Information and Software Technology 123 (2020) 106292

Contents lists available at ScienceDirect

Information and Software Technology

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

Orientation-based Ant colony algorithm for synthesizing the test scenarios in UML activity diagram

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

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

ARTICLE INFO

Keywords:
Soft computing
Ant colony
UML activity diagram
Concurrency
Computational Intelligence
Test Scenario

ABSTRACT

Context: The model-based analysis is preferred over the code-based analysis as it speeds up the development process and directs the guiding effort. In the software industry, the Unified Modeling Language (UML) is a set standard followed by the developers as well as system analysts to extract all attainable paths of controls, usually known as scenarios under an activity diagram.

Objective: In this manuscript, a bio-inspired methodology has been applied on concurrent sub-part of a UML activity diagram to fetch various feasible test scenarios.

Method: The food search pattern of an ant has been taken as a base heuristic. An orientation factor has been introduced in the existing ant colony optimization algorithm. Experiments have been performed using three student projects, five synthetic models and an openly available model repository named LINDHOLMEN data-set at Github.

Results: The statistical analysis has validated the results obtained through various existing approaches and the proposed approach. Experimentation shows that the orientation-based ant colony algorithm has produced better results as compared to the existing Genetic Algorithm (GA) and Ant Colony Optimization (ACO) on the basis of feasible test scenarios generated.

1. Introduction

Keeping in view the utility of object-oriented analysis and design practices, the community of software developers and system software analysts have recommended their use [1]. The said practices lower the cost incurred for developing and maintaining the software. UML system was introduced in the year 1996 to aid the practices of object orientation. UML depicts a variety of views in the software development process; and it can be employed for traceability analysis, software maintenance, testing, *etc.* [2]. A UML Activity Diagram (AD) outlines specific system behavior *w.r.t.* Control flow constructs. UML activity diagram is good enough to depict the control flow in an object-oriented system [3].

A Use-case is a type of diagram in UML that depicts functional requirements with oval shape constructs. Each oval in a Use case diagram can be expanded further as an Activity diagram, where a sequence of activities is referred to as a scenario or test scenario.

In the light of software development process, validation and verification, or software maintenance, every scenario gives a path of activities that occurs in a system. In a UML activity diagram, it is required to cognize all feasible start-to-end paths with all associated control entities for testing a use-case sufficiently. Various control flow entities include choice node, concurrency, looping, *etc.* that select the order of execution for different events. For depicting concurrency in a UML activity diagram, a fork-join pair has been used. Fork means beginning of more than one parallel control flows. A Join node is related to electronic “AND” gate that specifies when all the incoming flows to a specific join node are received, only then the final control flow will get directed on out-edge from a join element. The concurrent behavior implies non-determinism, which lets the control switch after any node of the independent subsequences of activity nodes present between a fork-join entity. This kind of interleaved control switching leads to permutation amongst activity nodes, which results in an exponentially huge number of end-to-end test scenarios generated between fork-join node. Thus, creating test scenarios for fork-join construct in AD can be considered as a problem under the category of combinatorial optimization. As per the systematic review undertaken by Arora et al. [4], about 179 well-studied research papers and inferential interpretations related to non-determinism and interleaved activity execution in UML models, and concurrent construct in model-based testing advocate that generating test sequence (or scenario) is mainly accompanied by the exact algorithms. The outcome of survey advocates that there lies the magnificence of scope for using the meta-heuristic practices to generate the test scenarios for concurrent sub-section in the UML activity diagram. When commonly exercised ex-

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

https://doi.org/10.1016/j.infsof.2020.106292
Received 11 October 2019; Received in revised form 21 February 2020; Accepted 23 February 2020
Available online 1 March 2020

act approaches (or algorithms) become ineffective to regulate the cases *w.r.t.* size, structure stretching within the required search interval, the application of meta-heuristic algorithm is considered justifiable [5,6]. Researchers have anticipated multiple bio-inspired techniques in recent decades for generating test sequences for the activity nodes under fork-join in AD. Existing research work reported high redundancy in test scenarios created as final sequences. Also, high randomness resulted in low count of final test scenarios. This aspect has motivated for us to explore a customized meta-heuristic approach that provides an improved exploration of search space. The heuristic in Ant Colony Optimization (ACO) draws its inspiration from the food search mechanism of the ants Formicidae, in which incorporating an orientation (angle) factor in movement of ants for exploring the search space (finding paths) improved the findings for the feature under consideration and resulted in reduced duplicity, when compared to ACO, and GA. Orientation factor in ACO has been used and applied on a variety of subject systems, *viz.* student projects [7], openly available resource at Github named LINDHOLMEN data-set [8,9], and synthetic cases drafted for UML activity diagrams. The proposed approach has been compared with the existing ACO and GA based on the number of feasible sequences generated as output of the algorithm.

The section-wise division of the research manuscript is as follows: Already existing meta-heuristic approches to generate paths (or sequences) in AD using fork-join constructs have been listed in Section 2. Section 3 describes the scope of work under consideration, the main concept behind the area of our research and prime motivational factor for using Orientation-Based Ant Colony Optimization (OBACO). Section 4 explains the proposed approach using an orientation factor in Ant-based system. Experimental results, comparative analysis with the existing algorithms, and null hypothesis for statistical analysis are presented in Section 5. Threat to validity of OBACO has been described in Section 6. Finally, Section 7 presents the findings of the proposed work, and its application in allied domains.

2. Related work

Though vast literature and technical content are available on the multi-threaded code and white-box testing technique [4,10]; so far the algorithms available under the umbrella of exact methods cannot be applied for generating the scenarios (or sequences) under the concurrent construct of AD. This is since an intermediate graphical form, like dependency graph, control/data flow graph, etc. are an integral part of the exact algorithms. Information extraction from these intermediate graphical forms incurs a processing overhead. This extra load can be avoided by using Extensible Markup Language Message Interface (XMI) of the UML model under consideration [11–14]. In the software industry as well as for the student projects in academics, the test scenarios can be generated either from the source code at the time of development process or initial design level using UML activity diagram. Scenario generation from UML activity diagram at the initial design phase is always preferred over generating scenarios as less information is required to be processed at early phase of SDLC [15]. This enables software developers, testers and especially software system analysts to craft a useful development as well as testing plan, before starting the actual development process. This section of the manuscript discusses the status of existing research in the domain of meta-heuristics approaches and test scenario (or sequence) generation. Table 1 highlights the research contributions related to bio-inspired methods used for generating test sequences (or paths) from AD.

Observation: The research trend in the domain related to meta-heuristics and test sequence (or scenario) generation for UML activity diagram shows that researchers in the said area have missed the benchmark models in their discussion, and even not validated their proposed approach using existing standard models. Benefits from the gelling of two domains, *i.e.* meta-heuristics and test scenario generation are prominent when feasible test scenarios can be generated for real-life software projects easily. Accumulation of a chemical named “Pheromone” bound the ants in the ant colony system to follow some particular paths repeatedly; this results in the redundantly explored/traversed paths. Whereas for genetic algorithm, the chromosomal theory plays a vital role, and the internal factor of randomness here leads to the generation of very few routes under concurrency construct.

3. Scope of the present work and orientation-based ant colony approach

Fig. 1 depicts a sample UML activity diagram with a concurrency construct having two sequences of activity nodes between fork-join node, *viz.* *Activity* 5 → *Activity* 6 → *Activity* 7 and *Activity* 8 → *Activity* 9.

At the fork node in the UML activity diagram, on the out edge, the incoming flow of control generates the multiple concurrent outflows as output. Join node acts as its counterpart, where it produces a solo control flow as output when all incoming control streams are completed on it. Concurrency implies non-deterministic flow. At the fork node, more than one flows get initiated at once with control switching between the activity nodes present in adjacent sub-queues of the concurrent construct under consideration. The non-deterministic factor in control switching results in permutation for selecting next activity hop in test sequence (or scenario). Making the control switch deterministic will loss the inherent nature of concurrency for the nodes under fork-join.

Interleaved execution among the nodes of sub-queues under fork-join node has been depicted in Fig. 2(a); this resulted in a sequence explosion in the count of test sequences generated. Fig. 2(b) illustrates a tree-type structure giving a realistic view for all the feasible and infeasible test scenarios as created. The icon $\surd$ symbolizes feasible (or a valid) path and ✗ indicates in-feasible test scenario. Fig. 2(c) shows a set of valid test paths (or scenarios) generated from the sub-queues under fork-join nodes (*Activity* 5 → *Activity* 6 → *Activity* 7, and *Activity* 8 → *Activity* 9).

The count of test scenarios increases exponentially when permutations among activity nodes increase. And the same is directly influenced either by the number of sub-queues under concurrent construct, or the count of activity nodes in the sub-queues. Table 2 describes the combinatorial explosion *w.r.t.* the formula $\frac{(n_1+n_2+\ldots n_i)!}{(n_1!\times n_2!\times\ldots\ldots n_i!)}$, where, i is number of sub-queues; and n is the number of UML activity nodes present in the ith sub-queue [19].

Fig. 3 depicts a sample AD *w.r.t.* the cases mentioned in Table 2. Table 3 indicates that a waiting time of more than 7 min occurred after the execution of Depth-first Search (DFS) [3,26] on Case (d).

Application of the exact algorithms like DFS, BFS, *etc.* for the cases where node count in sub-queues and number of sub-queues is/are medium or large, is not practical due to high computational time [26].

Applicability of ACO for generating feasible test scenarios is quite convincing, but the number of test scenarios generated can be increased

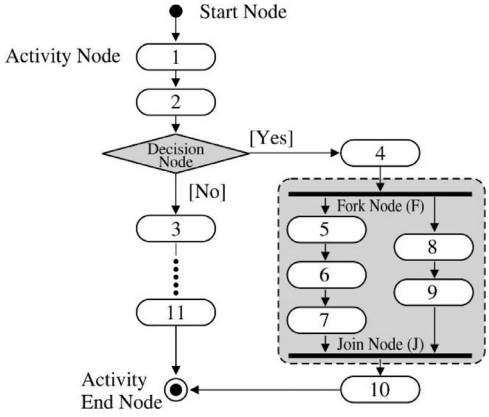

Fig. 1. UML activity diagram with a concurrency construct [25].

Table 1
Reported literature on generating test sequence (or scenario) with bio-inspired techniques.

S.No.	Year	Author(s)	Meta-heuristic/bio-inspired technique used	Hitch
1.	2005 2010	H. Li and C. P. Lam [16] C. P. Lam [17]	Anti-ant agents	The conversion of a UML activity diagram into a state machine representation involves significant overhead. Authors missed the discussion for scalability of the technique proposed.
2.	2008	U. Farooq, C. P. Lam, and H. Li [18]	Random walk approach	The system results in a set of noisy test sequences; when the wrong node gets selected, and symmetric flows go asymmetric.
3.	2012	M. Shirole, M. Kommuri, and R. Kumar [19]	Evolutionary algorithm (EA) [EA is a variation of GA]	Researchers haven't mentioned the way for selecting the values of different features used in their proposed approach — parameters, *viz.* selection of point for the crossover, percentage of crossover, percentage for selection. As the number of activity nodes under concurrent construct increases the chances for getting feasible paths become less due to EA's randomness factor at various stages.
4.	2014	A. Mishra and D. P. Mohapatra [20]	ACO	Authors missed the discussion on interleaved processing of activity nodes under the fork-join entity. Here, instead of using Ant colony optimization for test scenario generation, researchers used it for test sequence prioritization. Validation of the proposed approach was not provided using some sample model with fork-join node(s).
5.	2015	F. Sayyari and S. Emadi [21]	Markov chain and ACO	Researchers haven't taken into account the concurrent entity of UML models. To validate the proposed technique, no set standard benchmark models have been taken.
6.	2016	V. Panthi and D. P. Mohapatra [22]	ACO with Depth First Search (DFS), Breadth First Search (BFS)	The ant-based system has been used to prioritize test cases instead of test case/sequence/scenario generation. Exact algorithms, *viz.* DFS, and BFS are used to traverse an intermediate graphical construct named Activity Interaction Graph (AIG). For validating the proposed hybrid approach of ACO with DFS and BFS, no standard benchmark models have been taken. The subject system under consideration is having very less number of elements under concurrent entity in AD.
7.	2016	A.A. Kyaw and M.M. Min [23]	GA	Interleaving among the activity nodes of the sub-sequences under fork-join has not been discussed. Detailing about UML activity diagrams taken as the subject system has been missing.
8.	2018	G. Bhattacharjeeand S. Dash [24]	Hybridization of ACO and GA	Although the researchers have discussed the realistic UML activity diagram as an example, but they failed to cater the scenarios, *viz.* single fork and multiple joins, nested fork-join, *etc.*

