Computer Languages, Systems & Structures 50 (2017) 1–19

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

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

Synthesizing test scenarios in UML activity diagram using a bio-inspired approach

Vinay Arora [a,*], Rajesh Bhatia [b], Maninder Singh [a]

[a] Computer Science & Engineering Department, Thapar University, Patiala, Punjab, India
[b] Computer Science & Engineering Department, PEC University of Technology, Chandigarh, India

ARTICLE INFO

Article history:
Received 10 February 2017
Revised 4 May 2017
Accepted 15 May 2017
Available online 25 May 2017

Keywords:
Bio-inspired algorithm
Amoeboid organism
Computational intelligence
UML activity diagram
Test scenario generation
Concurrency

ABSTRACT

The model-based analysis is receiving a wide acceptance as compare to code-based analysis in the context of prioritizing and guiding the testing effort and speeding up the development process. Ordinarily, system analysts as well as developers follow Unified Modeling Language (UML) activity diagrams to render all realizable flows of controls commonly recognized as scenarios of use cases. This paper applies a bio-inspired algorithm to produce test scenarios for the concurrent section in UML activity diagram. Here, the heuristic draws its inspiration from the internal mechanism of the slime mould *Physarum Polycephalum*, a large single-celled amoeboid organism. Simulations are performed using eight subject systems taken from the LINDHOLMEN data-set, two models taken from real life student projects and five synthetic models. The results obtained through different approaches are validated through the statistical analysis which demonstrates that our proposed approach is better than the existing Ant Colony Optimization (ACO) and Genetic Algorithm (GA) by a number of feasible test scenarios generated.

1. Introduction

It has been observed that the system analysts and software developers commonly recommend the practice of object-oriented analysis and design standards [1] to decrease the cost with respect to development and maintenance of the software system. In the year 1996, the Object Management Group (OMG) introduced UML system to support the object-oriented practice. UML, being a visual modeling language, has found usages in various software engineering sub-domains, *viz.* testing, traceability analysis, maintenance, *etc.* by modeling different views of software. In an object-oriented system, UML activity diagram is possibly the only design object that is good at depicting the flow of control. The system's behavior in an activity diagram is outlined with regard to activities integrated by control flow entities [2].

In an activity diagram of a use case, a series of activities is known as test scenario or just a scenario. In context of software development, testing and maintenance of object-oriented systems, each and every scenario is related to a trail of an execution of activities in the system. To test a use case adequately, it is necessary to recognize all feasible begin-to-end paths in an activity diagram for covering all activities with its associated control constructs. The control flow constructs, *viz.* decision, looping, synchronization, concurrency, *etc.* decide the execution sequence of various activities. A fork-join node in the UML activity diagram serves as a concurrent entity. The existence of a fork element commences several parallel control flows. The default activity related to join is "AND" that indicates when all the in-flows to a particular join node are

* Corresponding author.
E-mail address: vinay.arora@thapar.edu (V. Arora).

http://dx.doi.org/10.1016/j.cl.2017.05.002
1477-8424/

complete, the final flow is directed on out-edge of a join element. The non-deterministic characteristic of concurrency allows the switching after any activity node of the sub-queues existing between fork-join. This interleaving result into permutation among activity nodes; and the same leads to a sequence explosion for test scenarios that are created amongst fork-join. Therefore, test scenario generation for a concurrent segment in the UML activity diagram is considered equivalent to the problem of combinatorial optimization.

The analytical article of 2016 by Arora et al. [3] encompassing a wide-ranging survey of 179 well-researched papers and inferential observations for the homogeneous groupings, such as testing concurrent programs, concurrency in UML models, and model-based testing with concurrency studied to infer that the work related to test scenario (or path) generation is conducted by human intervention and mostly by applying the exact methods. The survey suggests that there lies the opulence of scope for using the meta-heuristic techniques for generating the test scenarios using the UML activity diagram. The reasonable use of the meta-heuristic algorithm is highly recommended to get the effective solution in cases where popularly practiced exact algorithms cannot efficiently control the instances (size, structure) ranging within the necessitated search time [4,5]. In the recent decades, multiple choices of bio-inspired methods have been anticipated to generate test scenarios for the concurrent section in an activity diagram. All these works have either huge redundancy in generated test scenarios or have a high randomness that results into a less number of final test scenarios. This factor has driven us to explore the utility of meta-heuristic technique that involves less time with better exploration of search space.

We find that the heuristic in biologically inspired approach named Amoeboid Organism Algorithm (AOA) draws its inspiration from the internal mechanism of the slime mould *Physarum Polycephalum,* which explores the paths from an intermediate entity with lesser redundancy as compared to its peer meta-heuristics like ACO, and GA. We apply the AOA on the subject systems taken from the LINDHOLMEN data-set [6], some student projects [7], and synthetic activity diagrams. Then we determine the count of feasible paths obtained, and algorithm execution time using AOA in comparison to ACO and GA.

The rest of the manuscript is structured as follows: we have studied the existing approaches related to generating scenarios for the concurrent section of UML activity diagram in Section 2. The scope of work, conceptualization related to the area of our research and motivation of using Amoeboid Organism (AO) has been presented in Section 3. Our proposed approach using AO is described in Section 4. Section 5 contains our experimental results, comparison with the existing techniques, and statistical analysis using null hypothesis. Section 6 describes the threat to validity for the proposed methodology. Finally, Section 7 concludes with the findings of the proposed work, and sums up with the scope and application of AO for furthering the advancements of the research in the near future.

2. Related work

Although rich literature is available on white-box testing of the multi-threaded code [3,8]; yet the existing exact algorithms under this category cannot be applied to generate scenarios from UML activity diagram. Usage of exact algorithms involves an intermediate graphical form, *viz.* flow graph, dependence graph, *etc*. which lead to processing overhead for fetching information from intermediate notation. This can be bypassed using UML models where the Extensible Markup Language Message Interface (XMI) of the corresponding model can be easily parsed to get the necessary information [9–12].

In a software system, test scenarios can be produced by examining either design level activity diagram or its corresponding source code. UML-based scenario generation has the subsequent edge over code-based scenario generation. First, we require fewer quanta of data to be processed alternative to the large code. Second, we can discover test scenarios at the early stage of system (or software) development life cycle (SDLC) [13]. The same enables software analysts, developers or testers to build a useful test plan and development pattern before the start of development phase. In this section, we further study the existing research work on test scenario generation using meta-heuristic techniques. Table 1 shows the contribution of the researchers who have used bio-inspired techniques for generating scenarios from UML activity diagrams.

Observation: if we perceive research trend for test scenario (or sequence) generation using meta-heuristics, we can observe that the earlier researches missed to present and perform validations on any existing benchmark models. Merits can be recognized only when substantial scenarios are computed in real life implementations; and feasible scenarios can be detected comfortably. The techniques that use ant-like agent suffer from redundancy, as ant may traverse the same path repeatedly due to the accumulation of pheromone on some specific routes. The techniques getting inspiration from chromosomal behavior in genetic theory have an inherent randomness that leads to very fewer paths (or scenarios) generated from fork-join.

3. Scope of work and amoeboid organism algorithm

Example. Consider a sample activity diagram containing fork-join construct as shown in Fig. 1. Here, we can see that there are two sub-queues, namely, $5 \rightarrow 6 \rightarrow 7$ and $8 \rightarrow 9$ between fork-join node.

The incoming control flow creates the concurrent control flow on each out edge at fork node. The join node combines multiple inflows into a single flow. A single control flow is produced only when all the incoming control flows have been completed at the output node. In concurrency, multiple flows of activities start at fork node simultaneously; and the same might produce a permutation for next activity node(s) in test scenario (or sequence) by alternating control between nodes

Table 1
Related work for test scenario (or sequence) generation using meta-heuristic techniques.

S. No.	Year	Author(s)	Meta-heuristic used	Drawback
1.	2005	Xu et al. [14]	Behavior of bacteria (Maxococcus xanthus) as agent	Biologically inspired agents always use some probability factor for its directional orientation (like pheromone in ant colony and flux in amoeboid approach). The study does not ponder much upon the fact that governs the movement of the agent in any specific direction. The agent will clone, as and when required. The exploration of scenario tree uses an approach similar to parallel graph search to obtain all the scenarios, which come under exact algorithm. For a large search space that exists for the problem here, exact algorithms are not viable. The study lacks any benchmark subject system to validate the proposed approach.
2.	2005 2010	Li and Lam [15], Lam [16]	Agents using anti-ant variant	There exists overstrain of converting an activity diagram to a state machine diagram first. The problem of scalability of the proposed algorithm is not discussed here.
3.	2008	Farooq et al. [17]	Random walk algorithm	Symmetric flows turn asymmetric due to the selection of wrong node that may result into a noisy test sequence.
4.	2012	Shirole et al. [18]	Evolutionary algorithm (EA) [where EA is a flavor of GA]	Values for the various parameters like selection percentage, crossover percentage, and crossover point have not been provided. With an increase in the count of nodes in sub-queues under fork-join, the probability of getting the right sequence diminishes due to the cumulative effect of randomness at various steps in EA.
5.	2014	Mishra and Mohapatra [19]	Ant colony optimization (ACO)	The interleaved execution of nodes under the concurrent section of activity diagram has not been discussed. ACO has been used for test sequence prioritization but not for their generation. The study has missed demonstration of the fork-join nodes in sample model taken for validating the proposed approach.
6.	2015	Sayyari and Emadi [20]	ACO with Markov chain	Researchers have missed giving the discussion on UML models having concurrency construct. Benchmark models are not taken to validate their proposed approach.

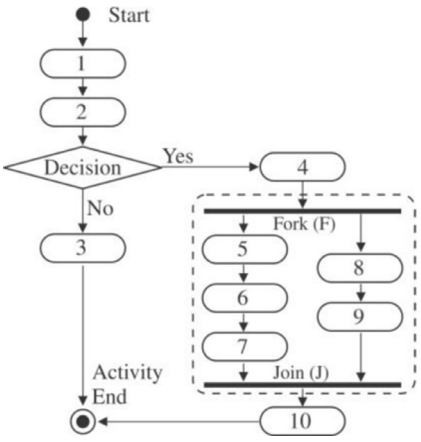

Fig. 1. An activity diagram with concurrent element.

