparable to PSO.
- To facilitate the applications, parameter analysis is also conducted to provide some guidelines for ACO's settings to generate high-quality test data set.

The paper is structured as follows. In the next section, we will briefly review the basic algorithm of ACO. At the same time, related work for automated test data generation and the applications of ACO are surveyed. In Section 3, the overall framework of ACO-based test data generation is addressed. In Section 4, an adapted ACO algorithm for generating test data is presented. Then, the experimental analysis is employed in Section 5. Besides the comparative and sensitivity analysis, the potential threats in our studies are also discussed in this section. Finally, the concluding remarks are given in Section 6.

2. Background

2.1. The outline of basic ACO

ACO algorithm is presented by Marco Dorigo through learning from the ant colony foraging behavior [12–15]. During the foraging process, ant will secrete pheromone on its path for exchanging information with other ants. Since each ant is capable of perceiving the pheromone trail, it can adjust the forward direction according to the strength of pheromone on the path. Finally, it can reach the food destination via several times of adjustments. As mentioned above, ACO algorithm can find the best solution with fast speed through the positive feedback mechanism. Furthermore, the algorithm also possesses such advantages as solid robustness and support for distributed computing. Nowadays, ACO has been widely used to settle the optimization problems in industrial fields [15–17]. However, ACO-based software testing has not been fully studied and is still a challenging problem.

In the past, ACO has mainly been used to find good solutions for difficult discrete optimization problems. The earliest application of this algorithm is to use the traveling salesman problem (TSP) as a test problem [12,13]. In ant system (AS), searching is a process of constructing solutions can be regarded as a path on the construction graph $G=(V,E)$. The set of solutions may be associated either with the set V of nodes of the graph G or with the set E of its edges [17]. Given a graph with n nodes, m ants are used to build a tour in the graph, and the amount of pheromone trail $\tau(i,j)$ associated to edge (i,j) is intended to represent the learned desirability of choosing node j when the ant is on node i. For the k-th ant, suppose it stays on node i now, then the set of neighborhood nodes of the current position (i.e. node i) for such ant can be denoted as $N_{k(i)}$. Obviously, $N_{k(i)}$ contains the nodes which can be possibly visited by ant i in next step. In general, The choice of a node from $N_{k(i)}$ is done probabilistically at each step.

$$p_k(i,j)=\frac{\tau(i,j)\cdot[\eta(i,j)]^{\beta}}{\sum_{u\in N_{k(i)}}\tau(i,u)\cdot[\eta(i,u)]^{\beta}} \tag{1}$$

where $\eta(i,u)$ is a heuristic information and can usually be expressed as $1/d(i,j)$, here $d(i,j)$ is the length of edge (i,j), which is the distance between node i and j. β is a parameter that controls the relative weight of pheromone trail and heuristic value.

Once all ants have completed their tour, pheromone is updated on all edges according to the following equation. It is clear that the aim of pheromone update is to increase the pheromone values associated with good or promising solutions, and decrease those that are associated with bad ones.

$$\tau(i,j)\leftarrow(1-\alpha)\cdot\tau(i,j)+\Delta\tau(i,j) \tag{2}$$

where $\alpha\in(0,1)$ is a pheromone decay parameter, $\Delta\tau(i,j)=\sum_{k=1}^{m}\Delta\tau_k(i,j)$, and $\Delta\tau_k(i,j)$ represents the quantity of pheromone deposited on edge (i,j) by ant k. It is usually defined as

$$\Delta\tau_k(i,j)=\begin{cases}1/L_k & \text{if } (i,j)\in T_k\\ 0 & \text{otherwise}\end{cases} \tag{3}$$

where T_k is the tour done by ant k, and L_k is its length. From the definition of $\Delta\tau_k(i,j)$, it is easy to see that, its value highly depends on how well the ant has performed: the shorter the tour done, the greater the amount of pheromone deposited.

In the latter ant colony system (ACS) [14], $\Delta\tau(i,j)$ is defined on only the globally best ant, that is,

$$\Delta\tau(i,j)=\begin{cases}1/L_{gb} & \text{if } (i,j)\in \text{global}-\text{best}-\text{tour}\\ 0 & \text{otherwise}\end{cases} \tag{4}$$

where L_{gb} is the length of the globally best tour from the beginning of the trial.

In general, ACO algorithm is typically used for discrete optimization problem, especially for problems in graph. In most cases, test inputs of program are chosen from the Euclidean space. Therefore, it is necessary to modify the basic algorithm to handle it.

2.2. Related work

Test data generation is a critical task in automated software testing. In the past decades, search-based test data generation for structural testing has been extensively studied. Here, we review the researches most closely to our work.

In the 1990s, genetic algorithm was adapted to generate test data. Jones [6] and Pargas [18] investigated the usage of GA for test data automated generation for branch coverage. Their experiments on some small programs showed that GA usually outperformed random algorithm significantly. In recent years, Harman and