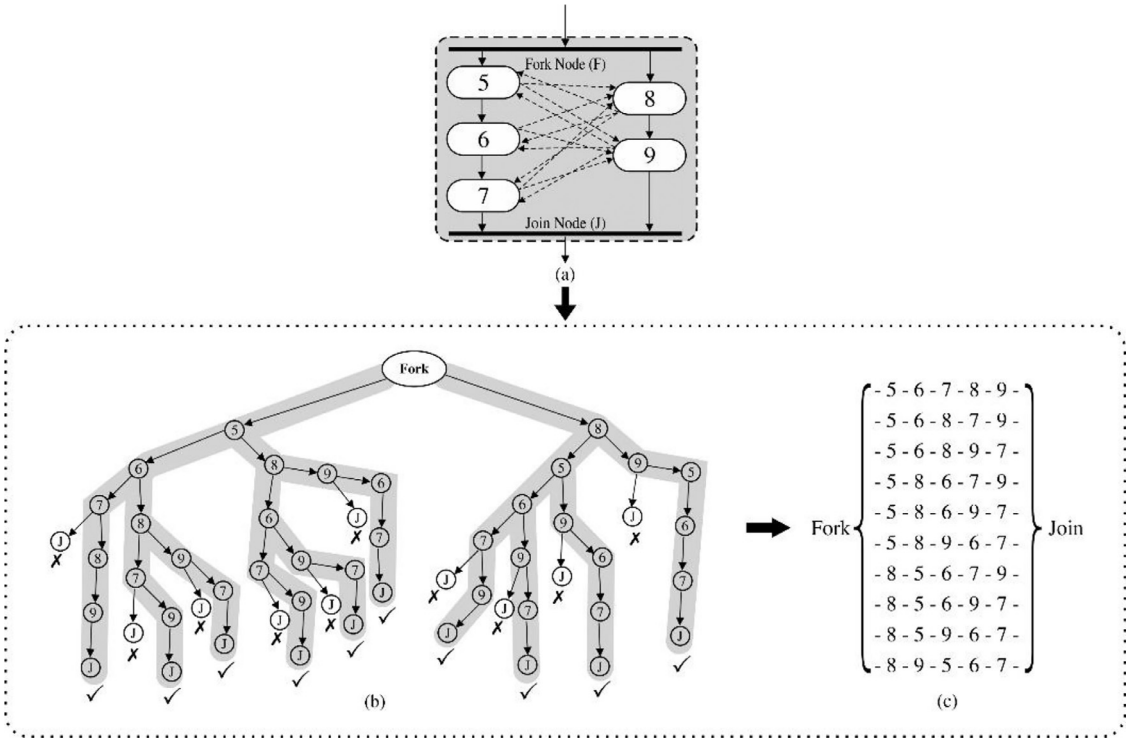

Fig. 2. (a) Selected sub-part from UML activity diagram having concurrency, (b) exploratory tree structure resulted from non-deterministic execution among sub-queues under fork-join, (c) final set of feasible scenarios.

Table 2
Combinatorial explosion in the count of test scenarios generated for different sample values for size and count of sub-queues under concurrent construct, and count of elements in individual sub-queue [25].

AD given in Fig. 3	Sub-queues under concurrent construct (fork-join)	Number of activity nodes in each sub-queue under fork-join	No. of feasible test scenarios computed by formula [19]
Activity Diagram (a)	2	$queue_1 = 3$, $queue_2 = 2$	10
Activity Diagram (b)	2	$queue_1 = 5$, $queue_2 = 4$	126
Activity Diagram (c)	3	$queue_1 = 5$, $queue_2 = 4$, $queue_3 = 3$	27,720
Activity Diagram (d)	4	$queue_1 = 5$, $queue_2 = 4$, $queue_3 = 3$, $queue_4 = 2$	2,522,520

Table 3
Number of Test Scenarios (NTS) and execution time (ET) of algorithm obtained [in millisecond] after applying DFS as an exact algorithm; and ACO, and GA as meta-heuristic algorithms deployed for sample AD of Fig. 3. [Here, the population parameter for GA or count of ants in ACO has been taken as 100. The count of generations in GA or count of trials for ACO has been considered as 1000]. The average results achieved after performing the above said approaches with 30 instance runs have been listed here.

UML Activity diagram (taken from Fig. 3)	DFS [26]		ACO [27]		GA [19]	
	NTS	*ET(ms)*	*NTS*	*ET(ms)*	*NTS*	*ET(ms)*
Activity Diagram (a)	10	1.5	10	418	10	117
Activity Diagram (b)	120	13	57	928	0	30
Activity Diagram (c)	27,720	1149	610	2412	0	26
Activity Diagram (d)	No result after wait of 7 min	No result after wait of 7 min	1424	4499	0	28

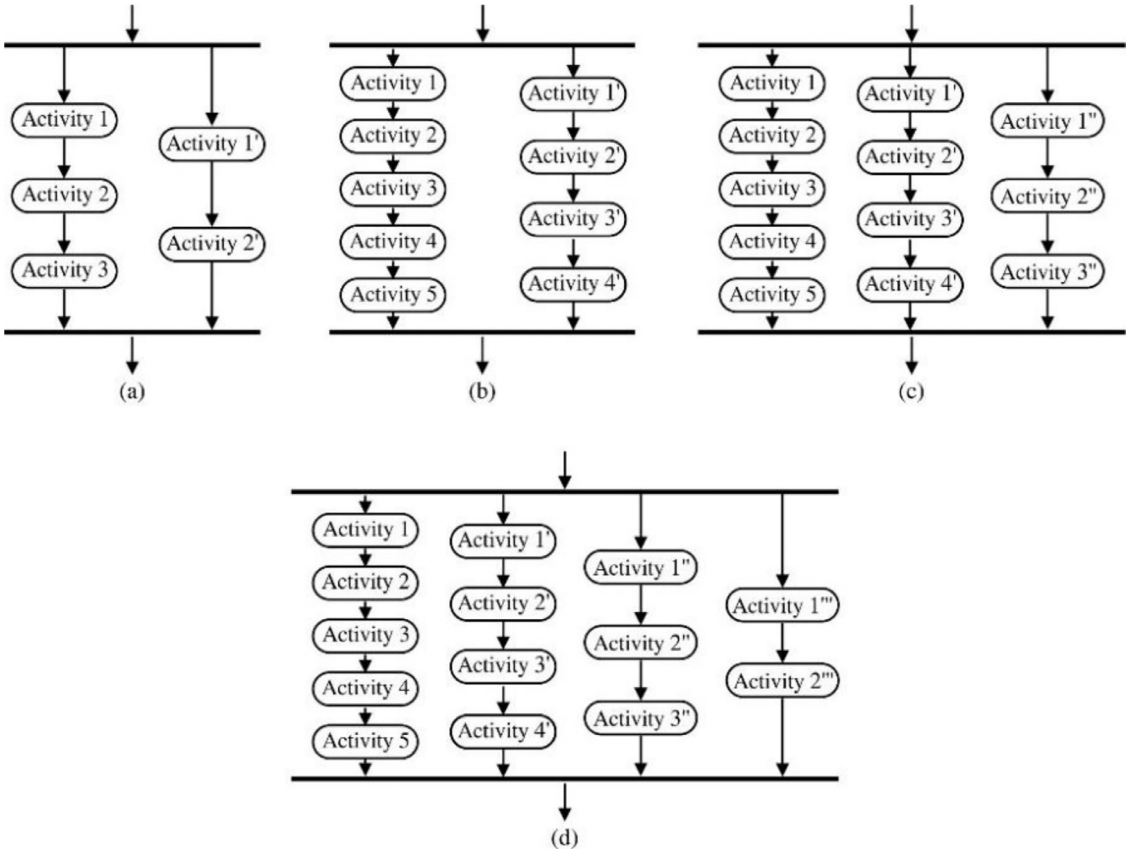

Fig. 3. UML activity diagrams with a variable number of activity nodes and sub-queues under the concurrent element [25].

by customizing it where an ant agent happens to be less redundant for following the same path again [27]. This redundancy in path traversal is due to high accumulation of pheromone chemical, which attracts ants to follow a particular trail. For sample activity diagrams b, c, and d; the Genetic Algorithm (or EA) is not providing any test scenario in the result. This is because the inherent randomness factor of GA influences the generation as well as the selection of feasible test scenarios, and provides vague test scenarios after interleaved execution [19].

3.1. Ant colony system and inspiration for using it

Path taken by the ants from their nest to a food source is tentatively the shortest among the set of available paths. The researchers

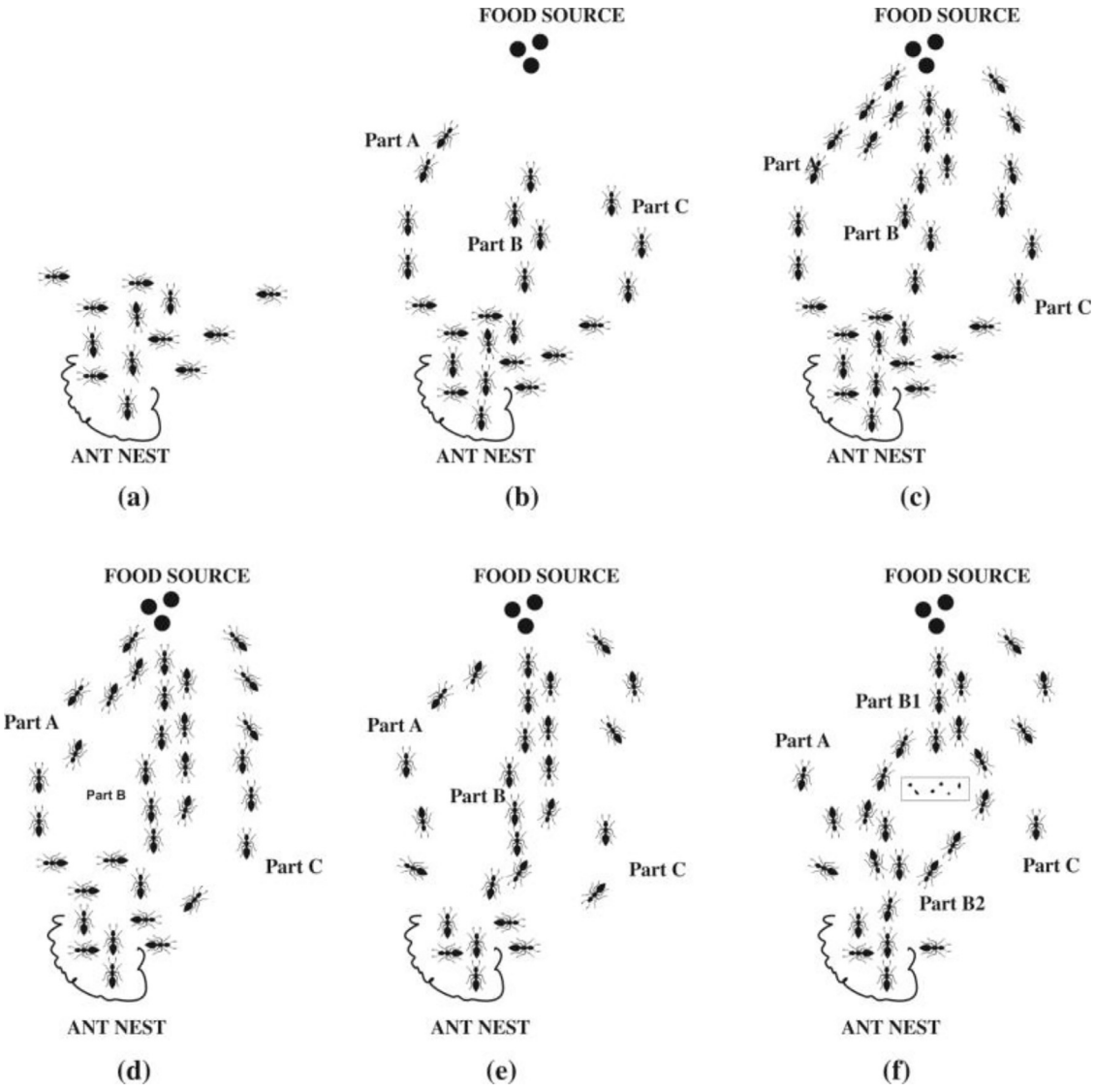

Fig. 4. Schematic of foraging behavior in ACO [30].

have found that ants follow the same path by tracing the pheromone laid by them. Further, Deneubourg et al. [28] through their research on biological Ants supported that the probability for following a path is dependent on the pheromone level as well as the heuristic factor. Dorigo et al. [29] proposed the usage of the ant analogy related to the behavior of searching for the food in optimization algorithms; and it's a fusion of heuristic, positive feedback, and distributed computation.

Fig. 4(a) and (b) show the random selection and traversal of the possible route from the initial point (ant nest) to the food source (destination). Fig. 4(c) depicts the backward route from food source to the initial nest position. Ants form "pheromone trails" by depositing a chemical substance called "pheromone". Trailing of pheromone by other ants in the colony has been displayed in Fig. 4(d). As shown in Fig. 4(e), successive ants smell the deposited chemical and traverse the route marked by rich pheromone concentration. In Fig. 4(f), traversal of the route by the ants is self-adjustable and correctable, even in the presence of unforeseen obstacles. The collective behavior of ants increases the tendency of path selection with the count of ants having selected a specific path in the previous steps. The Ant System (AS) uses intelligence gained by one ant/artificial agent during its search process for constructing the next feasible (or optimal) solution in the domain of the path-traversing problem.

There exists a similarity between pheromone-influenced route traversal from ants' nest to source of food in ant system, and the traversal of test sequences (or scenarios) that get created while interleaved control switching over the activity nodes under fork-join construct. Where interleaving over UML activity nodes results in a tree type structure. This existing similarity is the prime motivational factor for using ant colony algorithms to generate test scenarios (or paths) for concurrent construct.

Fig. 5 depicts a single ant colony and food source structure that gets created for multiple paths due to food foraging behavior of ants in a specific colony. Fig. 6 shows all practically feasible and unfeasible paths generated for concurrent sub-section of UML activity diagram (Fig. 1). The icon $\surd$ symbolizes a feasible path in actual; ✗ denotes an unfeasible route. In Fig. 6, color variations from dark to light at marked tubular structure represent the path with varying deposition of pheromones. Higher pheromone deposition leads to a redundant traversal of the same route by the ant-like agents present in the ant colony.