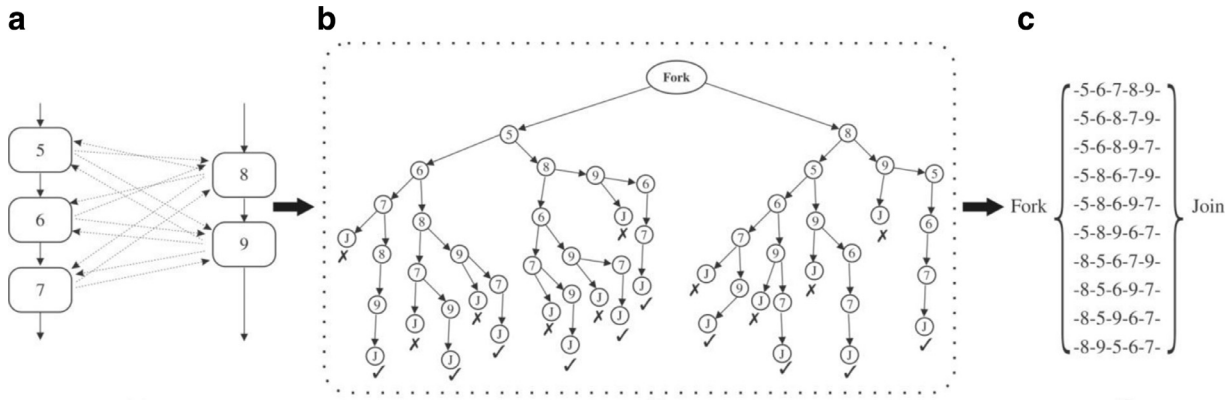

Fig. 2. Concurrent region of activity diagram with an exploratory tree structure and final feasible scenarios.

Table 2
Combinatorial explosion of test scenarios for variable size and count of fork-join queues.

Case label	Count of fork-join sub-queues	Elements in each sub-queue	Count of feasible test scenarios generated using formula [18]
a.	2	$q_1=3$, $q_2=2$	10
b.	2	$q_1=5$, $q_2=4$	120
c.	3	$q_1=5$, $q_2=4$, $q_3=3$	27,720
d.	4	$q_1=5$, $q_2=4$, $q_3=3$, $q_4=2$	2,522,520

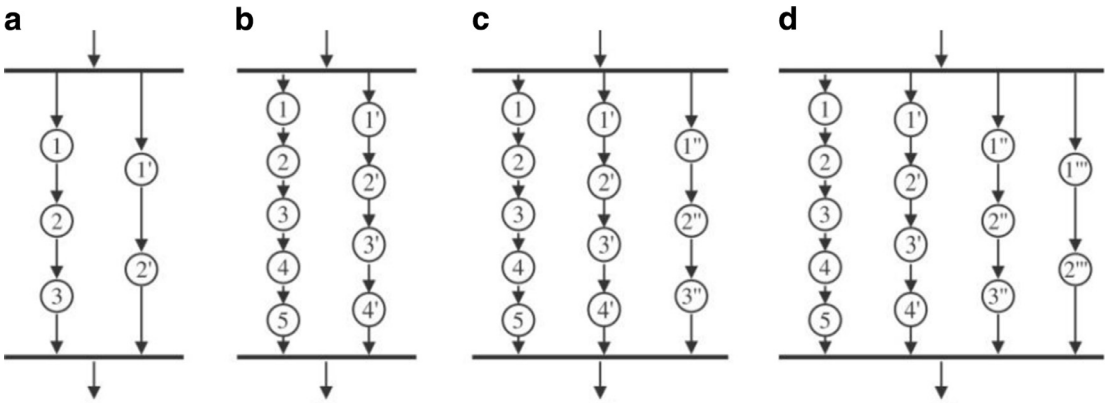

Fig. 3. Variable count of activity nodes and sub-queues under fork-join.

under fork-join. In the absence of switch point(s) in concurrent sub-queues, all nodes will get executed in the serial order, consequently ensuing loss of concurrency among the activities.

Fig. 2(a) shows the interleaved execution and the switching after each node present in the sub-queues, which result into an exponential sequence count for the generated test scenarios. The tree structure in Fig. 2(b) depicts an exploratory view with all the valid and invalid scenarios generated. In the above diagram, the $\surd$ stands for valid and ✗ indicates invalid paths. Fig. 2(c) represents the set of feasible test scenarios generated from the sub-queues, namely, $5\rightarrow6\rightarrow7$ and $8\rightarrow9$.

The case becomes problematic when the count of sub-queues and (or) the node count in the sub-queues varies increasingly. Both the situations lead to enhance the number of permutations among the nodes and henceforth raise the count of test sequences exponentially. The combinatorial explosion with regard to the number of succeeding test scenarios by adopting the formula $\frac{(n_1+n_2+\cdots n_i)!}{(n_1!\ \times n_2!\ \times\ \cdots n_i!)}$ is described in Table 2. In the formula used, i stands for the count of sub-queues and n represents the number of activity nodes in ith sub-queue [18].

Fig. 3 shows four sample activity diagrams corresponding to the cases listed in Table 2. Table 3 shows that a wait time of more than 7 min exists while executing Depth-first Search (DFS) [2,21] on Case d. In the case of moderate and significant instances, it would be quite impractical to assess all possible solutions received through exact algorithms like DFS, Breadth-first Search, *etc.* [21] as it becomes a time-consuming process. Here, the application of ACO [22] is giving a good number of feasible test scenarios, but this count can be increased further by using such a variant where the agent is less redundant in following the same path repeatedly. As the path with high pheromone accumulation becomes influential, there exists duplicity in the traversed paths. For Cases b–d, GA fails as the count of activity nodes in any sub-queue or number of

Table 3
Count of test scenarios (CTS) and algorithm execution time (AET) in millisecond obtained using DFS exact algorithm and meta-heuristic approaches on the synthetic activity diagram in Fig. 3. (Population in GA or number of ants in ACO is taken as 100 and number of generations in GA or number of trials in ACO is taken as 10.) Here, average results obtained after executing the specific approach with 30 instance runs have been listed.

Activity diagram	DFS [21]		ACO [22]		GA [18]	
	CTS	*AET* (ms)	*CTS*	*AET* (ms)	*CTS*	*AET* (ms)
a	10	1.5	10	418	10	117
b	120	13	57	928	0	30
c	27,720	1149	610	2412	0	26
d	No result after wait of 7 min		1424	4499	0	28

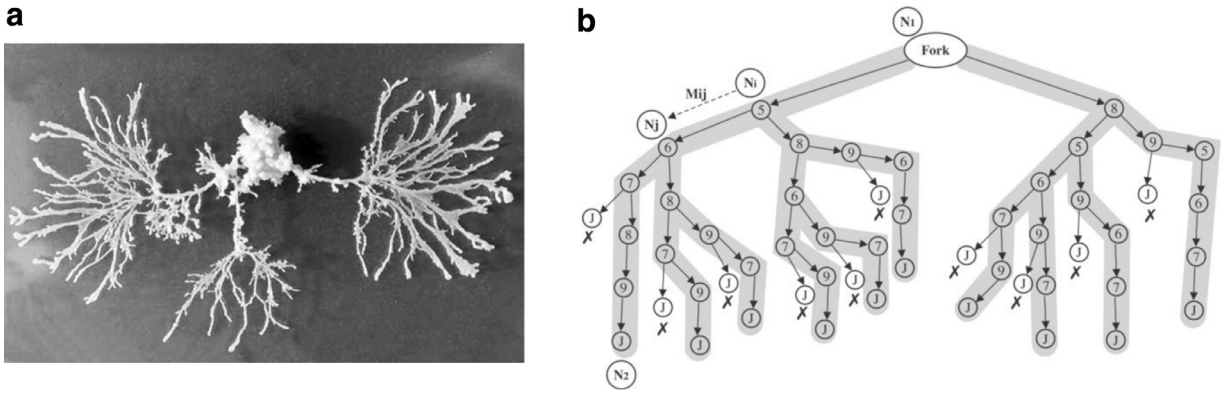

Fig. 4. (a) Tubular structure of veins in *Physarum Polycephalum* [23, 32] (b) Tubular structure in the exploratory tree representing test scenarios among existing permutations.

sub-queues increases. GA (or EA) [18] uses a random function to generate and select the final scenarios. However, it remains inefficient in offering the desired number as well as quality due to the large set of vague test scenarios, resulted from inherent randomness.

3.1. Amoeboid organism algorithm and motivational factor for its usage

In the context of biological taxonomy, the *Physarum Polycephalum* belongs to the super-group Amoebozoa, phylum Mycetozoa, class Myxogastria. It is a single-celled amoeboid organism that shapes a dynamic net of protoplasmic tubes spanning sources of nutrients. The biological system that governs the tube formation and selection leads to the *Physarum*'s path-finding capability: tubes thicken in the direction in which the flux persists. Hence, the plasmodium of *Physarum Polycephalum* can be considered as one of the biological models of natural computation.

The amoeboid organism algorithm is one of the emerging biology-based computational intelligent (CI) algorithms derived from the amoeboid inspired studies [23–29]. Amoeboid organism exhibits sophisticated computing capabilities where the shape-changing dynamics of such organism includes stochastic fluctuations; and the Plasmodium spread by networks can be made programmable. The network of tubes for *Physarum Polycephalum* through which nutrients and chemical signals circulate throughout the organism is shown in Fig. 4(a) [30]. Required circulation for maintaining the appropriate level of nutrients and chemical signals is dependent on streaming across a mixed-up net of Tubular Channels (TCs). Henceforth, the meshed-up and complicated framework of the tubular structure is associated with the carriage of materials and forewarnings within the entity. The tube expands with adequate flux. Summarily, the widening results in the further build-up of flux for the opposition to the flow of sol subside in the expanded route. Thus, TCs having a substantial flux flourish, while those having a skimpy flux value vanish.

The amoeboid organism algorithm has already been applied in diversified areas that demand exclusivity of computational abilities such as travel salesman problem, the problem of vehicle routing and solving complex maze [31–34], shortest path [24,25, 35,36], transportation network [26,37], game theory [38], rough set [39,40], *etc*. However, to the best of our knowledge, this amoeboid organism approach is yet to be sufficiently researched for imbibing its abilities in solving the problem of test scenario generation for a concurrent section of the UML activity diagram.

The pivotal motivator behind the application of amoeboid organism algorithm to look out scenarios for concurrent section in an activity diagram comes from the existing similarity between the tubular veins like structure of *Physarum Polycephalum* and the test scenarios that might be generated by incremental traversal over the edges present in the similar tree type structure representing the permutations under fork-join node. Flux movements in *Physarum Polycephalum* generate articulated paths from the source node (root node) to sink node (leaf node). Representing a scenario with a single root node and

multiple sink nodes consequently developed into a structure, similar to the one shown in Fig. 4(a). Marked tubular structure in Fig. 4(b) depicts the feasible test scenarios generated by incremental traversal of the exploratory tree structure.

Undoubtedly, the characteristic feature of the duct diameter is auto-synergist. Specifically, the network is adaptable to the fluctuations of flux. The work, therefore, includes this adaptability in the model discussed. Finally, empirical evidence not being extensive, but subjective, there lies a scope of great liberty in devising the model.

3.2. Mathematical model for using amoeboid organism algorithm

The study on the amoeboid organism [41] provides an outline of the tube creation method. TCs thicken in a specified route when shuttle swarming of the protoplasm progresses in that path for a specific time. That means a positive return between flux and the diameter of the tube, as the sol conductivity remains higher in a thicker medium. Using the maze as depicted in [41], a mathematical formulation to generate the test scenarios for a concurrent section in the UML activity diagram has been constructed. Tubular structures shown in Fig. 4(b) represent scenarios under fork-join with two distinct nodes related to the food sources, named as N_1 *and* N_2, and the other intermediate nodes as N_3, N_4, N_5, so on and so forth. Each time one (N_1) among the source nodes becomes evident as a source and the other (N_2) imitates a sink. The segment of duct between N_i *and* N_j signifies an edge as M_{ij}.

3.3. Flux streaming through TCs

The identifier Q_{ij} has been taken for expressing the flux through M_{ij} from N_i *to* N_j. Assume that the flow over the tubular path is tentatively *Poiseuille flow;* and the same is conveyed by Eq. (1):

$$Q_{ij} = \frac{\pi a_{ij}^4}{8k} \frac{p_i - p_j}{L_{ij}} \tag{1}$$

where, a_{ij} and L_{ij} denote the radius and length of the tube for an edge M_{ij}, p_i specifies the pressure at the node N_i and k is the viscosity coefficient of the sol streaming inside the tubular structure. By setting $D_{ij} = \frac{\pi a_{ij}^4}{8k}$, as the conductivity of the edge M_{ij}, Eq. (1) can be reframed as under:

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}\left(p_i - p_j\right) \tag{1a}$$

Now, taking the law of conservation of sol and assuming zero capacity at each node, it can be summed as:

$$\sum_i Q_{ij} = 0 \ (j \neq 1,\ 2) \tag{2}$$

Eqs. (3a) and (3b) hold for the source N_1 and sink N_2,

$$\sum_i Q_{i1} + I_0 = 0, \tag{3a}$$

$$\sum_i Q_{i2} - I_0 = 0, \tag{3b}$$

where, I_0 is the flux from the source node (or the flux into the sink node). The value of I_0 is considered fixed in the model adopted here which implies a steady total flux throughout the process.

3.4. Adaptation

Experiments indicate [31–34] that the tubes with larger flux value continue to exist, whereas the tubes with the reduced flux values deteriorate. Concerning adaptation of tubular width, it is assumed that D_{ij} varies considerably over a time frame as per the flux Q_{ij}. Eq. (4) calculates the change of D_{ij}(t), where r is a deterioration rate of the duct.

$$\frac{d}{dt}D_{ij} = f\left(\left|Q_{ij}\right|\right) - rD_{ij} \tag{4}$$

Eq. (4) also means that the conductivity tends to disappear in the absence of flux along the edge, while the conductivity enhances in the presence of flux. Thus, it is likely to presume that f is a monotonically growing continuous function equates the value $f(0)=0$. The edge lengths $L_{ij}'s$ remain fixed all over the adaptation procedure.

Eqs. (1a)–(3b) have been used to derive the Poisson equation taking pressure as the parameter, as under:

$$\sum_i \frac{D_{ij}}{L_{ij}}\left(p_i - p_j\right) = \begin{cases} -I_0 \ for\ j = 1, \\ +I_0 \ for\ j = 2, \\ 0 \ otherwise. \end{cases} \tag{5}$$

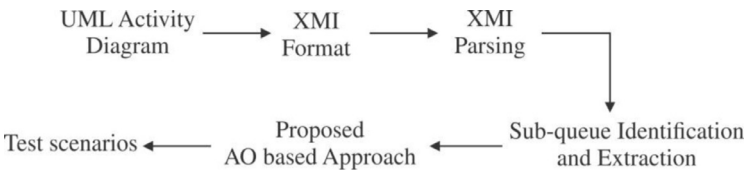

Fig. 5. Methodology to generate test scenarios from UML activity diagram.

The value for all $p_i's$ can be obtained by using Eq. (5); and each $Q_{ij} = \frac{D_{ij}}{L_{ij}}(p_i - p_j)$ is computed by framing $p_2 = 0$ as an elementary pressure level. The present model articulates the evolution process using the variable $D_{ij}'s$ that progresses by the adaptation of Eq. (4). The variables p_i and Q_{ij} are computed by Eq. (5) using the specified value of the $D_{ij}'s$ (and $L_{ij}'s$). The conductivity is proximally attributed to the tubular width. Henceforth, the fading of TCs is stated by the withering of the conductivity of edges. Specific edges flourish or persist, while others droop during the evolution of the model. It is acknowledged that the system has reached at a solution to generate test scenarios when the leftover edges build a path (or paths) bridging the two specific nodes $N_1(source)$ *and* $N_2(sink)$. Here, the proposed AO-based approach applies flux and conductivity values to decide for the next node to jump upon. Finally, this approach generates a path from source node to sink by incremental traversal over each intermediate edge.

4. Proposed approach

In this section, we first list the steps, given in Fig. 5, for getting test scenarios from UML activity diagram and subsequently, describe our technique. Activity diagram has been drawn using Rational Software Architect (RSA) and exported to its respective Extensible Markup Language Message Interface (XMI) format.

After parsing XMI code, sub-queues present under the fork-join are extracted. The proposed approach using amoeboid organism has been applied to generate the permutations among activity nodes of fork-join sub-queues. In order to develop the proposed approach, we have defined some notations which have been used throughout the manuscript.

SQ_i	sub-queues between *fork-join* structure/s of an activity diagram
a_k	agent inspired from amoeboid organism
DC_{ij}	set of nodes directly connected to the node at *i*th position
$InDPV_k$	set of child node(s) of the recently visited node(s), where child node(s) belong to the sub-queue(s) different from the one to which *i*th node belongs
Q_{ij}	flux between the node *i* and *j*
CPC	current position counter
TN	number of total activity nodes between fork-join pair
X_k	set of all unvisited nodes that belong to the sub-queue(s) different from the one to which *i*th node belongs
F_{ij}	set containing nodes that are directly (or indirectly) connected to the current node *i*
$p_{selection}$	probability value used to select between directly and indirectly connected nodes
TS_s	set having test scenarios which start with fork node and end up-to join node.

4.1. Algorithm to generate test scenarios for concurrent section

This sub-section outlines the steps for applying amoeboid organism approach so as to compute the test scenario in the concurrent section of UML activity diagram. The algorithm takes fork-join sub-queues as its input, and an amoeboid organism agent from a particular colony starts traversal from the fork node. A set $DC_{ij} = \{dc_{i1}, dc_{i2} \ldots dc_{ik}\}$ consists of entities directly connected to the fork node. The condition ($ith node \neq fork\ and \neq join$) checks the current node for being an intermediate activity node; and in such case, it also computes the indirectly connected nodes on which the control may jump. After this step, the probability P_{ij} for all the *j*th next nodes is computed. A set $InDPV_j$ is computed which constitutes indirectly connected nodes. The condition ($randomNumber \leq p_{selection}$) selects the set of nodes to be chosen by the amoeboid agent, *i.e.* either directly connected or indirectly connected activity nodes. Under loop construct (**For** $\forall f_{ij} \in F_{ij}$), the flux Q_{ij} is computed for all the *j*th elements of the set chosen in the previous step. Flux will be non-zero in two situations, firstly, when *j*th node(s) is not a join node. Secondly, when it is a join node and all the other available activity nodes in the sub-queues have already been covered and flux is zero, otherwise. A set Q_{ij} constitutes the flux value for all the *j*th elements. When the computed flux value comes out to be zero, the edge from the current *i*th node to next *j*th node is rejected. If the flux is non-zero, and the *j*th node is join, then accept the edge from *i*th node to *j*th node. The whole path from fork up-to-current join node is accepted as a feasible path. When flux is non-zero and *j*th node(s) is not a join node, then replicate (or clone) the amoeboid agent on the *j*th node(s) and these *j* nodes will become current node as *i*. From all *i* node(s) repeat the initial step to compute the directly connected nodes and execute till the stopping criteria are met.

Input: Sub-queues(SQ_i), where $i=\{1, 2, 3\ldots n\}$, TN
Output: Test scenario(s)(TS_s): $ts_1, ts_2, \ldots, ts_m$
Begin

1. Initiate the traversal of a_k from *fork* node at ith position with CPC_i
2. Compute $DC_{ij}=\{dc_{i1}, dc_{i2}\ldots dc_{ik}\}$
3. **If** ith *node* $\neq$ *fork and* $\neq$ *join* **then**

$$\forall\ j \in X_k,\quad compute\ P_{ij} = \begin{cases} \frac{Q_{ij}}{\sum_{j\in X_k} Q_{ij}} & if\ j \in InDPV_k \\ 0 & if\ j \notin InDPV_k \end{cases},\ Where\ InDPV_k \subseteq X_k$$

$InDPV_j = j$th *next node*(*s*) *with non* − *zero probability value*

Else

$InDPV_j = \emptyset$

EndIf

4. **If** *randomNumber* $\leq p_{selection}$ **then**

$F_{ij} \leftarrow DC_{ij}$

Else

$F_{ij} \leftarrow InDPV_{ij}$

EndIf

5. **For** $\forall f_{ij} \in F_{ij}$ **do**

$CPC_{j\text{th node}} = CPC_{i\text{th node}} + 1$

$$\Delta p_{ij} \begin{cases} = 1, & If\ j\text{th node} \notin join \\ = 1, & If\ ((j\text{th node} \in join)\ \wedge (CPC_j = TN_{fj} + 1)) \\ = 0, & If\ ((j\text{th node} \in join)\ \wedge (CPC_j \neq TN_{fj} + 1)) \end{cases}$$

$Q_{ij} = \frac{\pi a_{ij}^4\ \Delta p_{ij}}{8k\ L_{ij}}$, where $Q_{ij_m} \in Q_{ij} = \{Q_{ij_1}, Q_{ij_2}, \ldots, Q_{ij_n}\}$

EndFor

6. **For** $\forall\ Q_{ij} \in Q_{ij}$ **do**

If $\exists\ Q_{ij_m} = 0$ **then**

neglect edge $i \rightarrow j$

$\therefore$*neglect full path from fork up to join node*

Continue

EndIf

If $\exists\ (Q_{ij_m} \neq 0\ \wedge\ j \notin join)$ **then**

accept all edges from $i \rightarrow j$

replicate the agent a_k *on* jth *node*(*s*)

node j will become the current node now $(i \leftarrow j)$

For all jth *node*(*s*); *Move to step* 2

Continue

EndIf

If $\exists\ (Q_{ij_m} \neq 0\ \wedge\ j \in join)$ **then**

accept edge $i \rightarrow j$

Path from fork to current join node will result into feasible $\boldsymbol{ts_z}$

Load ts_z *in set* TS_s

Continue

EndIf

EndFor

4.2. A sample run of the proposed algorithm

Fig. 6 represents the sample run of the proposed approach on the concurrent section of activity diagram in Fig. 1. In Fig. 6, the solid line denotes directly connected, and dashed line represents the indirectly connected node. Cloud like symbol denotes an amoeboid organism agent. While simulating the proposed approach the initial value for the parameter CPC_i and $p_{selection}$ is taken as 0 and 0.8, respectively.