3.2. Mathematical model for using ant-based method

In literature, the first reported Ant Colony Optimization algorithm has been referred as Ant System (AS) [32] which can be deployed for

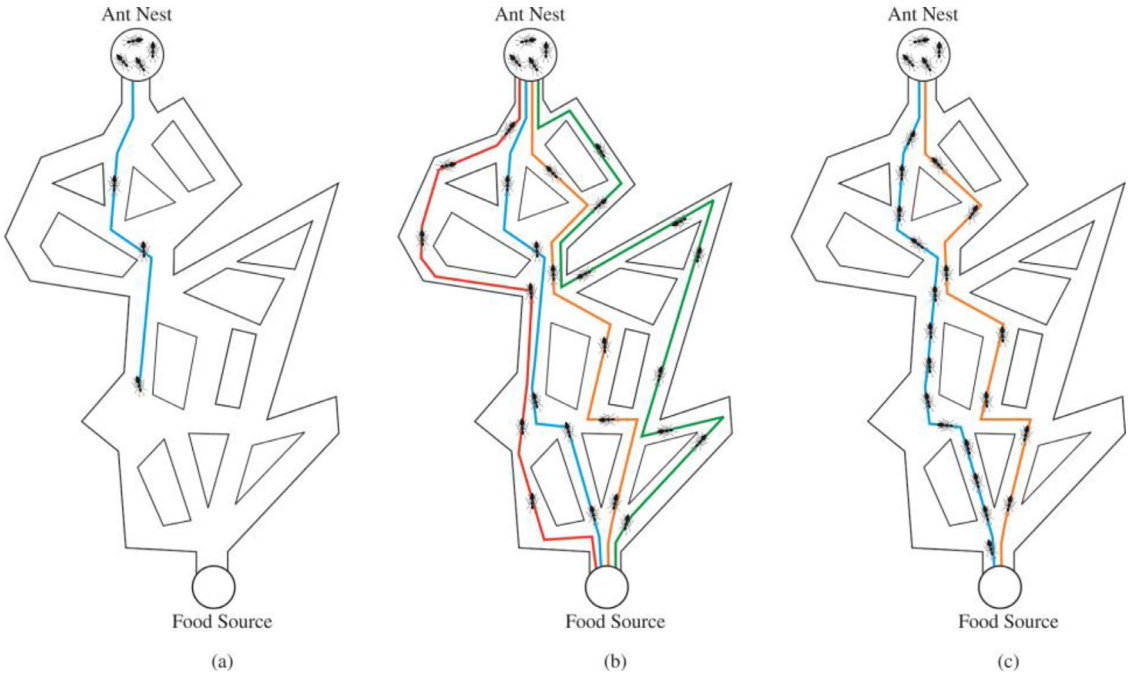

Fig. 5. (a)–(c): Path navigation by Ants from Ant colony (or nest) to source of food [30,31].

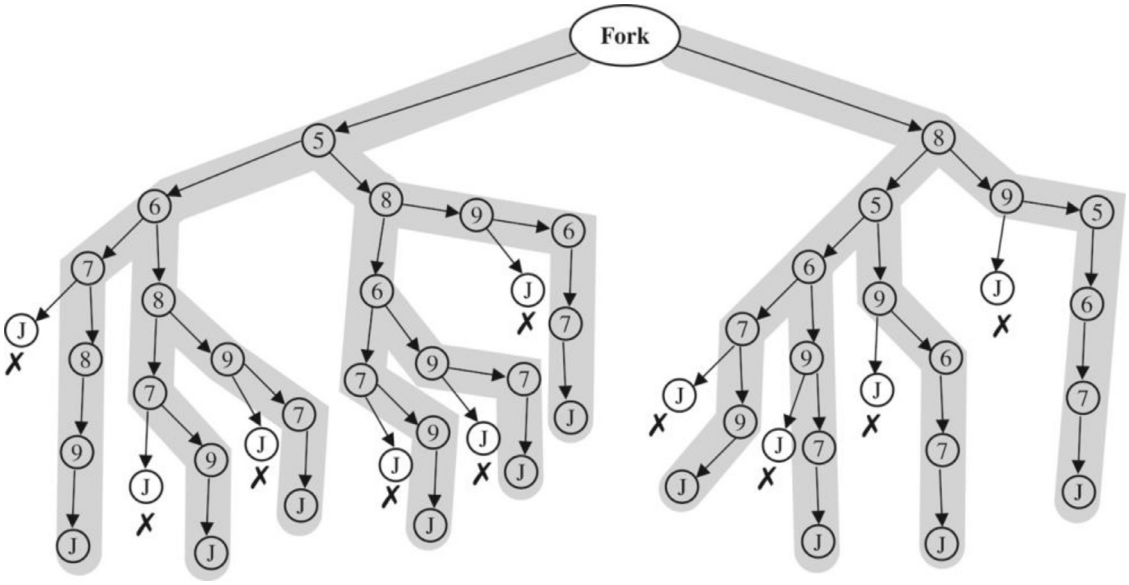

Fig. 6. Exploratory tree structure depicting the scenarios generated from permutations among the nodes of concurrent part in Fig. 1.

a vast variety of problems. In AS, an ant k at node i selects the next possible node j with probability computed through the use of random proportional rule. It can be defined as

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{h \in N^k} [\tau_{ih}]^{\alpha} \cdot [\eta_{ih}]^{\beta}} \tag{1}$$

where, τ_{ij} is the amount of pheromone on edge (i, j); α is a parameter that controls the influence of τ_{ij}; η_{ij} is termed as the desirability of edge (i, j); β is a parameter which manages the influence of η_{ij}. N^k denotes a possible set of nodes, excluding the ones already visited upto the current position/node of ant k; and this can be limited further to a set of nodes taken as nearby to node i. For lowering the impact of heuristic, the probability factor is inversely proportional to the heuristic value. Here, the parameter that controls the influence of η_{ij} is taken as $(-\beta)$, *i.e.,* customized value for the β. This changes Eq. (1) into Eq. (1a) as given below:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{-\beta}}{\sum_{h \in N^k} [\tau_{ih}]^{\alpha} \cdot [\eta_{ih}]^{-\beta}} \tag{1a}$$

The pheromone trail values are updated as

$$\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

where, ρ is the evaporation rate taken for the system; and $\Delta\tau_{ij}^{k}$ is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} F(k) & if\ edge\ (i,j) is\ sub-part\ of\ the\ route\ taken\ by\ ant\ k, \\ 0 & Otherwise \end{cases}$$

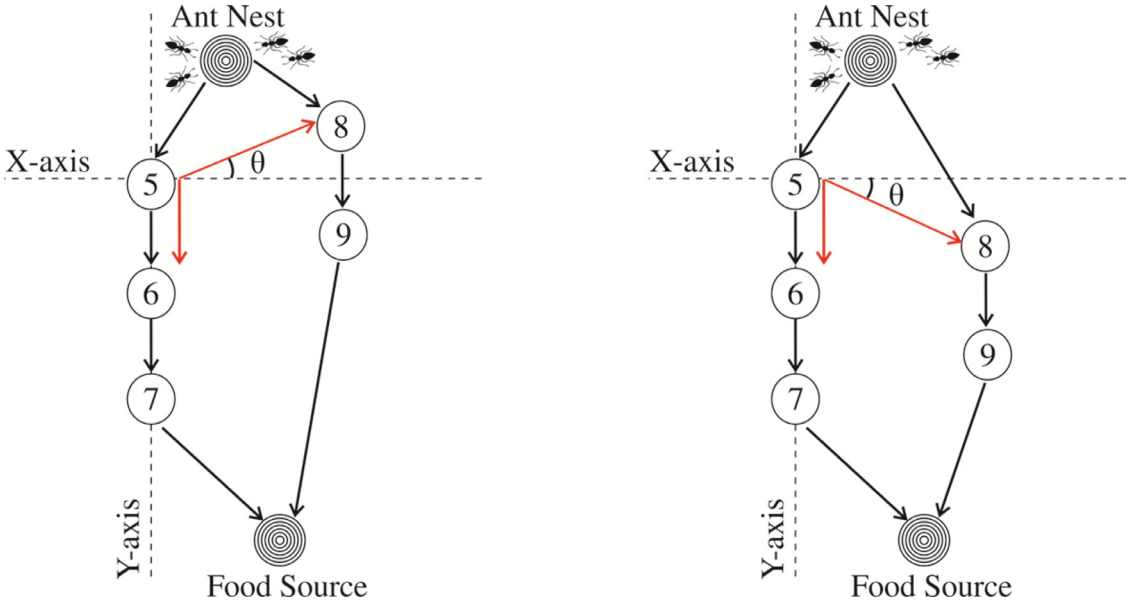

Fig. 7. Orientation as an influential factor by ant traversal.

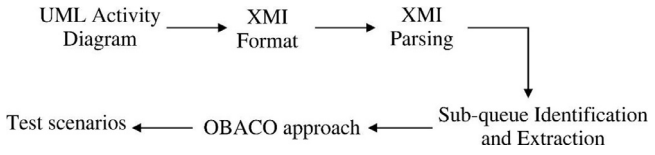

Fig. 8. Steps for generating test scenarios (or sequences) taking UML AD.

(2a)

where, $F(k)$ indicates the quantity of the pheromone placed on the edges of the route created by a k^{th} ant. The value for $F(k)$ feature is reciprocal of the cost incurred by ant k to construct the route; and it is taken as multiple of constant Q. The better solution implies greater pheromone deposition by an ant [33]. A different version of AS-based algorithms has been proposed by various researchers that differ in a few parameters like *Construction of Solutions* and procedure for *Updating Pheromone* [32].

3.3. Orientation factor in ant-based system and its adaptation

Experimental studies have shown that ACO is a powerful approach for finding the best solution, but it suffers from long search time and falling into local optimal for a large-scale computation. Initially, each agent in ACO moves randomly; and the amount of information is almost the same on each route available for traversal. For an effective path, the information symbolizing the amount of pheromone increases gradually through a positive feedback.

As shown in Fig. 7, the intensity of pheromone chemical at the edge from the current activity node and approximate next activity node decides the final node using a basic variant of ACO. In case the pheromone intensity along all edges among the present activity node and the activity nodes from the set of likely next nodes is the same; the ant cannot finalize between the edges with some difference in probability factor. For such cases, an orientation factor has been gelled with pheromone factor and used for computing the likelihood to select the next node. The proposed orientation-based ant colony optimization takes the level of pheromone chemical with Cosine of the angular factor (θ) between the present node and the subsequent activity node.

$$P'_{ij} = P_{ij} \times \frac{1}{Cos\ \theta} \tag{3}$$

However, in the case of an extensive search space, it is cumbersome to locate possible paths using a simple ant colony approach. Hence, by adding an orientation factor in the movement of ants, the count of feasible paths can be increased as compared to simple AS. Repetitive simulations proved that an orientation factor given in Eq. (3) above enhances the count of desired paths for the problem under consideration. The angle θ is orientation between X-axis and the vector taken from $N_{current}$ to the candidate node $N_{nexthop}$. As shown in Fig. 7, the orientation can be made between X-axis and the vector directed from the current node $N_{current}$ to the candidate node $N_{nexthop}$. The orientation is taken either for the node(s) having a parent-child relationship with its predecessor node, or for the one(s) which are not having a direct parent-child relationship, but whose parent node has already been visited.

The $Cos\,\theta$ is representing the cosine value between the two orientation vectors v_i and the X_{axis}. Using the formula for the dot product as $Cos\ \theta = v_i.X_{axis}/(|v_i|.|X_{axis}|)$, we can compute the cosine of angle directly. The angle factor impacts the choice for determining the next node rather than falling into local optima. After incorporating the orientation factor, Eq. (1a) gets converted to Eq. (4) as hereunder:

$$p'_{ij} = \frac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{-\beta}}{\sum_{h\in N^k}[\tau_{ih}]^{\alpha}.[\eta_{ih}]^{-\beta}} \times \frac{1}{Cos\ \theta} \tag{4}$$

The values for p_{ij}, and τ_{ij} are obtained by using Eqs. (1a) and (2); and each p'_{ij} is computed by adding an orientation factor, $(Cos\ \theta)^{-1}$ to p_{ij} while selecting the next node from a set of nodes. The approach presented here has adapted Eq. (4), and progresses using variable p'_{ij}. Depending on the value of probability, the ant-based system retains few edges, and droop many in the course of ant traversal. It is acknowledged that the system generates test scenarios by selecting the next neighbor node resulting into a bridge between two specific nodes, *i.e.*, $N_1(ant\ nest)$ and $N_2(food\ source)$. Here, the modified ant-based approach applies heuristic values and orientation factor to choose the next node among a set of activity nodes. The proposed approach produces a route from ant nest to food source by incremental traversal for each transitional edge.

The concept of using angular orientation in ant colony optimization technique has previously been used in varied domains, *viz.* multicast routing in multimedia networks [34], transport network [35], path planning for Unidentified Arial Vehicle (UAV) [36], multi-path routing [37], harmonic elimination [38], probabilistic Traveling Salesman Problem (TSP) [39], *etc.* However, to the best of our knowledge, the orientation-

Table 4
Values of various parameters in the proposed ant-based approach where values have computed using CRS-Tuning approach [40].

S.No.	Name of the associated parameter	Value
1.	Pheromone τ_{ij}	0.8
2.	Evaporation rate ρ	0.2
3.	Heuristic η_{ij}	2.0
4.	Alpha α	1.1
5.	Beta β	1.0
6.	Constant Q or Q"	3.0
7.	Distance factor between two neighbor nodes d_{ij}	1.0
8.	Orientation value θ	25° / 0°

based ant colony optimization technique is yet to be adequately investigated for imbibing its capacities to generate test scenarios under fork-join in UML activity diagrams.

4. Orientation-based proposed methodology

In this section, Fig. 8 highlights the phases for obtaining scenarios taking UML activity diagram as input. Afterwards, it defines the method used in current research work. IBM Rational Software Architect (RSA) has been used here for drawing UML activity diagram as well as to export in its specific Extensible Markup Language Message Interface (XMI) format.

Parsing of XMI code results into individual sub-queues of activity nodes present between fork-join node. Now, orientation based ant colony optimization is used for generating various combinations between the activity nodes of the sub-queues as extracted in the previous step. Some of the main symbols used all over the manuscript as well as in the proposed approach are defined as hereunder:

SQ_i	Sub-queues extracted after parsing XMI code for *fork-join* structure/s
a_k	kth ant
A	Set of n ants
A^c	cth set/colony of ants
$\mathcal{N}_i$	Set of neighboring nodes for ith node
p_{ij}	Probability of elements present under the set $\mathcal{N}_i$
τ_{ij}	Pheromone concentration at the edge between node i to j
$\Delta\tau_{ij}$	Small change in the value of pheromone for the edge from node i to j
ρ	Rate of pheromone evaporation
η_{ij}	Heuristic factor on the edge from node i to j
Q *or* Q''	Multiplying factor for heuristic value
j_{a_k}	Next node for ant a_k at ith node, whose probability has been selected from the set p_{ij}
α	Factor that controls the pheromone level
β	Factor that controls the heuristic value
d_{ij}	An identifier related to distance between current node (i) and the next node (j)
CSq	Count of sub-queues from fork node
S_g	gth sequence for an ant from set A
TS_s	Full set of final sequences/scenarios
θ	Orientation between the current node and the next one selected to be jumped upon

4.1. Procedure for generating test scenarios under concurrent segment

This section outlines the proposed methodology which uses Orientation-based ant colony to generate test sequences (or paths) for a concurrent segment of UML AD. Input to the proposed algorithm (Algorithm 1) named *Orientation-based Ant Colony Optimization (OBACO)* is sub-queues (SQ_i) under the fork-join construct of an AD. A^c is a set of ants belonging to a specific cth colony having m ants, *i.e.* $A^c = \{a_1, a_2, a_3, \ldots\ldots\ldots\ldots a_m\}$. The main iterative construct in the algorithm commences from fork node with a single ant a_k at a time taken from a specific ant colony. At the ith current position, a set ($\mathcal{N}_i$) of the feasible neighborhood is computed, whose parent node(s) has/have already been visited. The probability value $p_{ij} = \{p_{i1}, p_{i2} \ldots\ldots\ldots, p_{id}\}$ is computed for all the elements in $\mathcal{N}_i$. The next activity node is decided by pheromone, heuristic values, and orientation factor. Here, orientation can be taken between X-axis and the trajectory taken between the present ith node *i.e.* $N_{current}$ to the next candidate node from $\mathcal{N}_i$. After selecting a specific jth activity node, the values for pheromone as well as heuristic are updated for the selected one using the formula $\tau_{ij} = \{(1-\rho)\times\tau_{ij}^k\}^{\alpha} + \{Q''\times(\frac{1}{d_{ij}})\}^{-2\beta}$. The selected jth node will be marked for ant a_k and added into a set S_g denoting a particular sequence for an ant from a specific colony. Now, the node j turns out to be the current node as i, only when, j is not a join node. After one complete execution of inner $do\ldots While$ loop, one end-to-end sequence between the fork and join node is generated; and the same is considered as a test sequence/scenario generated by an ant a_k. Outer "*for*" loop gives a set of test sequences (TS_s) as formed by m ants available in a specific colony. Following the steps as mentioned above, the proposed algorithm results into a set of scenarios that are feasible, by neglecting the infeasible paths.

4.2. Complexity of the proposed algorithm

Consider an Ant Colony system, where, a_k denotes a single Ant, A^C represents one full colony of m Ants and u denotes count of the colonies. The Eqs. (5) and (6) represents single ant colony, and a set of ant colonies respectively.

$$A^C = \sum_{k=1}^{m} a_k \tag{5}$$

$$Set\ of\ Ant\ Colonies\ (S) = \sum_{C=1}^{u} A^C \tag{6}$$

If computational cost for a single Ant a_k is taken as constant c, then computational cost for a single ant colony is $A^C = \sum_{i=1}^{m} c_i = mc$ and the computational cost for a set of u colonies be $\sum_{col=1}^{u} mc = u \times mc = umc$. Now, for computing the complexity there exist three cases:

Case I if $u \gg m$
then the complexity is $O(u)$
Case II if $u \ll m$
then the complexity is $O(m)$
Case III if $u \cong m$
then the complexity will be $O(um) = O(u^2) = O(m^2)$

4.3. A model run using OBACO algorithm

Using a single ant agent, Fig. 9(a)–(e) depicts the model execution of the OBACO using the concurrent sub-section of a UML AD as illustrated in Fig. 1 (commencing at fork node and ending at join node). In Fig. 9, the co-ordinate axis (X and Y-axis) are represented by the dashed lines.

Table 4 presents the different factors and their respective values considered for simulating OBACO approach. Values for various parameters listed in the said table have been obtained by using CRS-tuning approach provided in [40]. The labels A, J, and N in the column header of Table 5 denote current node, Next node to be jumped upon, and Next probable node(s) respectively for an ant agent. The table also shows the intermediate computations involved when an ant agent initiates from the fork node and reaches up to join node by an incremental traversal over the intermediate activity nodes. An ant $a_k \in A^c$ starts its traversal from fork node, where it chooses the next directly connected node at random (either Node 1 or Node 4). Here, let us assume that Ant1 has chosen Node1. From the current node, a set constituting feasible neighbors has been computed ($\mathcal{N}_i$). The higher value of probability (calculated for the set $\mathcal{N}_i$) will provide the node on which the control will jump upon. Once the next node has been selected, the value of pheromone, as well as a

Algorithm 1
OBACO – Orientation-based Ant Colony Optimization.

Input: Sub-queues under fork-join(SQ_i), where $i = \{1, 2, 3 \dots n\}$
Output: Test scenario(s) generated(TS_s): $ts_1,\ ts_2,\ \dots ts_m$
Data: $TS_s = 0$, A set of test scenarios which is initially empty
$S_g = 0$, Set of activity nodes for a particular (gth) sequence generated by an ant, which is initially empty
Begin
1. **Let** $A^c = \{a_1\ ,\ a_2\ ,\ a_3\ , \dots\dots\dots\dots a_m\ \}$ be a set of ants at fork node for a cthcolony
Where, c= $\{1,\ 2,\ 3, \dots\dots\dots\dots u\}$.
2. **for each** $a_k \varepsilon A^c$
3. {
4. Start from *fork* node the traversal of ant a_k at ith position
5. **do** {
6. Compute the set of elements having jth next node(s) for a_k, where all the parent nodes for the j have already been visited. Denote this set by $\mathcal{N}_i$.
7.
8. Compute the probability of elements under $\mathcal{N}_i$ using the following Equation, where, $j\ \varepsilon \mathcal{N}_i$ and $p_{ij} \neq 0$

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{-\beta}}{\sum_{i\varepsilon\mathcal{N}_i}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{-\beta}} \times \frac{1}{Cos\theta}$$

Here, $\mathcal{N}_i$ is the set of feasible neighborhood for kth ant, when being at ith node.

$$\theta = \begin{cases} 25^0 & when\ node\ i\ and\ j\ are\ from\ different\ sub-queues \\ 0^0 & Otherwise \end{cases}$$

9. From ith node, let the probability values computed for $\forall j\varepsilon\mathcal{N}_i$ are
10. $p_{ij} = \{p_{i1},\ p_{i2}, \dots\dots\dots, p_{id}\}$
Select the jth next node as per the following rule
11. $$j_{a_k} = \begin{cases} max.\ value\ from\ set\ p_{ij}\ if\ there\ exists\ a\ variation\ in\ elements\ of\ set\ p_{ij} \\ any\ random\ value\ from\ p_{ij}\ Otherwise \end{cases}$$
12. Update the pheromone and heuristic for the edge connecting current node and selected jth node for ant a_k

$$\tau_{ij} = \{(1-\rho)\times\tau_{ij}^k\}^{\alpha} + \{\Delta\tau_{ij}^k\}^{-2\beta}$$
$$\tau_{ij} = \{(1-\rho)\times\tau_{ij}^k\}^{\alpha} + \{Q''\times(\tfrac{1}{d_{ij}})\}^{-2\beta}$$
$$where,\ Q'' = \begin{cases} \frac{Q}{CSq} & when\ i\ is\ parent\ of\ j\ and\ it\ is\ from\ the\ same\ sub-queue \\ Q\ otherwise \end{cases}$$
$$\eta_{ij} = Q''\times\eta_{ij}^k$$

13. $S_g = S_g \cup j$th node
14. Now, make node j as current node $i \leftarrow j$
15. } (**While** *current node* $i \neq$ *join node* for current *fork-join*)
16. $ts_i \leftarrow S_g$
17. $TS_s = TS_s \cup ts_i$
18. }
End

Table 5
Computations involved while performing the traversal of activity nodes from fork to join node using two ant agents (1 and 2) taken from a specific colony.

Ant No.	A	N	Computation Involved	J	Update of Pheromone and Heuristic of selected node
1.	5	{6, 8}	$P_{56} = \frac{(\tau_{56})^{1.1}\times(\eta_{56})^{-1}}{[(\tau_{56})^{1.1}\times(\eta_{56})^{-1}]+(\tau_{58})^{1.1}\times(\eta_{58})^{-1}} \times \frac{1}{Cos0^0} = 0.5$ $P_{58} = \frac{(\tau_{58})^{1.1}\times(\eta_{58})^{-1}}{[(\tau_{56})^{1.1}\times(\eta_{56})^{-1}]+(\tau_{58})^{1.1}\times(\eta_{58})^{-1}} \times \frac{1}{Cos25^0} = 0.552$	8	$\tau_{58} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{58} = 3\times 2 = 6$
	8	{6, 9}	$P_{86} = \frac{(\tau_{86})^{1.1}\times(\eta_{86})^{-1}}{[(\tau_{86})^{1.1}\times(\eta_{86})^{-1}]+(\tau_{89})^{1.1}\times(\eta_{89})^{-1}} \times \frac{1}{Cos25^0} = 0.552$ $P_{89} = \frac{(\tau_{89})^{1.1}\times(\eta_{89})^{-1}}{[(\tau_{86})^{1.1}\times(\eta_{86})^{-1}]+(\tau_{89})^{1.1}\times(\eta_{89})^{-1}} \times \frac{1}{Cos0^0} = 0.5$	6	$\tau_{86} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{86} = 3\times 2 = 6$
	6	{7, 9}	$P_{67} = \frac{(\tau_{67})^{1.1}\times(\eta_{67})^{-1}}{[(\tau_{67})^{1.1}\times(\eta_{67})^{-1}]+(\tau_{69})^{1.1}\times(\eta_{69})^{-1}} \times \frac{1}{Cos0^0} = 0.5$ $P_{69} = \frac{(\tau_{69})^{1.1}\times(\eta_{69})^{-1}}{[(\tau_{67})^{1.1}\times(\eta_{67})^{-1}]+(\tau_{69})^{1.1}\times(\eta_{69})^{-1}} \times \frac{1}{Cos25^0} = 0.552$	9	$\tau_{69} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{69} = 3\times 2 = 6$
	9	{7}	$P_{97} = \frac{(\tau_{97})^{1.1}\times(\eta_{97})^{-1}}{[(\tau_{97})^{1.1}\times(\eta_{97})^{-1}]} \times \frac{1}{Cos25^0} = 1.103$	7	$\tau_{97} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{97} = 3\times 2 = 6$
	7	Join	**Final Sequence as obtained by Ant-1** is *Fork* → 5 → 8 → 6 → 9 → 7 → *Join*		
2.	5	{6, 8}	$P_{56} = \frac{(\tau_{56})^{1.1}\times(\eta_{56})^{-1}}{[(\tau_{56})^{1.1}\times(\eta_{56})^{-1}]+(\tau_{14})^{1.1}\times(\eta_{14})^{-1}} \times \frac{1}{Cos0^0} = 0.768$ $P_{58} = \frac{(\tau_{58})^{1.1}\times(\eta_{58})^{-1}}{[(\tau_{56})^{1.1}\times(\eta_{56})^{-1}]+(\tau_{58})^{1.1}\times(\eta_{58})^{-1}} \times \frac{1}{Cos25^0} = 0.256$	6	$\tau_{56} = [(1-0.2)\times 0.8]^{1.1} + [1.5]^{-2(1)} = 1.05$ $\eta_{56} = 1.5\times 2 = 3$
	6	{7, 8}	$P_{23} = \frac{(\tau_{67})^{1.1}\times(\eta_{67})^{-1}}{[(\tau_{67})^{1.1}\times(\eta_{67})^{-1}]+(\tau_{68})^{1.1}\times(\eta_{68})^{-1}} \times \frac{1}{Cos0^0} = 0.5$ $P_{68} = \frac{(\tau_{68})^{1.1}\times(\eta_{68})^{-1}}{[(\tau_{67})^{1.1}\times(\eta_{67})^{-1}]+(\tau_{68})^{1.1}\times(\eta_{68})^{-1}} \times \frac{1}{Cos25^0} = 0.552$	8	$\tau_{68} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{68} = 3\times 2 = 6$
	8	{7, 9}	$P_{87} = \frac{(\tau_{87})^{1.1}\times(\eta_{87})^{-1}}{[(\tau_{87})^{1.1}\times(\eta_{87})^{-1}]+(\tau_{45})^{1.1}\times(\eta_{45})^{-1}} \times \frac{1}{Cos25^0} = 0.552$ $P_{89} = \frac{(\tau_{89})^{1.1}\times(\eta_{89})^{-1}}{[(\tau_{67})^{1.1}\times(\eta_{67})^{-1}]+(\tau_{89})^{1.1}\times(\eta_{89})^{-1}} \times \frac{1}{Cos0^0} = 0.5$	7	$\tau_{87} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{87} = 3\times 2 = 6$
	7	{9}	$P_{79} = \frac{(\tau_{79})^{1.1}\times(\eta_{79})^{-1}}{[(\tau_{79})^{1.1}\times(\eta_{79})^{-1}]} \times \frac{1}{Cos25^0} = 1.103$	9	$\tau_{79} = [(1-0.2)\times 0.8]^{1.1} + [3]^{-2(1)} = .732$ $\eta_{79} = 3\times 2 = 6$
	9	Join	**Final Sequence as obtained by Ant-2** is *Fork* → 5 → 6 → 8 → 7 → 9 → *Join*		