The algorithm commences with the traversal of a single agent a_k, from a particular colony, initiated at the fork node. From the current position, a set DC_{ij} is computed, which contains the nodes that are directly connected to the current node. This set may contain more than one element. Using *Poiseuille flow* (Eq. (1)), elements for set $InDPV_j$ are computed. $p_{selection}$ stands for a fixed number probability value which is used to compare with a randomly generated number. This comparison selects the tentative element(s) for agent a_k to jump upon by choosing among directly and indirectly connected elements. Flux is computed for the elements of chosen set. When the computed flux for the element(s) is non-zero, and the same is not a join node, then edge between $i \rightarrow j$ will be accepted. Agent a_k will get virtually replicated on each jth element, and flow is repeated with the computation of set DC_{ij}. A particular thread of an agent stops traversing when a join node is encountered to an associated non-zero flux value. Using a single agent here, two paths are obtained, namely, $Fork \rightarrow 5 \rightarrow 8 \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow Join$ and $Fork \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow Join$.

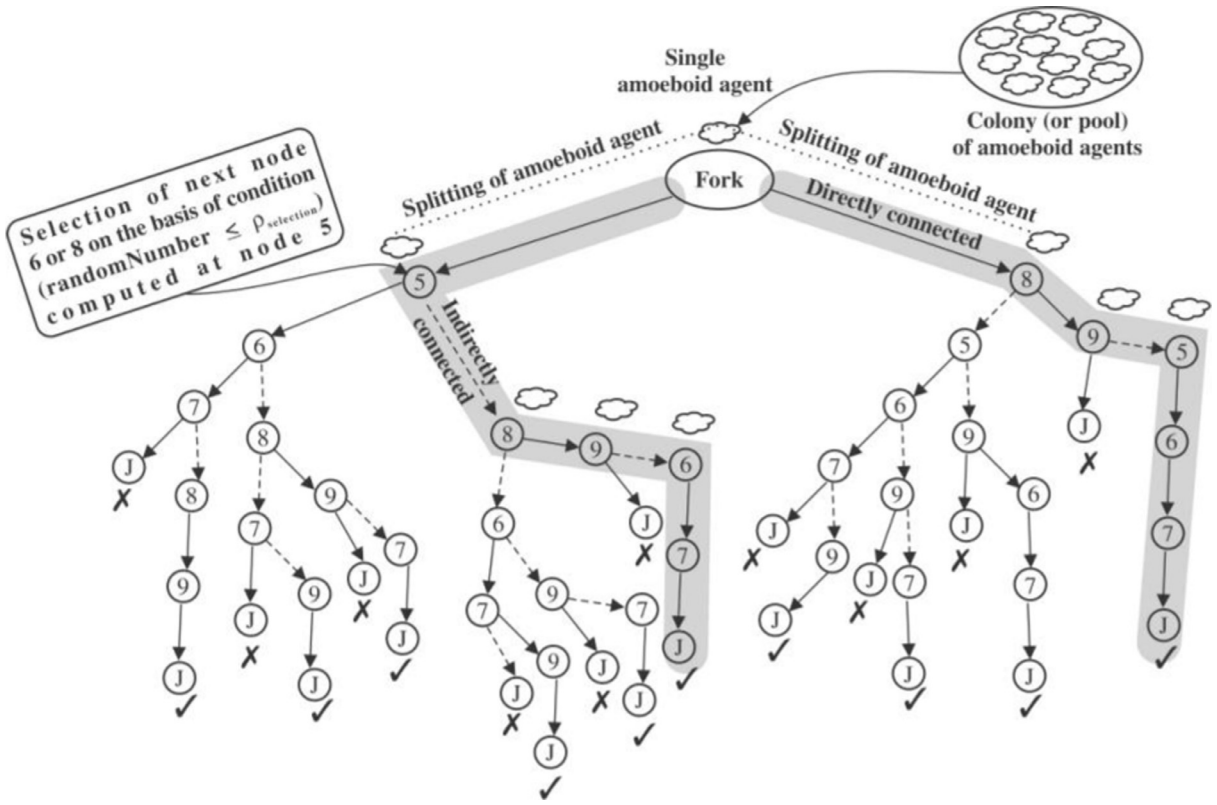

Fig. 6. Sample run of the proposed approach on an example activity diagram.

5. Experimental results and analysis

5.1. Parameter setting and stopping criteria

To simulate the work, the considered experiments have been performed on Intel Core i3-6100 U, 2.30 GHz, 4GB RAM, Windows 10, 64-bit operating system, which has been referred as configuration C1. Guidelines provided in the study by Črepinšek et al. [42] have been referred for setting the values of control parameters while simulating existing ACO, GA, and the proposed AOA approaches. Further, we have set the parameters to different algorithms as follows:

- GA parameters

GA has already been applied in multiple sub-domains such as gene theory, genetics, multiple sequence alignment, bio-informatics, knapsack problem, software optimization, *etc.* for sequence generation [43,44]. Here, AOA has been compared with EA (that is a version of GA) of the Shirole et al. [18]. Fig. 7 outlines the approach adopted for simulating EA along with control parameters.

- ACO parameters

The technique of using ant colony optimization for generating paths was provided by the Srivastava et al. [22]; and the same has been simulated to have a comparison with the proposed approach of this work. Existing ACO-based method [22] uses the fundamentals as derived from the study conducted by Dorigo and Stützle [45]. Various parameters with their associated values taken for simulating ACO are as follow: τ_{ij} is the pheromone level at the edge from node *i to j* (0.8), ρ being the evaporation rate of pheromone (0.1), η_{ij} heuristic value for the edge between node *i to j* (1.5), α factor controlling the pheromone level (0.9), and β factor controlling the heuristic value (0.8).

- AOA parameters

Various parameters with their associated values taken for simulating the technique using amoeboid organism for generating the test scenarios are as follows: CPC_i for an agent a_k at *fork* node is 0 and value for $p_{selection}$ is taken as 0.8.

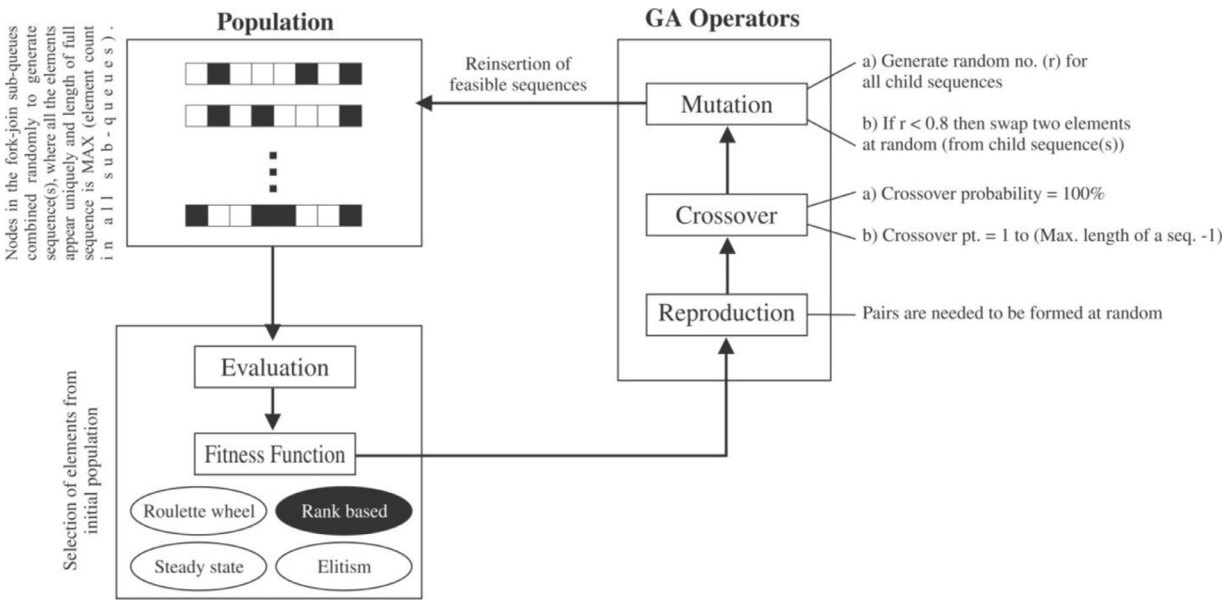

Fig. 7. Generating test scenarios through EA approach [33, 41].

- Stopping criteria

Various bio-inspired approaches (*e.g.* ant colony, genetic algorithm) usually engage varied amount of fitness evaluation functions. Researchers in their studies [46,47] suggested considering an equal number of consumed fitness function evaluations to achieve fair comparison amongst different meta-heuristics. Following this recommendation, we set a maximum number of fitness evaluation function equal to 300,000 as stopping criteria for all the algorithms (AOA, GA, and ACO) simulated here for performing the experimentation.

- Generic settings for all the algorithms

For conducting an empirical analysis, the computation efforts for the meta-heuristic can be used to compare the given instance [4]. In order to execute a defined set of statements for the given iterations, computational effort is the total execution time taken by the machine. The computational effort of the proposed AOA has been analyzed by considering two factors, namely, (i) Time in milliseconds for the execution of algorithm only, and (ii) Time in milliseconds to convert UML activity diagram into XMI and further up to the execution of the algorithm. In the experimental setup, where AOA, GA, and ACO are simulated, the parameter named the number of agents in AOA is considered as equivalent to the population in GA or number of ants in ACO. The number of trials in AOA is considered same as the number of generations in GA or number of trials in ACO. All the approaches are executed thirty times for each subject system. For executing the algorithms on C1, the trial count is taken as 10 and population as 100. For comparing the computational effort, the time (in milliseconds) to execute the algorithm only has been considered as the parameter.