heuristic for the path between the current node and the selected one, has to be increased using the formula

$$\tau_{ij} = \left\{(1-\rho)\times\tau_{ij}^k\right\}^{\alpha} + \left\{Q''\times\left(\frac{1}{d_{ij}}\right)\right\}^{-2\beta} \text{ and } \eta_{ij} = Q''\times\eta_{ij}^k.$$

Table 5 demonstrates the step-wise and intermediate computations required to compute the next possible node from the current node, where the current node is considered to be any among those connected directly to the fork node. Here, sub-section for fork-join has been taken from Fig. 1.

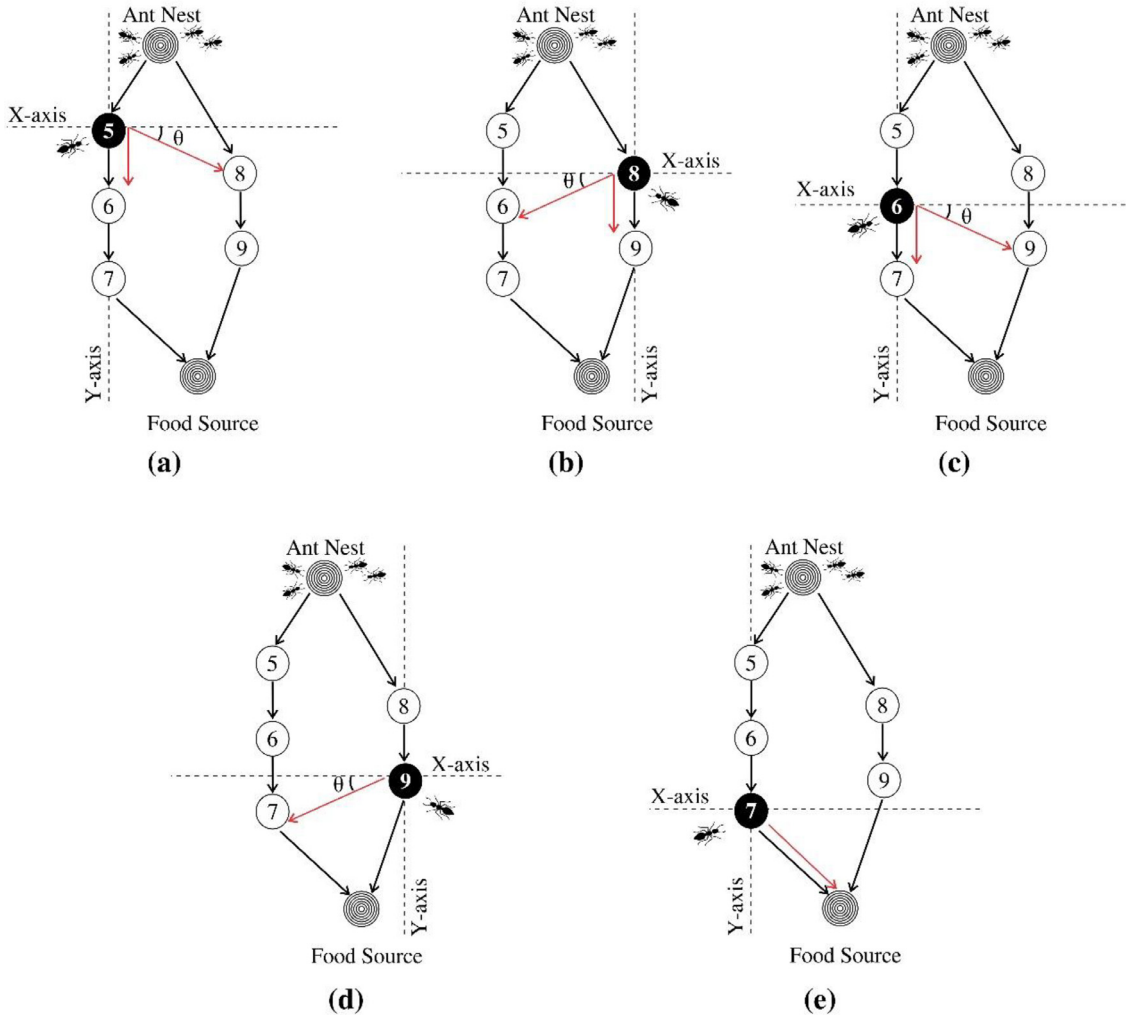

Fig. 9. (a)–(e): Execution of OBACO on the activity nodes under concurrent unit. (a) Node 5 has been chosen randomly from the fork/start node. Control may jump upon either 6 or 8; using orientation-based approach the next selected node will be 8; (b) At node 8, application of OBACO resulted in the next node as 6; (c) at node 6, OBACO, gives node 9 as next probable node to jump upon; (d) From node 9, ant will jump upon node 7; (e) Finally, the control jumps upon Join node as end node.

The tube-like structure, as highlighted in Fig. 10 explains the paths/sequences obtained in Table 5 through the use of two separate ant agents for performing traversal with the help of OBACO algorithm. Here, two ant agents resulted into two paths, first as *Fork* → 5 → 8 → 6 → 9 → 7 → *Join*; and second as *Fork* → 5 → 6 → 8 → 7 → 9 → *Join*.

The implementation snapshots depicting Eclipse code window for the development of the proposed Orientation based Ant Colony Algorithm for generating test scenarios from the XMI of UML AD have been placed in Annexure A. Annexure B depicts the snapshot of a sample XMI file generated using RSA for the sample student projects named Web Crawling and SDLC.

4.4. Concurrent Coverage (CC) criteria for the generated test scenarios

Let TS_s be a set of test scenarios for concurrent section of an activity diagram. TS_s will justify the concurrent coverage criteria, if the following condition is satisfied:

- For each concurrent node in an activity diagram, TS_s must include one scenario corresponding to every valid test scenario from fork node to join node.

Under fork-join node, the activity nodes can be interleaved as long as the ordering imposed by each sub-queue (SQ_i) is maintained. In other words, certain node combinations can be invalid because they do not recognize the ordering between nodes of a sub-queue. We need to ensure coverage of all valid interleaving sequences. Let V be the count of valid interleaving sequences, it can be computed as $V = \frac{(n_1+n_2+\ldots n_i)!}{(n_1!\times n_2!\times \ldots\ldots n_i!)}$. Where, the variable "*i*" is the count of sub-queues under fork-join; and "*n*" is the count of activity nodes in the *i*th sub-queue. To compute the Concurrent Coverage (CC) of the algorithm proposed here, formula given in Eq. (7) has been used:

$$Concurrent\ Coverage\ (CC) = \frac{TS_s}{V} \tag{7}$$

TS_s is the count of the valid test scenarios obtained after the application of a specific approach which is OBACO here; and V is the count of maximum valid scenarios that can be obtained. This computation assumes that the activity nodes within a fork-join block have single incoming and single outgoing edge only.

5. Experimentation and analysis of the results

5.1. Stopping criteria and parameter setting for the algorithms

To simulate the OBACO, ACO, and GA, two computationally unique environments, *viz.* C1 and C2 have been taken. Where, C1 is Windows

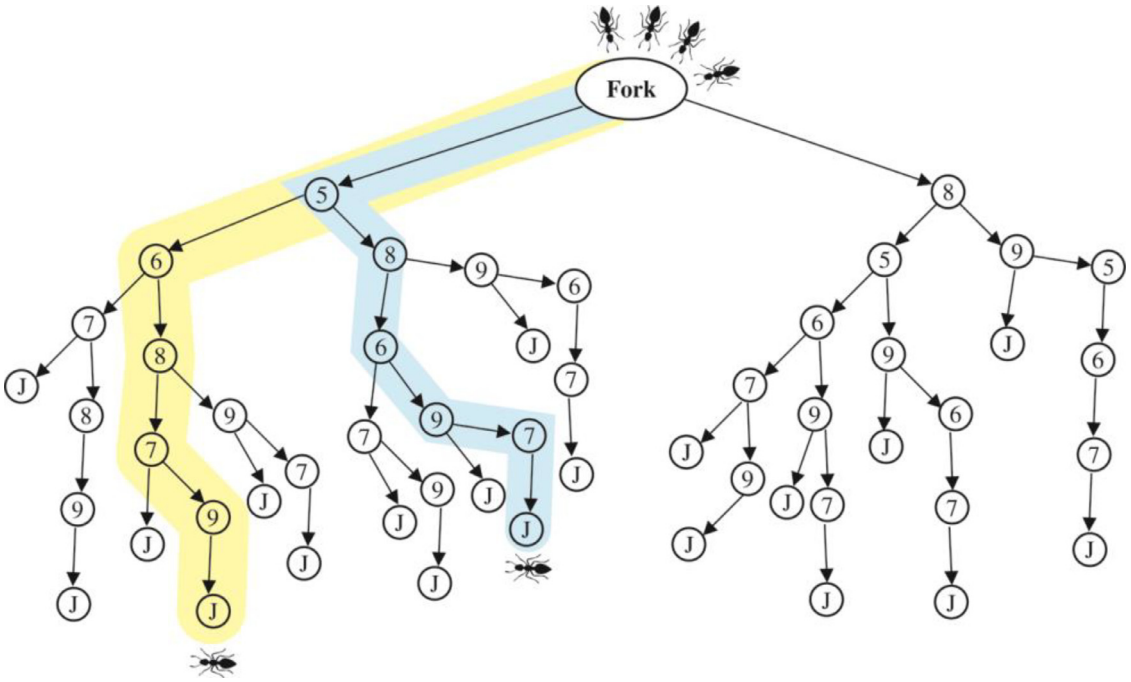

Fig. 10. Executing OBACO on a sample UML activity diagram.

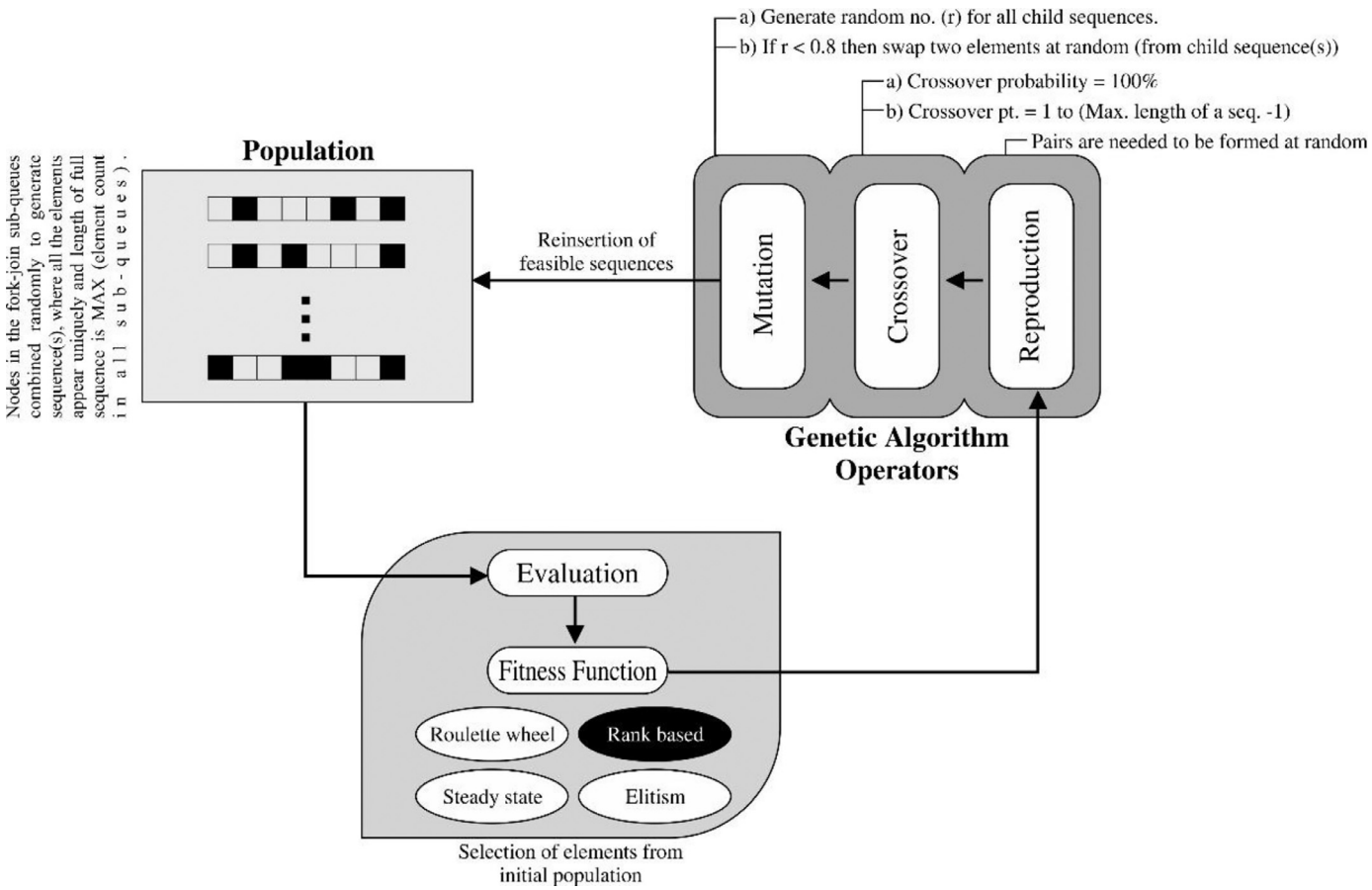

Fig. 11. EA approach used for generating test scenarios [25].

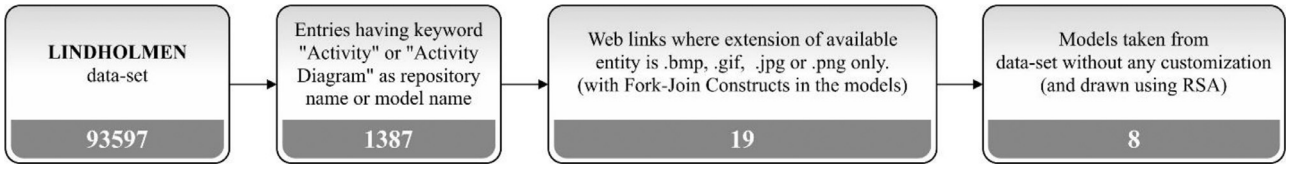

Fig. 12. Steps followed on LINDHOLMEN data-set to select UML activity diagrams [8,25].

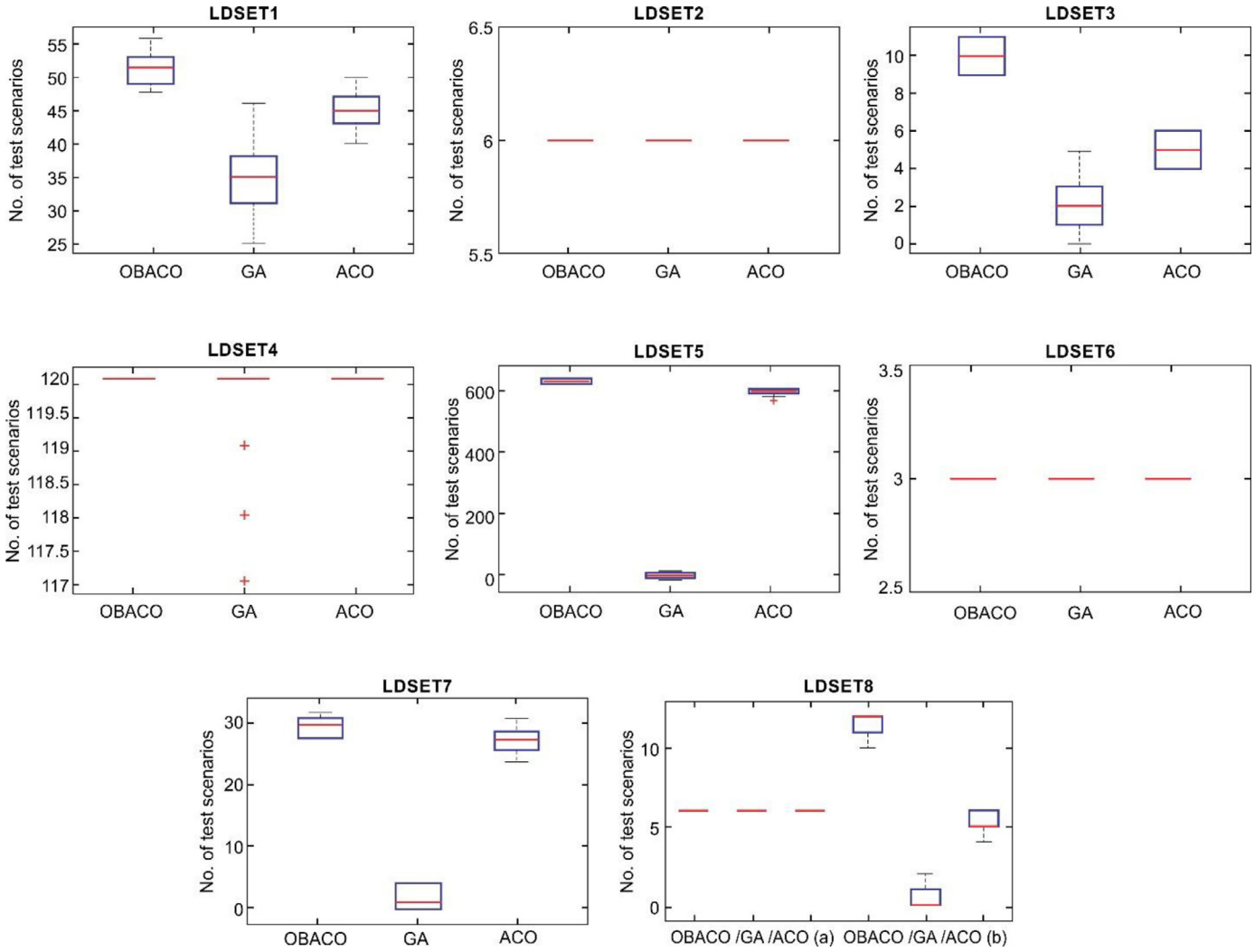

Fig. 13. Comparing OBACO, GA and ACO for generating feasible test scenarios.

10, a 64-bit operating system with Intel Core i3-6100 U, 2.30 GHz processor, and 4GB RAM. C2 is Windows 10, a 64-bit operating system with Intel Core i7-6700 CPU, 3.40 GHz, and 8GB RAM. Guidelines laid down by Črepinšek et al. [41] have been referred to set the control parameters for OBACO, ACO, and GA. For algorithms under consideration, the setting of different parameters has been done as follows:

- Parameters in Genetic algorithm

Genetic algorithm has its broad applicability in the areas such as bioinformatics, software optimization, multiple sequence alignment, gene theory, sequence generation and/or optimization [42,43]. Fig. 11 depicts EA [a form of GA] [19] which has been adopted here for the purpose of having a comparison with the proposed algorithm OBACO. Values used against various parameters in GA have been obtained by CRS-Tuning method as proposed in [40].

- Setting of Parameters in ACO

The technique provided by Srivastava et al. [27] to generate test paths through the use of ACO has been compared with the OBACO proposed here. The fundamentals provided by Dorigo et al. [44] for ant-based system have been used in the ACO-based methodology by Srivastava et al. The various features in ACO with their corresponding values taken for experimentation are as follows: τ_{ij} depicts the pheromone concentration at the edge between node i to j (1.0); ρ is the evaporation rate for pheromone chemical (0.2); η_{ij} heuristic value at edge from node i to j (2.0); α is the controlling factor of pheromone intensity (1.0); and β is the controlling element of heuristic value (1.0). The values taken for various parameters under consideration for OBACO are exhibited in Table 4. These have been selected through the CRS-Tuning approach [40].

- Stopping criteria

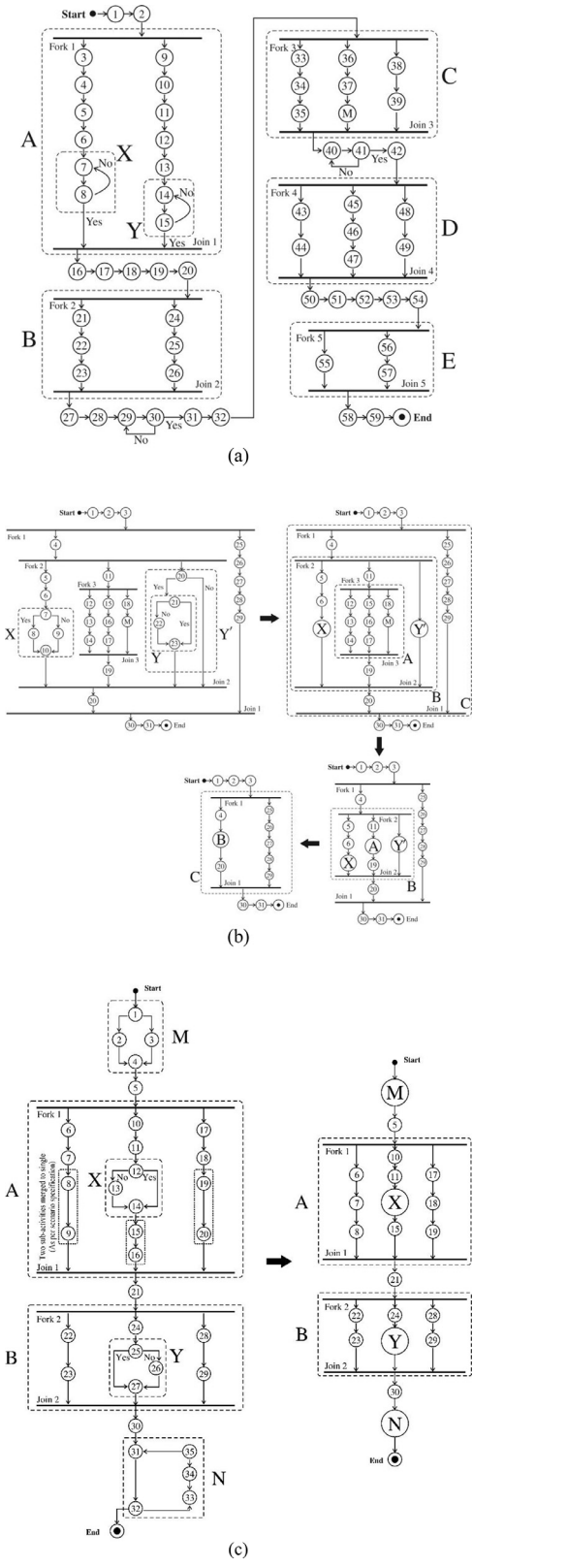

Fig. 14. (a) Transformational process for the activity diagram titled SDLC; (b) Web crawling; and (c) Torrents.

Research studies [45,46] revealed for taking equal units of fitness function evaluation to get a fair assessment among different bio-inspired approaches (*e.g.* orientation-based ant colony optimization, ant colony optimization, and genetic algorithm) used. Here, referring to the above-said studies, the stopping criteria for OBACO, ACO and GA, a maximum count for fitness evaluation function has been taken as 100,000.

- General parameter settings for algorithms under consideration

In an empirical analysis of the meta-heuristic, the computational features can be considered for having a comparison among the instances [5]. Computation effort is defined as the time taken by the machine to execute a defined set of statements. It is analyzed by two aspects, *viz.* (i) time taken in algorithm execution only (measured in millisecond), and (ii) time taken to transform an activity diagram in the corresponding XMI and executing algorithm afterwards (measured in millisecond). While experimenting, count of ants in ACO or population in GA has been taken equivalent to the number of ants in the proposed approach, OBACO. Similarly, the count of trials in ACO or the generation count in GA has been taken the same as the number of trials for OBACO. For each subject system, the OBACO, ACO, and GA have been executed thirty times. The computational effort has been compared for the said algorithms, considering only the algorithm execution time; and C1 as configuration with the population as 100 and trial count as 1000.

5.2. Goals for conducting experiments

Orientation-based ant colony optimization is deployed for the UML activity diagrams available in LINDHOLMEN data-set [8,9], undergraduate projects, and few artificial models. The main goals of this research are as below:

(i) To compare the number of viable test scenarios produced for OBACO with the existing algorithms, *viz.* ACO and GA.
(ii) To validate the final results obtained on data-set, student projects, and synthetic UML activity diagram after applying the proposed approach, using statistical analysis technique.

5.3. Benchmark model as subject system

The approach proposed here generates feasible test scenarios from UML activity diagrams at the design phase of Software Development Life Cycle (SDLC); hence, it is mandatory to have design level constructs to validate it. In project-based or product-based software houses, it is customary to prepare UML diagrams prior proceeding for development phase for a software (or a system). Most of the software companies do not provide these types of design elements related to the projects/products for the reason of maintaining privacy. Due to this sensitive concern, the realistic UML diagrams could not be used here to validate the proposed approach. Hence, an openly available repository named “LINDHOLMEN data-set” [8,9] that constitutes a list of UML projects has been used. Fig. 12 describes the steps taken to select final UML activity diagrams considered for validation of the proposed approach [8,9]. Features for the final eight activity diagrams filtered from LINDHOLMEN are listed in Table 6.