5.2. Objectives of experimentation

The proposed approach is executed on the activity diagrams taken from the LINDHOLMEN dataset [6], student projects, and few synthetic cases. The prime intents of our experimentation are as follows:

(i) To compare the proposed approach with ant colony and genetic algorithm by the count of feasible test scenarios (or sequences) generated.
(ii) To conduct statistical analysis for validating the results obtained through the amoeboid organism approach.

5.3. Subject systems

Since the proposed approach aims to produce feasible test scenarios in design level UML activity diagrams, accessibility of UML design level requirements is quite essential for its validation. In the product-based software industry, design level activity diagrams having sufficient details are prepared prior to the coding phase for software (or system). However, such types of designs are not obtainable in the open world for the sake of privacy and sensitivity concerns, and that is why these could not be exercised in our research. Hence, we used the LINDHOLMEN data-set [6] that provides a list of open source

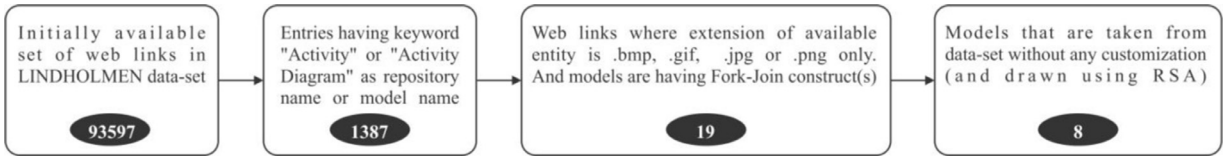

Fig. 8. Selection procedure of activity diagram from the LINDHOLMEN data-set.

Table 4
Activity diagrams from the LINDHOLMEN data-set selected for experiments.

Model no.	Repository name in LINDHOLMEN data-set	# of elements in AD	# of fork-join(s) in AD	# of elements under fork-join(s)	Control constructs present in fork-join	# of scenarios generated using [18]
LDSET1	267492/moje_rep/	34	1	18	Y	120
LDSET2	atadeesom/BubbleBeeDoc/	31	1	10	Y	6
LDSET3	BGCX261/zimeoprocess-svn-to-git/	65	2	6 (FJ1), **20(FJ2)**	Y	20
LDSET4	DmitriyG/Diplom/	36	1	15	N	120
LDSET5	HandreWatkins/COS-301-Phase2/	38	1	24	Y	1680
LDSET6	ooad-2014–2015/JotaPlanet/	33	2	**9(FJ1)**, 6(FJ2)	Y	7
LDSET7	85pando/Zusammenfassungen/	22	1	16	N	35
LDSET8	bsaunder/cs414-discussions/	56	3(1, 2*)	**11(FJ1), 34**(FJ2,contain nested FJ3 with 6 elements)	Y	9, 13

projects that use UML. A flowchart presenting modus operandi for selecting activity diagrams from [6] is shown in Fig. 8 and the details of finally selected activity diagrams are shown in Table 4.

Model numbers LDSET1, LDSET2, LDSET4, LDSET5, and LDSET7 have one fork-join (FJ) node. LDSET3 and LDSET6 contain two fork-join pairs, but we have considered the fork-join having more number of elements in it. Model number LDSET8 constitutes three fork-join constructs, one independent (FJ1) and one with nested fork-join (FJ2, FJ3; where FJ3 is the part of FJ2). We are referring to independent fork-join (FJ1) of LDSET8 as LDSET8 (a), and another (FJ2) as LDSET8 (b) while using these in our experiments.

5.4. Comparison of AOA with existing GA and ACO

In this sub-section, benchmarks as identified from the LINDHOLMEN dataset are taken for comparing the proposed AOA with existing GA and ACO.

For the cases taken from the LINDHOLMEN dataset, the box-plot shown in Fig. 9 depicts that the AOA is providing more number of feasible test scenarios as compare to GA and ACO for LDSET1, LDSET3, LDSET5, LDSET7 and LDSET8b. For the benchmarks LDSET2, LDSET6 and LDSET8a, AOA is giving results equivalent to GA and ACO. For LDSET4, AOA is providing results equivalent to ACO but better as compared to GA.

5.5. Student project as subject system

In addition to the LINDHOLMEN dataset, researchers took the liberty to use the models created by eight bachelor of engineering (B.E.) students as a part of the academic assignment in the software engineering class tutored by the corresponding author in the year 2015, and the same were further fine-tuned [7]. The students worked together in two concurrent groups of three and five participants respectively, each spending around 45 working hours on devising and creating the models with its corresponding contextual description. All the students went through the process of similar methodology discussed during the course. The method used was at par with the one being employed in the industrial projects. The students used RSA with a significant portion of UML 2.2. Table 5 describes the size measures for the two AD models created by the student groups. Here, the count of elements under fork-join is tentatively three times as compared to almost all the cases of LINDHOLMEN dataset. Hence, the models under student project also served a great purpose to test the scalability of the proposed approach.

All the activity diagrams with nested fork-join structure and control constructs like the loop, if-else, *etc.* have been transformed first into an Intermediate Testable Model (ITM) that serve as an input activity diagram with no nested/control

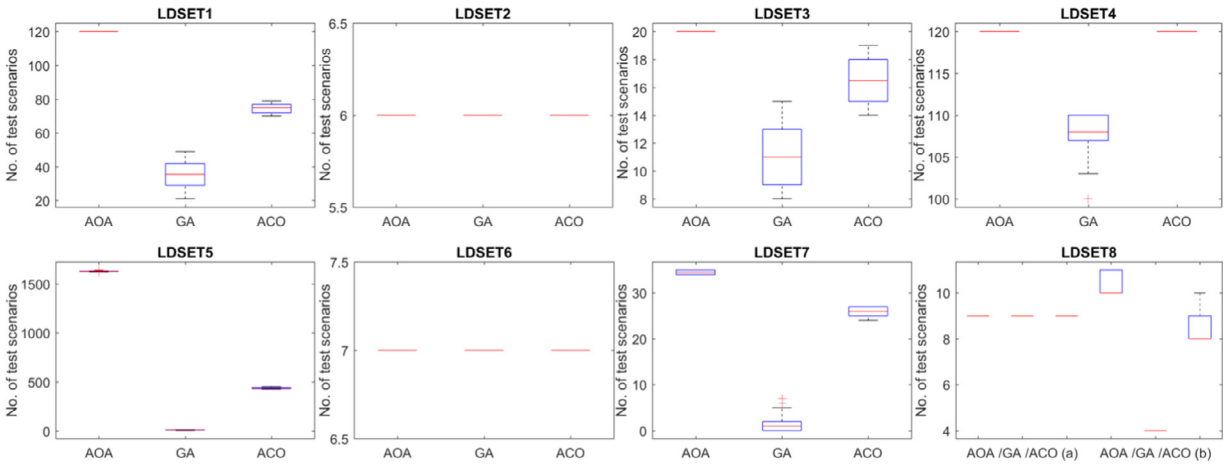

Fig. 9. Generation of feasible test scenarios using the AOA, GA and ACO.

Table 5
Size measures of the activity diagrams with number of total elements.

Model no.	AD model [7]	# of students involved	# of elements in AD	XMI file (KB)	# of fork-join(s) in AD	# of control construct(s) in AD	Nested fork-join
STDP1	Manchurian	3	98	29	4	3	Yes
STDP2	VideoFile	5	89	26	2	2	Yes

Table 6
Count of test scenarios generated using formula in [18].

Model no.	Model name	Annotation for the sub-section in Fig. 10	# of scenarios generated using [18]
STDP1	Manchurian	*A*	*2*
		B	120
		C	35
		D	792
STDP2	VideoFile	A	70
		B	705,432

construct [2]. For attaining the desired simplified activity diagram, each one of the nested fork-join structure or the control construct has been replaced by a single annotated node. Starting from the innermost fork-join (or control construct) the replacement is headed towards outer construct until there remained only one fork-join. Here, the proposed AOA has been applied to each independent and simplified activity diagram having concurrency construct (with no nested or control constructs) for generating the test scenarios. Fig. 10 highlights the whole transformational process of receiving a simplified AD from the complex nested version of activity diagrams which have been taken from [7]. Consequently, the resulted final activity diagram remains 'the derived final non-nested simplified version' generated to serve the experiment.

Table 6 shows the count of scenarios that can be generated using the formula in [18] for each sub-section of the graph shown in Fig. 10 for STDP1 and STDP2.

The box-plot shown in Fig. 11 explains that the AOA is providing more number of feasible test scenarios as compared to its meta-heuristic peers, GA and ACO for the sub-activity diagrams (A, B, C, D for student project 1; and A, B for student project 2) of Fig. 10. The application of AOA on the final simplified version of AD has resulted into generating feasible test scenarios with a lot of annotated nodes (each representing nested fork-join or control construct). AOA is recursively applied to generate further feasible test scenarios on annotated nodes representing nested fork-join. Annotated nodes representing control constructs get substituted by considering its all simplified paths from start to the end node [2]. By doing the substitution for annotated nodes, the final count of feasible test scenarios increases greatly.

For the student projects considered here, GA is performing ghastly because of inherent randomness at crossover and mutation phase in the said algorithm. Experiment shows that when node count in the sub-queues under fork-join increases, the randomness factor in algorithm will lead to erratic combination of the nodes in the final scenarios.