Eight activity diagrams have been finalized from the LINDHOLMEN repository for experimentation. Models labelled as LDSET1-2, LDSET4-5, and LDSET7 have only one fork-join (FJ) construct. Models marked LDSET3 and LDSET6 are having two fork-join nodes, but for the purpose of experimentation here, only the fork-join having a large count of activity nodes in it has been considered. LDSET8 contains three fork-join elements, in which one is independent fork-join (FJ1), and another is nested fork-join (FJ2, FJ3; with FJ3 as a sub-part of FJ2). Experimentations here have referred FJ1 of LDSET8 as LDSET8(a), and FJ2 as LDSET8(b).

Table 6
Selected activity diagrams for experiments; taken from the LINDHOLMEN data-set [8,25].

UML Model No.	fork-join(s) under AD	Count of activity nodes/entities in fork-join(s)	Control constructs in fork-join	element count in AD	Count of scenarios generated using [19]	Repository name in LINDHOLMEN data-set
LDSET1	1	18	Y	34	120	267,492/moje_rep/
LDSET2	1	10	Y	31	6	atadeesom/BubbleBeeDoc/
LDSET3	2	6 (FJ1), **20(FJ2)**	Y	65	20	BGCX261/zimeoprocess-svn-to-git/
LDSET4	1	15	N	36	120	DmitriyG/Diplom/
LDSET5	1	24	Y	38	1680	HandreWatkins/COS-301-Phase2/
LDSET6	2	**9(FJ1)**, 6(FJ2)	Y	33	7	ooad-2014-2015/JotaPlanet/
LDSET7	1	16	N	22	35	85pando/Zusammenfassungen/
LDSET8	3(1, 2*)	**11(FJ1), 34**(FJ2,contain nested FJ3 with 6 elements)	Y	56	9, 13	bsaunder/cs414-discussions/

Table 7
Features of UML activity diagrams derived from Student Project (STDP) assignment.

Model Label	Nested Fork-Join	Count of undergraduates involved	Number of all constructs in AD	XMI File (KB)	Count of fork-join(s) in AD	Count of control entities in AD	AD Model [7]
STDP1	No	3	154	45	5	4	SDLC
STDP2	Yes	5	87	26	3	3	WebCrawling
STDP3	No	3	83	26	2	4	Torrent

Table 8
Number of test scenarios (or paths).

Model No.	Name of UML AD Model	Annotation for the sub-part in Fig. 14	Ideal number of sequences generated with [19]
STDP1	SDLC	A	462
		B	20
		C	560
		D	210
		E	3
STDP2	WebCrawling	A	560
		B	140
		C	56
STDP3	Torrent	A	4200
		B	90

5.4. OBACO Vs GA and ACO

Orientation-based ACO has been compared using the UML activity diagrams taken from LINDHOLMEN repository with existing ACO and GA techniques for generating test scenarios.

Fig. 13 depicts box-plots for the cases selected from LINDHOLMEN data-set for validation of the proposed approach. It can be observed that OBACO is giving an added count for feasible test scenarios as compared to ACO and GA for the models LDSET1, LDSET3, LDSET5, and LDSET8b. For the models, LDSET2, LDSET4, LDSET6 and LDSET8a, the proposed approach has provided comparable results to those of ACO and GA. For the model LDSET7, the proposed technique has provided similar results as produced by ACO, but better than those given by GA. The CC criteria values obtained through the OBACO approach in LDSET1, LDSET2, LDSET3, LDSET4, LDSET5, LDSET6, LDSET7, LDSET8a and LDSET8b are 0.43, 1.0, 0.55, 1.0, 0.37, 0.43, 0.83, 0.66, and 0.84 respectively.

5.5. Student assignment models as subject system

UML activity diagrams, other than LINDHOLMEN dataset, have been taken from the project assignment of eight Bachelor of Engineering (B.E.) students in the software engineering stream. The work was carried out under the guidance of the author of this research manuscript in the year 2015. The diagrams submitted by the students have been fine-tuned further [7]. The students put their efforts by creating two parallel clusters of three and five students correspondingly. They spent about 45 person-hours to create UML activity diagram and its corresponding scenario. The same approach was followed by the two student groups for drafting the models. The teams used IBM Rational Software Architect (RSA) and UML 2.2, as employed in the software industries for software/system projects.

The relevant features of UML activity diagrams as drafted by students have been described in Table 7. Further, the elements under fork-join construct here are tentatively twice in comparison to the cases derived from LINDHOLMEN. Thus, these models also contributed significantly in the testing of OBACO regarding its scalability.

Activity diagrams having nesting in fork-join, looping, and if-else, *etc.* are converted into an Intermediate Testable Model (ITM) with no nesting and loop constructs. This serves as an input AD for the algorithm used for test scenario generation [3]. To convert an AD into its simplified version, every nested concurrent or control element is substituted by an annotated node. Replacement of constructs starts from the innermost element (fork-join or decision element); it further moves towards outer construct till a single fork-join is left.

OBACO has been applied on each simplified and independent AD with fork-join (with no nested or control constructs) to generate test scenarios. The steps taken to convert complex AD [7] into its corresponding simplified version are highlighted in Fig. 14. Therefore, the final AD resulting from the conversion process remains 'the derived final non-nested simplified version' produced to work in the experimentation.

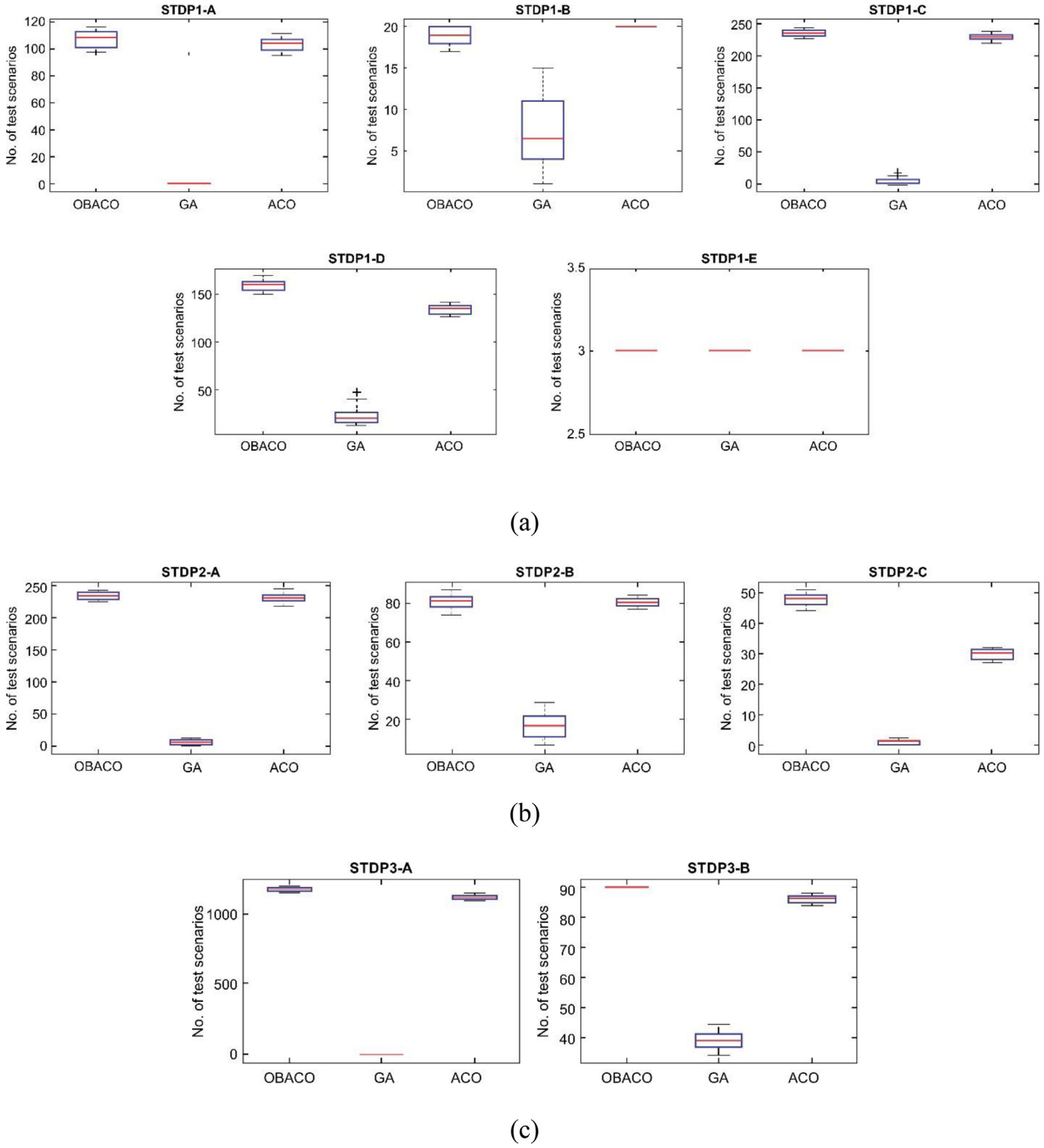

Fig. 15. Box-plot depicting comparisons among OBACO, ACO and GA *w.r.t.* number of test scenarios generated for STDP1, STDP2, and STDP3.

Number of test scenarios generated by using the formula described in the work [19] for each sub-section of the graph *w.r.t.* STDP1, STDP2, and STDP3 [Fig. 14(a), (b), and (c)] are shown in Table 8.

The graphs appearing in Fig. 15 show that the proposed approach delivers either more or the same count of feasible test scenarios as compared to those in ACO and GA, applied for the sub-activity diagrams, *viz.* A, B, C, D, E for student project 1 (STDP1), and A, B, C for student project 2 (STDP2), and A, B for student project 3 (STDP3) (as shown in Fig. 14).

The application of the proposed approach on a simplified version of an AD resulted into test scenarios with many annotated activity nodes (each annotation depicting a nested fork-join element or a control entity). Recursive application of OBACO on annotated nodes (nested fork-join) creates more test scenarios in the existing set of test se-

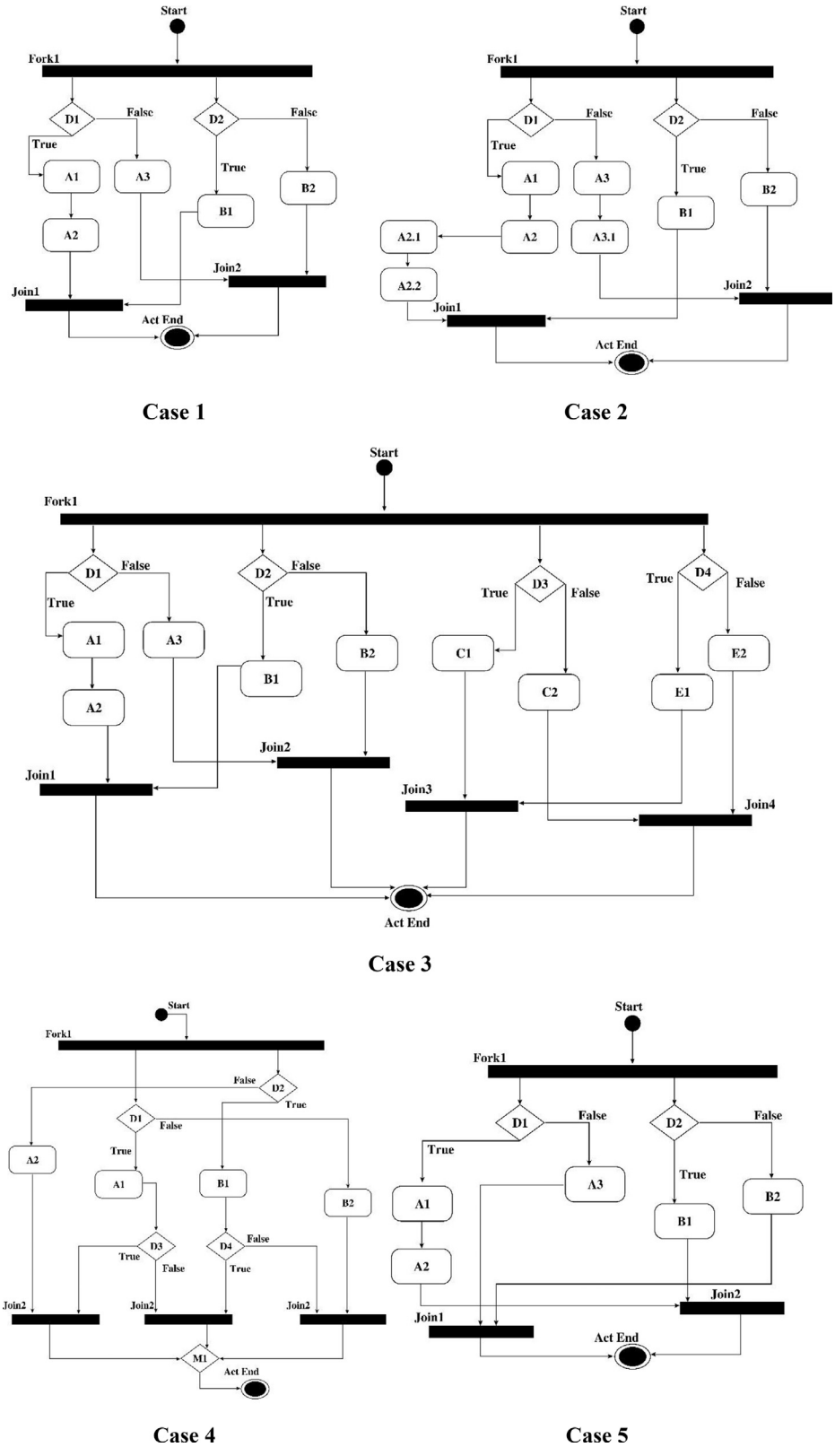

Fig. 16. UML AD with one fork and a number of joins [25].

quences. Annotated nodes get replaced by their corresponding simple paths from fork to join node [3]. This substitution in place of an annotated node results into a significant count generation in the feasible scenarios. Inherent randomness present at the crossover as well as mutation stage in GA resulted into poor performance of the said algorithm due to unusual combinations of the nodes in the final test sequences.