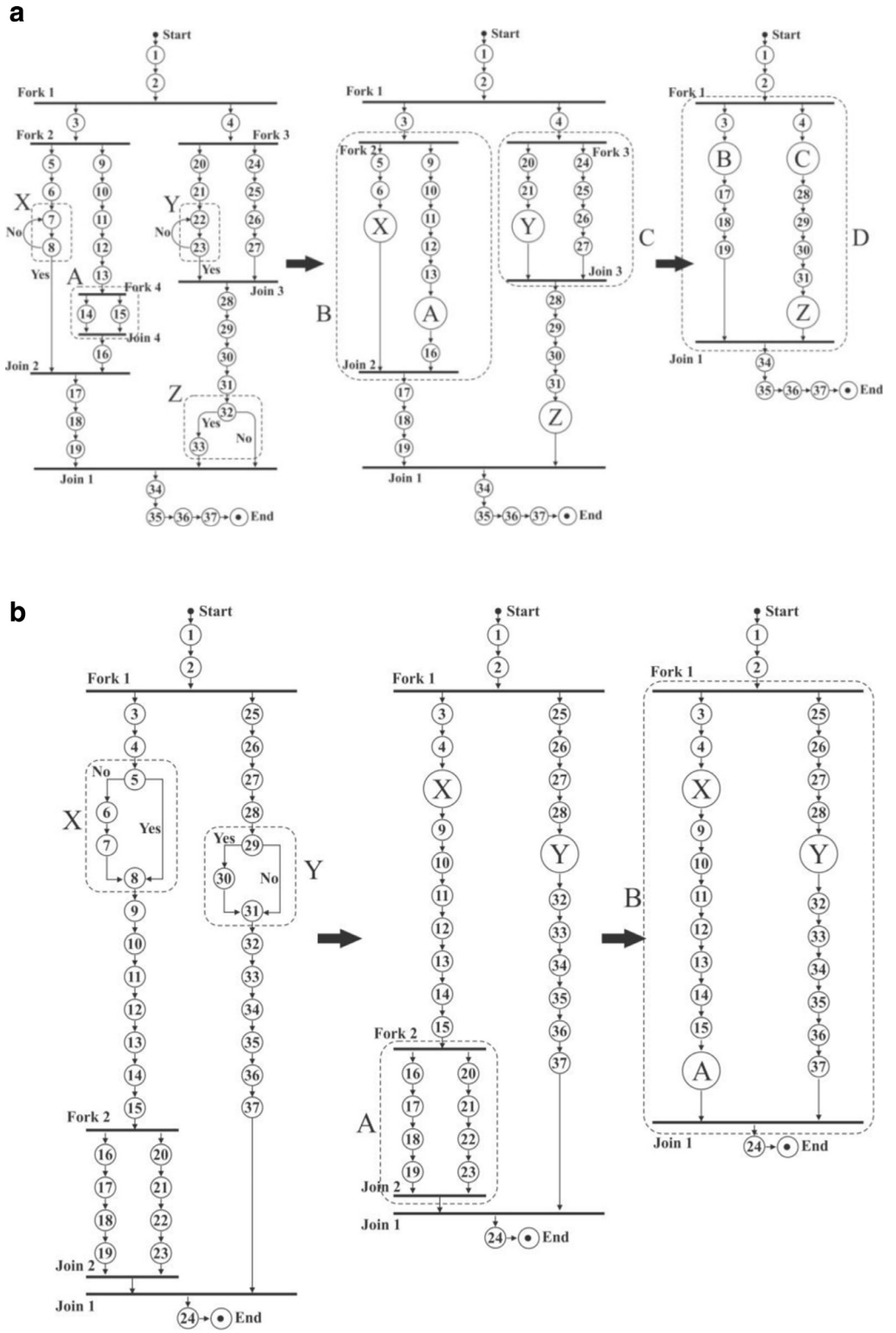

Fig. 10. (a) and (b) sums up the whole transformational process of receiving a simplified activity diagram from its complex nested version titled Manchurian and VideoFile respectively.

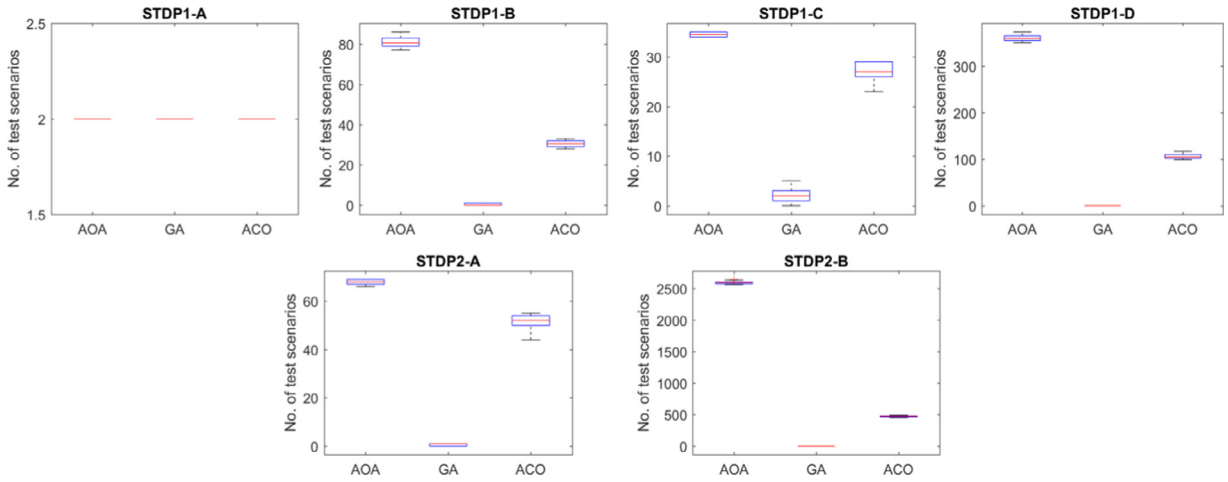

Fig. 11. Count of feasible test scenarios generated using AOA, GA and ACO on student project 1 (STDP1) and student project 2 (STDP2).

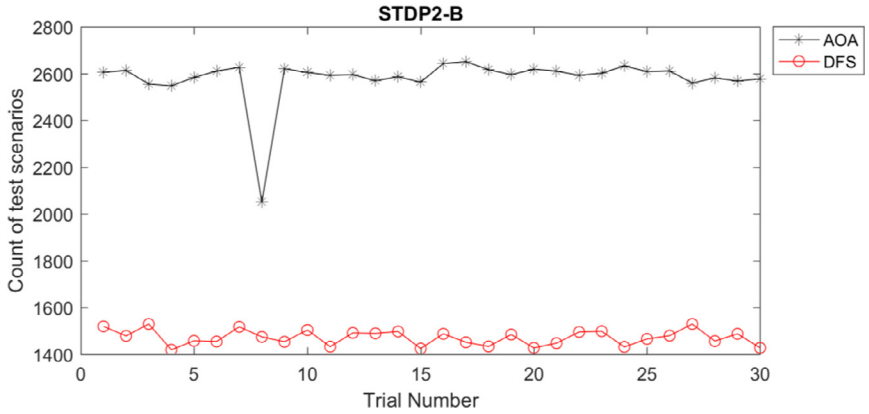

Fig. 12. Count of test scenarios generated using AOA and DFS using STDP2-B on C1.

5.6. Comparison of proposed AOA with DFS on large search space

Table 2 shows that using DFS is not worth for the activity diagrams where the count of test scenarios is large. To justify this statement, we compared DFS with the proposed AOA by prematurely ending the exact algorithm for sub-activity diagram B of STDP2, where the count of test scenarios generated using formula [18] is huge. The time taken for premature termination of the DFS is same as taken by the proposed AOA (using other parameter settings same as taken earlier while comparing AOA with its peers, except the maximum count of fitness function evaluations which has been raised to 3,000,000). Fig. 12 shows that the AOA is better as compare to the DFS for the UML activity diagrams where the count of generated test scenarios is large.

5.7. Activity diagram with multiple join nodes

In this sub-section, Fig. 13 exhibits the synthetic cases for activity diagrams having one fork node and multiple join nodes.

Line graph in Fig. 14, and box-plot in Fig. 15 present the results of executing AOA, GA, and ACO on synthetic activity diagrams as shown in Fig. 13.

The results shown in Figs. 14 and 15 indicate that the AOA is either better or equivalent to GA and ACO in context of number of feasible test scenarios obtained. And at the same time AOA is always better than GA and ACO on the basis of time consumed in milliseconds for executing the algorithm.

5.8. Statistical analysis

On the basis of research studies conducted by Derrac et al. [48] and Garg [49], *t*-test was performed for the purpose of this study using MATLAB tools on the pair of algorithms to (using the count of test scenarios obtained) examine the

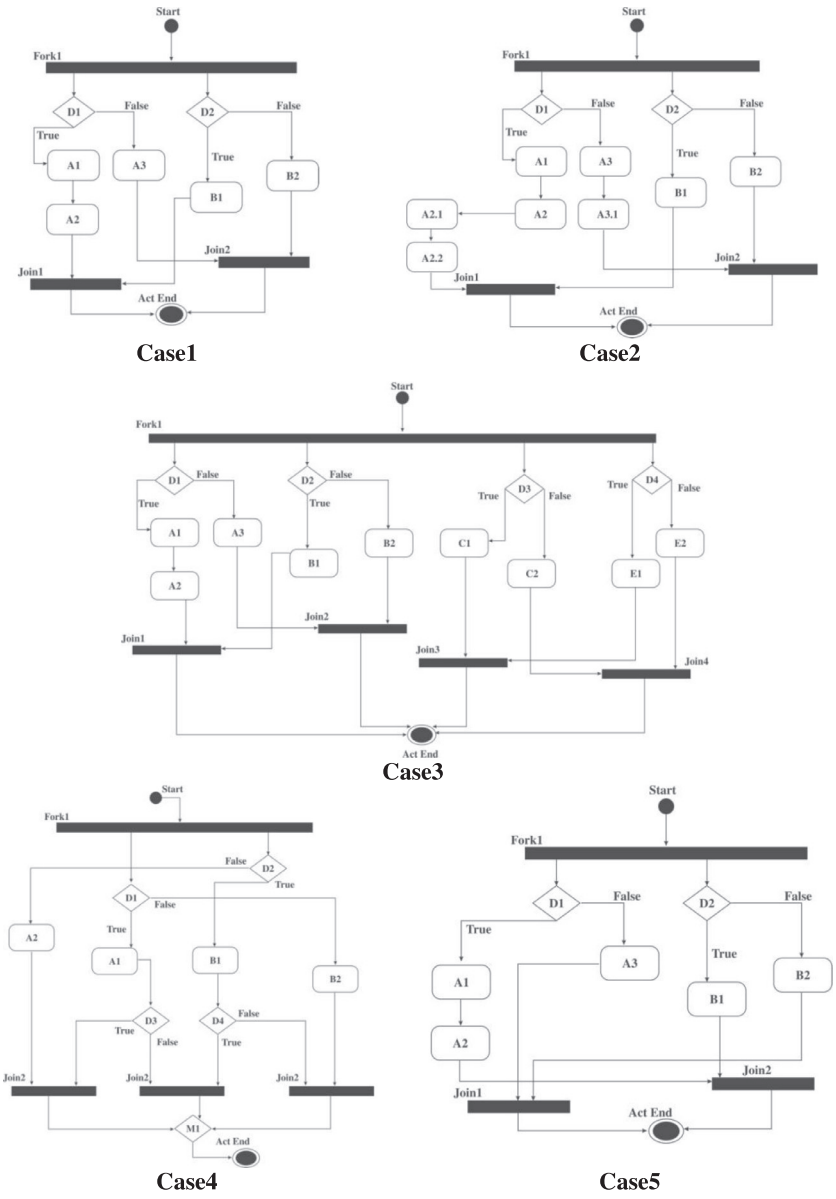

Fig. 13. Synthetic activity diagrams with single fork and multiple join nodes.

statistical significance of the results as highlighted in Fig. 9. Moreover, statistical evaluation has been done with the presumption that the populations have identical variances at the significance level of $\alpha = 0.05$ for the presented AOA outcomes with ACO and GA results.

Tables 7 and 8 show the *t*-test results considering parameters *h, p*, SDEV and Mean; where, *h* is hypothesis test result, *p* is the probability of observing a test statistic as extreme as, or more extreme than, the observed value under the null hypothesis, SDEV is the standard deviation, and Mean is the average of results obtained for all the instance runs. Entry, Not a Number (NaN) for *h* and *p* in *t*-test, denotes that the values taken by comparing the results obtained from the pair of algorithms are the same for all the instances. Thus, SDEV comes out to be zero in this case.

The test has been carried out antagonistic towards the null hypothesis that there exists no distinction in their population means. Other than NaN, the value of *h* as one denotes rejection of the null hypothesis. In each case, the *p*-values obtained are less than the significance level $\alpha = 0.05$. Hence, there exists a difference between the two types of means. In addition to it, the average of the test scenarios obtained for the proposed AOA approach is better and standard deviation (SDEV) for the results obtained using AOA is less than the existing ACO and GA. Thus, the results are statistically significant.

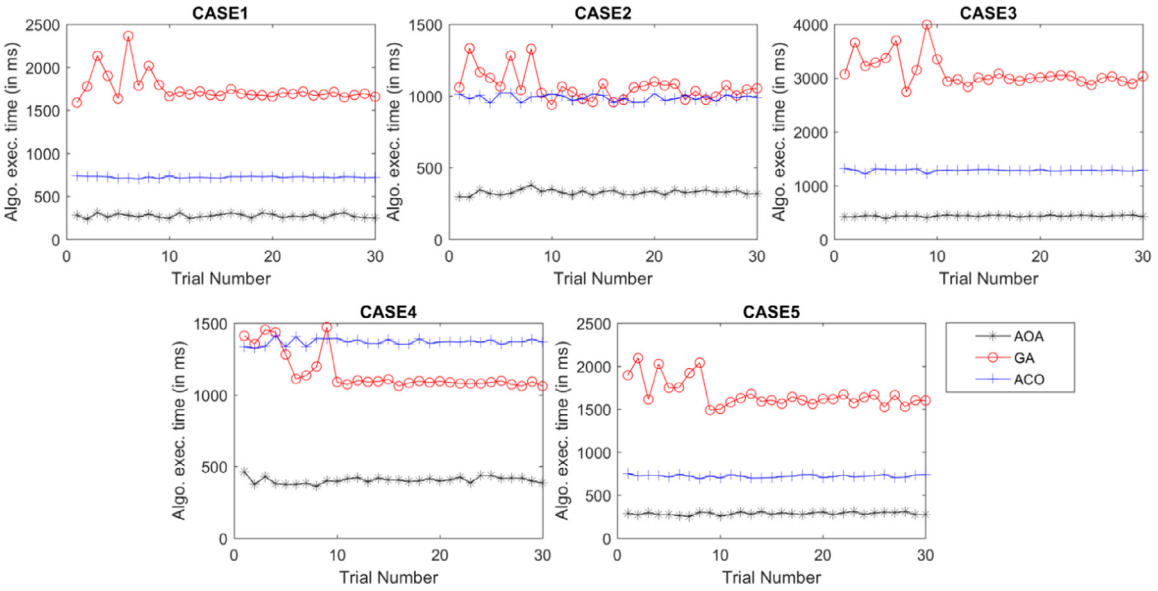

Fig. 14. Computational effort (in millisecond) for AOA, GA and ACO (algorithm execution only).

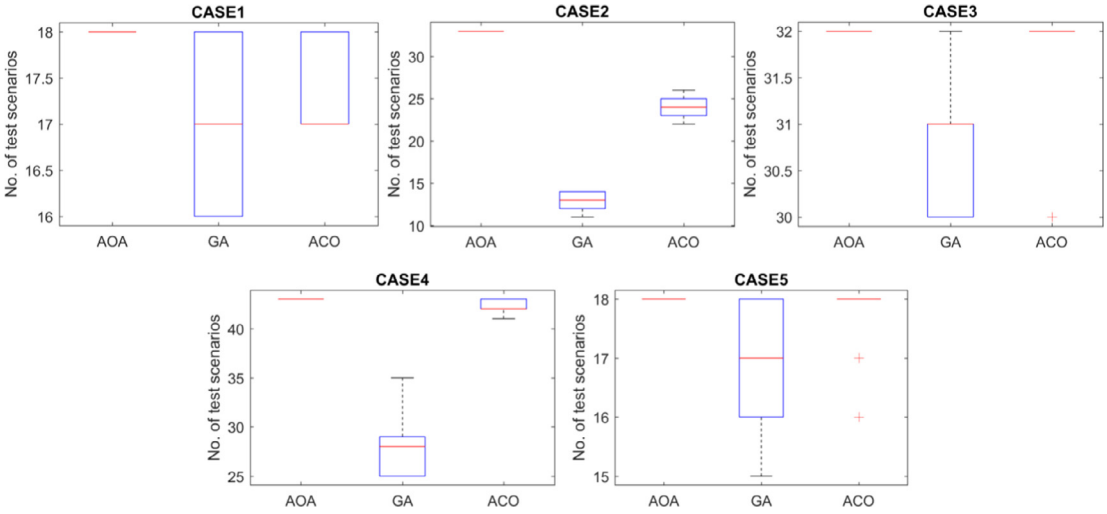

Fig. 15. Generation of feasible test scenarios using the AOA, GA and ACO.

Table 7
Results of *t*-test taking count of feasible test scenarios generated as a parameter for AOA and GA.

Model no.	AOA vs. GA						
		h	p	SDEV AOA	SDEV GA	Mean AOA	Mean GA
LDSET1		1	1.92E−57	0	6.683398544	**120**	35.56667
LDSET2		NaN	NaN	0	0	6	6
LDSET3		1	1.72E−30	0	2.063364065	**20**	11.46667
LDSET4		1	1.20E−36	0	2.157318472	**120**	108.3667
LDSET5		1	8.78E−135	5.144353003	2.896886954	**1628.133333**	11.23333
LDSET6		NaN	NaN	0	0	7	7
LDSET7		1	5.94E−66	0.508547628	1.792705526	**34.5**	1.4
LDSET8	a	NaN	NaN	0	0	9	9
	b	1	1.49E−58	0.479463301	0	**10.33333333**	4

Table 8.
Results of *t*-test taking count of feasible test scenarios generated as a parameter for AOA and ACO.

Model no.	AOA vs. ACO						
		h	p	SDEV AOA	SDEV ACO	Mean AOA	Mean ACO
LDSET1		1	1.68E−65	0	2.62722265	**120**	74.16667
LDSET2		NaN	NaN	0	0	6	6
LDSET3		1	1.05E−17	0	1.52224879	**20**	16.6
LDSET4		NaN	NaN	0	0	120	120
LDSET5		1	1.14E−116	5.144353003	7.308002227	**1628.133333**	435.8
LDSET6		NaN	NaN	0	0	7	7
LDSET7		1	2.76E−46	0.508547628	0.949894126	**34.5**	25.83333
LDSET8	a	NaN	NaN	0	0	9	9
	b	1	3.74E−12	0.479463301	0.858359837	**10.33333333**	8.766667

6. Threats to validity

The present section discusses the prospective threats about the validity of subject system used. There have been three types of potential threats for consideration:

- The threats to *construct validity* deal with the issue of measurement and raise concerns about the things to be measured and ignored in simulations. Failing the same will pose serious threats to validity. The primary threat to construct validity is where test sequence generation techniques using AOA involve flux (Q_{ij}) value calculations. Flux values play a significant role in deciding the next node to be added in the sequence, and thus, it is a vital deciding factor for full sequence generation. We use some static initial values for the Q_{ij} constructs, whereas some other estimates may turn to be of greater significance to enhance the efficiency of AOA.
- Threats to *internal validity* are concerned about the changes, which if introduced will affect the outcome. A major concern with regard to internal validity in such studies has been instrumentation effects, failing which they may lead to the biased results. To minimize and control the resemblance of similar effects, the validation is performed by algorithmic execution on benchmark and synthetic activity diagrams. One source of such effects is the difference in the test process inputs, in which the locality of the model varies. At this time, the effects related to the locality of model changes (*i.e.* cross-synchronization, automatic detection of missing nodes, *etc.*) are not considered. To limit the problems associated with this, we applied the proposed algorithm to each subject model and validated the results for scalability.
- The threats to *external validity* of the present study are based on the issues of how pivotal the subject systems of our studies are. The subject models taken from LINDHOLMEN dataset and student projects are small (or medium) in size. The models, larger in size may yield flexible cost-benefit trade-offs. The decision to use a single type of enhancement at a time in the model to check the scalability of the algorithm may involve another source of threats. Although the single type of enhancement facilitated the process of simulation and analysis, yet in practice, the model extension might crop up in various other distributions. In general, such threats can be addressed only by conducting further studies on additional subjects.

7. Conclusion

After having an analysis of the experimental evaluation of AOA on UML activity diagram, it has been observed that important concern crops up in case the sub-queue(s) under fork-join interleaved with the execution of the respective adjacent sub-queue(s). The same creates a sequence explosion. At present, the studies of amoeboid organism based test sequence generation techniques are pre-eminent to researchers and advanced research on heuristic approaches is pivotal to determine resource-feasibility. Studies can be empowered and enriched by opting amoeboid organism-based approaches. The AOA is an excellent alternative as compared to GA and ACO to find the feasible set of scenarios when elements in the set of total generated paths are affluent.

Further research may investigate the implementation of the proposed amoeboid organism-based algorithm in various other ways. Firstly, the proposed approach can be merged with parallel BAT algorithm to automatically select the best suitable scenarios having maximum coverage. Secondly, artificial bee colony (ABC) algorithm can be applied with AOA for generating the test case from test scenarios, where values for the variable used in test scenario can be determined automatically. Thirdly, AOA can be applied in a parallel computing environment to facilitate the computation of sequences under fork-join pair more efficiently for effective model-based testing. In spite of certain limitations, the proposed amoeboid-based metaheuristic algorithm may trigger the present technological research by providing abundant of model-based software testing prospects for attaining academic excellence and leveraging industrial pull.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cl.2017.05.002.

References

[1] Kundu D, Sarma M, Samanta D. A UML model-based approach to detect infeasible paths. J Syst Softw 2015;107:71–92. doi:10.1016/j.jss.2015.05.007.
[2] Nayak A, Samanta D. Synthesis of test scenarios using UML activity diagrams. Softw Syst Model 2011;10(1):63–89. doi:10.1007/s10270-009-0133-4.
[3] Arora V, Bhatia R, Singh M. A systematic review of approaches for testing concurrent programs. Concurr Comput Pract Exp 2016;28(5):1572–611. doi:10.1002/cpe.3711.
[4] Talbi EG. Metaheuristics from design to implementation. John Wiley & Sons; 2009.
[5] Yang XS. Nature-inspired optimization algorithms. Elsevier; 2014.
[6] Hebig R, Quang TH, Chaudron MRV, Robles G, Fernandez MA. The quest for open source projects that use UML: mining GitHub. In: Proceedings of ACM/IEEE nineteenth international conference on model driven engineering languages and systems. Saint-Malo; 2016. p. 173–83. [Dataset]. doi:10.1145/2976767.2976778.
[7] UML Activity Diagram Repository, https://github.com/UMLADRepo/AD-Repository [as accessed on 10th September 2016] [Dataset].
[8] Shirole M, Kumar R. UML behavioral model based test case generation: a survey. ACM SIGSOFT Softw Eng Notes 2013;38(4):1–13. doi:10.1145/2492248.2492274.
[9] Verma V, Arora V. Code and model based test sequence generation for multithreaded programs. Patiala, India: Thapar University; 2014.
[10] Jyoti A, Arora V. Visualization of deadlock and wait-notify anomaly in multithreaded programs. Patiala, India: Thapar University; 2014.
[11] Jyoti A, Arora V. Debugging and visualization techniques for multithreaded programs: a survey. In: Proceedings of the IEEE international conference on recent advances and innovations in engineering (ICRAIE); 2014. p. 1–6. doi:10.1109/ICRAIE.2014.6909202.
[12] Asadollah SA, Sundmark D, Eldh S, Hansson H, Afzal W. 10 years of research on debugging concurrent and multicore software: a systematic mapping study. Softw Qual J 2016;25:1–34 [published online]. doi:10.1007/s11219-015-9301-7.
[13] Pressman RS. Software engineering: a practitioner's approach. 7th ed. New York, NY: McGraw-Hill; 2010.
[14] Xu D, Li H, Lam CP. Using adaptive agents to automatically generate test scenarios from the UML activity diagrams. In: Proceeding of the twelfth Asia-Pacific software engineering conference; 2005. p. 8. doi:10.1109/APSEC.2005.110.
[15] Li H, Lam CP. Using anti-ant-like agents to generate test threads from the UML diagrams. Testing of communicating systems, 3502. Berlin Heidelberg: Springer; 2005. p. 69–80. doi:101007/11430230_6.
[16] Lam CP. Computational intelligence for functional testing, artificial intelligence applications for improved software engineering development: new prospects, information science reference. 1st ed. IGI Global; 2010. p. 233–58.
[17] Farooq U, Lam CP, Li H. Towards automated test sequence generation. In: Proceedings of the nineteenth Australian conference on software engineering; 2008. p. 441–50. doi:10.1109/ASWEC.2008.4483233.
[18] Shirole M, Kommuri M, Kumar R. Transition sequence exploration of UML activity diagram using evolutionary algorithm. In: Proceedings of the fifth India software engineering conference. ACM; 2012. p. 97–100. doi:10.1145/2134254.2134271.
[19] Mishra A, Mohapatra DP. Generation and prioritization of test sequences using UML activity diagram, Rourkela, India: National Institute of Technology; 2014. B.Tech. Dissertation.
[20] Sayyari F, Emadi S. Automated generation of software testing path based on ant colony. In: Proceedings of the second international congress on technology, communication and knowledge (ICTCK). Mashhad, Iran: Islamic Azad University; 2015. p. 435–40. doi:10.1109/ICTCK.2015.7582709.
[21] Kundu D, Samanta D. A novel approach to generate test cases from UML activity diagrams. J Object Technol 2009;8(3):65–83.
[22] Srivastava PR, Baby K, Raghurama G. An approach of optimal path generation using ant colony optimization. In: Proceedings of the IEEE Region 10 Conference (TENCON 2009); 2009. p. 1–6. doi:10.1109/TENCON.2009.5396088.
[23] Xing B, Jing GW. Innovative computational intelligence: a rough guide to 134 clever algorithms. Heidelberg: Springer Cham; 2014.
[24] Zhang X, Zhang Y, Zhang Z, Mahadevan S, Adamatzky A, Deng Y. Rapid Physarum Algorithm for shortest path problem. Appl Soft Comput 2014;23:19–26. doi:10.1016/j.asoc.2014.05.032.
[25] Zhang X, Wang Q, Adamatzky A, Chan FT, Mahadevan S, Deng Y. A biologically inspired optimization algorithm for solving fuzzy shortest path problems with mixed fuzzy arc lengths. J Optim Theory Appl 2014;163(3):1049–56. doi:10.1007/s10957-014-0542-6.
[26] Zhang X, Adamatzky A, Yang H, Mahadaven S, Yang XS, Wang Q, et al. A bio-inspired algorithm for identification of critical components in the transportation networks. Appl Math Comput 2014;248:18–27. doi:10.1016/j.amc.2014.09.055.
[27] Schumann A, Adamatzky A. Physarum spatial logic. New Math Nat Comput 2011;7(3):483–98. doi:10.1142/S1793005711002037.
[28] Schumann A, Pancerz K. Towards an object-oriented programming language for Physarum Polycephalum computing: a petri net model approach. Fundam Inf 2014;133(2–3):271–85. doi:10.3233/FI-2014-1076.
[29] Pancerz K, Schumann A. Some issues on an object-oriented programming language for Physarum machines. Applications of computational intelligence in biomedical technology. Springer International Publishing; 2016. p. 185–99. doi:101007/978-3-319-19147-8_11.
[30] Tero A, Kobayashi R, Nakagaki T. A mathematical model for adaptive transport network in path finding by true slime mold. J Theor Biol 2007;244:553–64. doi:10.1016/j.jtbi.2006.07.015.
[31] Zhang X, Huang S, Hu Y, Zhang Y, Mahadevan S, Deng Y. Solving 0-1 knapsack problems based on amoeboid organism algorithm. Appl Math Comput 2013;219(19):9959–70. doi:10.1016/j.amc.2013.04.023.
[32] Liu Y, Gao C, Zhang Z, Wu Y, Liang M, Tao L, et al. A new multi-agent system to simulate the foraging behaviors of Physarum. Nat Comput 2015:1–15. doi:10.1007/s11047-015-9530-5.
[33] Zhang X, Wang Q, Adamatzky A, Chan FT, Mahadevan S, Deng Y. An improved Physarum Polycephalum algorithm for the shortest path problem. Sci World J 2014;2014:1–9. Article ID 487069. doi:10.1155/2014/487069.
[34] Masi L, Vasile M. Multidirectional physarum solver: an innovative bio-inspired algorithm for optimal discrete decision making. Glasgow, United Kingdom: University of Strathclyde; 2013 Report 01012013.
[35] Zhang X, Wang Q, Adamatzky A, Chan FTS, Mahadevan S, Deng Y. An improved Physarum Polycephalum algorithm for the shortest path problem. Sci World J 2014;9:2014 Article ID 487069. doi:10.1155/2014/487069.
[36] Zhang X, Wang Q, Chan FTS, Mahadevan S, Deng Y. A Physarum Polycephalum optimization algorithm for the bi-objective shortest path problem. Int J Unconv Comput 2014;10(1/2):143–62.
[37] Zhang X, Adamatzky A, Chan FTS, Deng Y, Yang H, Yang XS, et al. A biologically inspired network design model. Sci Rep 2015;5:10794 Article number:. doi:10.1038/srep10794.
[38] Schumann A, Pancerz K, Adamatzky A, Grube M. Bio-inspired game theory: the case of Physarum Polycephalum. In: Proceedings of the eighth international conference on bio-inspired information and communications technologies, December 01–03, 2014, Boston, Massachusetts; 2014. p. 9–16. doi:10.4108/icst.bict.2014.257869.
[39] Pancerz K, Schumann A. Rough set models of physarum machines. Int J Gen Syst 2015;44(3):314–25. doi:10.1080/03081079.2014.997529.
[40] Pancerz K, Schumann A. Rough set description of strategy games on physarum machines, in advances in unconventional computing. Prototypes, models and algorithms, 2. Springer International Publishing; 2017. p. 615–36. doi:101007/978-3-319-33921-4_23.
[41] Tero A, Kobayashi R, Nakagaki T. Physarum solver: a biologically inspired method of road-network navigation. Phys A Stat Mech Appl 2006;363(1):115–19. doi:10.1016/j.physa.2006.01.053.
[42] Črepinšek M, Liu SH, Mernik M. Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Appl Soft Comput 2014;19:161–70. doi:10.1016/j.asoc.2014.02.009.
[43] Arcuri A, Fraser G. On parameter tuning in search based software engineering, search based software engineering, 6956. Berlin Heidelberg: Lecture Notes in Computer Science, Springer; 2011. p. 33–47. doi:10.1007/978-3-642-23716-4_6.

[44] Karimpour R, Ruhe G. Evolutionary robust optimization for software product line scoping: an explorative study. Comput Lang Syst Struct 2017;47:189–210. doi:10.1016/j.cl.2016.07.007.
[45] Dorigo M, Stützle T. Ant colony optimization. Cambridge, Massachusetts: The MIT Press; 2004.
[46] Mernik M, Liu SH, Karaboga D, Črepinšek M. On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. Inf Sci 2015;291:115–27. doi:10.1016/j.ins.2014.08.040.
[47] Amarjeet JK, Chhabra. Harmony search based remodularization for object-oriented software systems. Comput Lang Syst Struct 2017;47:153–69. doi:10.1016/j.cl.2016.09.003.
[48] Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 2011;1(1):3–18. doi:10.1016/j.swevo.2011.02.002.
[49] Garg H. A hybrid PSO-GA algorithm for constrained optimization problems. Appl Math Comput 2016;274:292–305. doi:10.1016/j.amc.2015.11.001.