The CC criteria values obtained after the application of OBACO in STDP1-A, STDP1-B, STDP1-C, STDP1-D, and STDP1-E are 0.23, 0.94, 0.42, 0.76, and 1.0 respectively. In STDP2-A, STDP2-B, and STDP2-C,

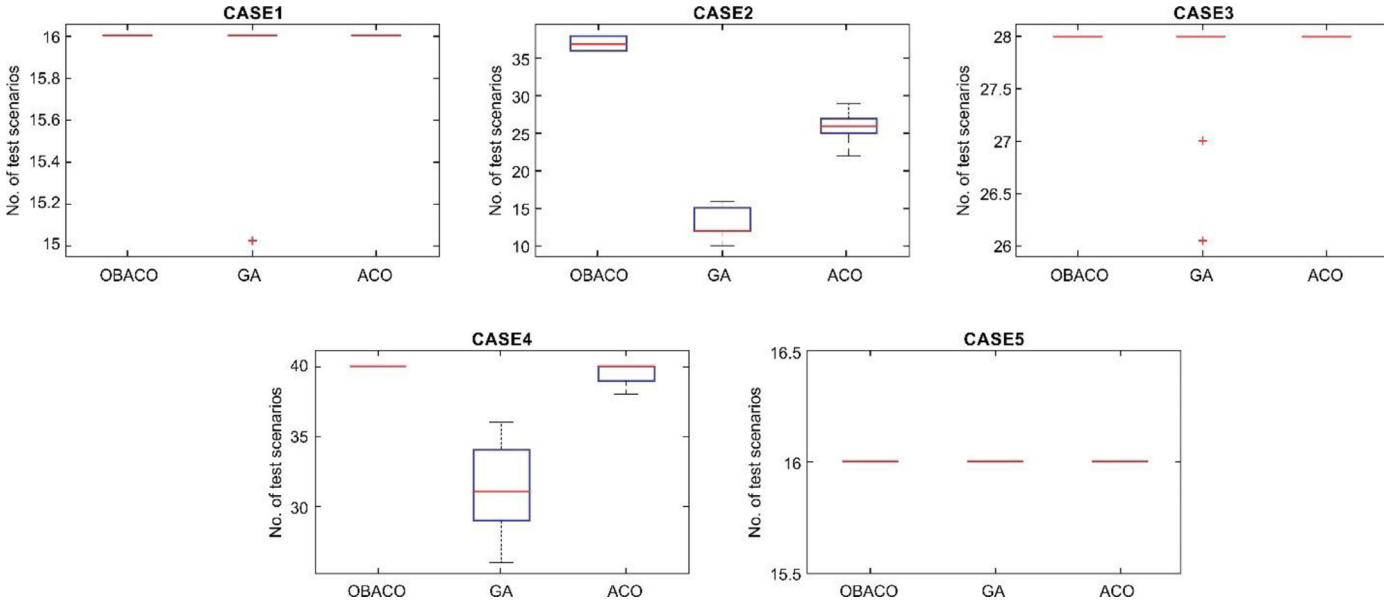

Fig. 17. Count of feasible test scenarios generated through OBACO, GA and ACO on synthetic models.

Table 9
t-test results taking number of feasible test scenarios for OBACO, GA, and ACO.

Model No.		OBACO Vs GA				OBACO Vs ACO				OBACO Vs (GA/ACO)	
		h	p	SD GA	Mean GA	h	p	SD ACO	Mean ACO	SD OBACO	Mean OBACO
LDSET1		1	5.51E−22	5.59	34.8	1	2.38E−14	2.59	45.33	2.12	51.5
LDSET2		NaN	NaN	0	6	NaN	NaN	0	6	0	6
LDSET3		1	1.09E−34	1.43	1.93	1	1.56E−31	0.81	5.13	0.80	10.1
LDSET4		1	0.0110414	0.76	119.63	NaN	NaN	0	120	0	120
LDSET5		1	1.58E−116	3.37	5.36	1	9.33E−39	5.84	590.66	3.28	630.16
LDSET6		NaN	NaN	0	3	NaN	NaN	0	3	0	3
LDSET7		1	7.75E−57	1.60	1.63	1	1.58E−06	2.02	26.36	1.51	28.83
LDSET8	a	NaN	NaN	0	6	NaN	NaN	0	6	0	6
	b	1	1.72E−51	0.77	0.56	1	2.60E−37	0.80	5.2	0.75	11.33

the OBACO has provided the values as 0.42, 0.57, and 0.84 respectively. Further, in STDP3-A and STDP3-B, the CC criteria values obtained are 0.28 and 1.0 respectively. Less values of CC in LDSET5, STDP1-A and STDP3-A is due to a large number of interleaving among activity nodes under fork-join. Further, decrease in the value can also be attributed to the redundant traversals of the paths by the ants *i.e.* they tend to traverse an already visited path (test scenario).

5.6. UML activity diagram having more than one join elements

Fig. 16 exhibits the artificially created models for the UML activity diagrams with one fork construct and a number of joins.

Box-plot diagram, as shown in Fig. 17, depicts that OBACO used to compute test scenarios for synthetic cases resulted better or equivalent as compared to GA and ACO.

The CC criteria values obtained after the application of OBACO in synthetic Case1, Case2, Case3, Case4 and Case5 are 1.0, 0.8, 1,0, 1.0 and 1.0 respectively.

5.7. Statistical Analysis

The *t*-test [47,48] is a frequently used method to examine the statistical significance of algorithmic outcomes. This research work makes a statistical assessment of the results obtained for the algorithms, *viz.* OBACO, ACO and GA. An assumption of equal variances for the populations under consideration has been taken at the significance level of $\alpha = 0.05$.

The *t*-test results are listed in Table 9. The parameters taken into consideration include p, h, Mean, and SD. Where, p is the probability of detecting statistical test values as extreme than the experiential value in the Null hypothesis; h is hypothesis test result; mean is the average of count of test scenarios generated; and SD is the standard deviation of the results achieved for all executions. Further, NaN stands for Not a Number. Entry value as NaN for the parameters h and p indicates similarity in the results obtained from the algorithms under consideration. Due to similarity, the value of SD is zero for these cases.

The *t*-test has been conducted to negate the Null hypothesis, *i.e.* taking population means as same. The value as numeric one for the parameter h indicates a denial of the Null hypothesis. The results show that two mean values are different when p-values, as obtained, are less than 0.05, taken as the level of significance α. The results obtained are statistically significant as the standard deviation for OBACO is less than ACO and GA, whereas the mean value for count of test scenarios generated is higher.

6. Threats to validity

This sub-section highlights the potential threats related to the validity of proposed OBACO algorithm and subject systems. The potential risks taken up for consideration are as follows:

- **Internal validity** is associated with the performance analysis of meta-heuristic search techniques having a biased selection of datasets that can favor a particular method. The experiments have been conducted using the student projects and datasets available

publicly at Github repository. A more significant number of datasets seem to be necessary as real-world datasets are kept confidential by the companies that own them. At this point, the properties associated with changes in the locality of the model (*i.e.* automatically detecting missing nodes, cross-synchronization, *etc.*) are not well thought-out.

- **External validity** associated with the generality of perceived outcomes and is another significant threat to validity. Higher the complexity of a real-world UML activity diagram, more would be its generalization. But it is difficult to obtain the UML diagrams related to real-world software projects due to the privacy policies followed by the software development companies. As a result, it is quite challenging to assess the extent to which the proposed OBACO algorithm would be helpful to the software developers, testers and analysts. However, it is believed that the proposed technique would be quite helpful in generating the scenarios for development as well as testing module in SDLC.
- **Construct validity** is another crucial threat associated with the process of measurement. In the proposed approach, pheromone-heuristic (PH) value calculations have been taken as a threat to construct validity as the test scenario generation involves computation for pheromone-heuristic (PH) value. PH value ascertains which node would appear next in the test scenario. Here, certain fixed initial values have been taken for computing PH as other estimations may turn to be more significant to improve the efficacy of pheromone-heuristic centred approaches. Further, although the allied factor related to orientation has been taken after the analysis of parameter sensitivity, and the corresponding value has been taken as a whole number, but some floating point values may help to improve the efficacy of the proposed OBACO.

7. Conclusion

The experimental evaluation of OBACO, GA, and ACO on UML activity diagram has revealed that the primary concern of sequence explosion arises, when the activity nodes of the sub-queue(s) under concurrent construct undergo interleaved execution with each other. The analysis recognizes superior solutions for the analogous computational power. At present, the studies of ant colony algorithm for generating test sequences are pre-eminent to researchers; and further investigation of heuristic techniques is vital to determine their cost-effectiveness. The use of orientation-based ant colony algorithm can be of great significance. The OBACO approach is a good substitute of basic GA and ACO used for finding the feasible paths when the count of total paths is substantially large.

The proposed orientation-based ant colony algorithm may be further investigated by implementing it in different directions. Firstly, OBACO can be explored for making clusters of test sequences as generated from UML activity diagram to devise a cost-effective approach for regression testing which executes a specific cluster after having a change in the requirement specifications of the software. Secondly, in conjunction with software re-modularization, OBACO can be applied to group the test sequences in the context of coupling and cohesion for better structural testing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that couId have appeared to influence the work reported in this paper.

Annexure A

Figs. 18 and 19.

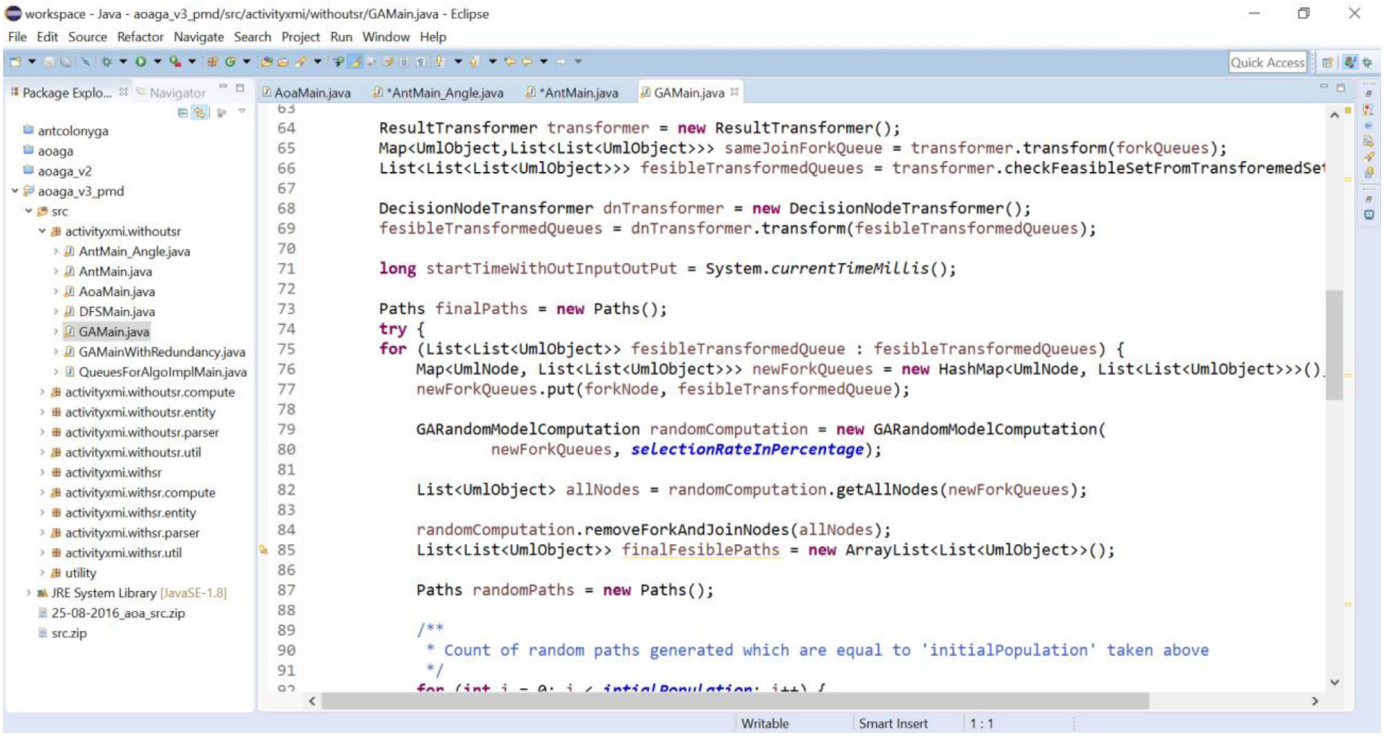

Fig. 18. Implementation Snapshot Depicting Eclipse code window for the deployment and development of Genetic Algorithm for generating test scenarios from the XMI of UML AD.

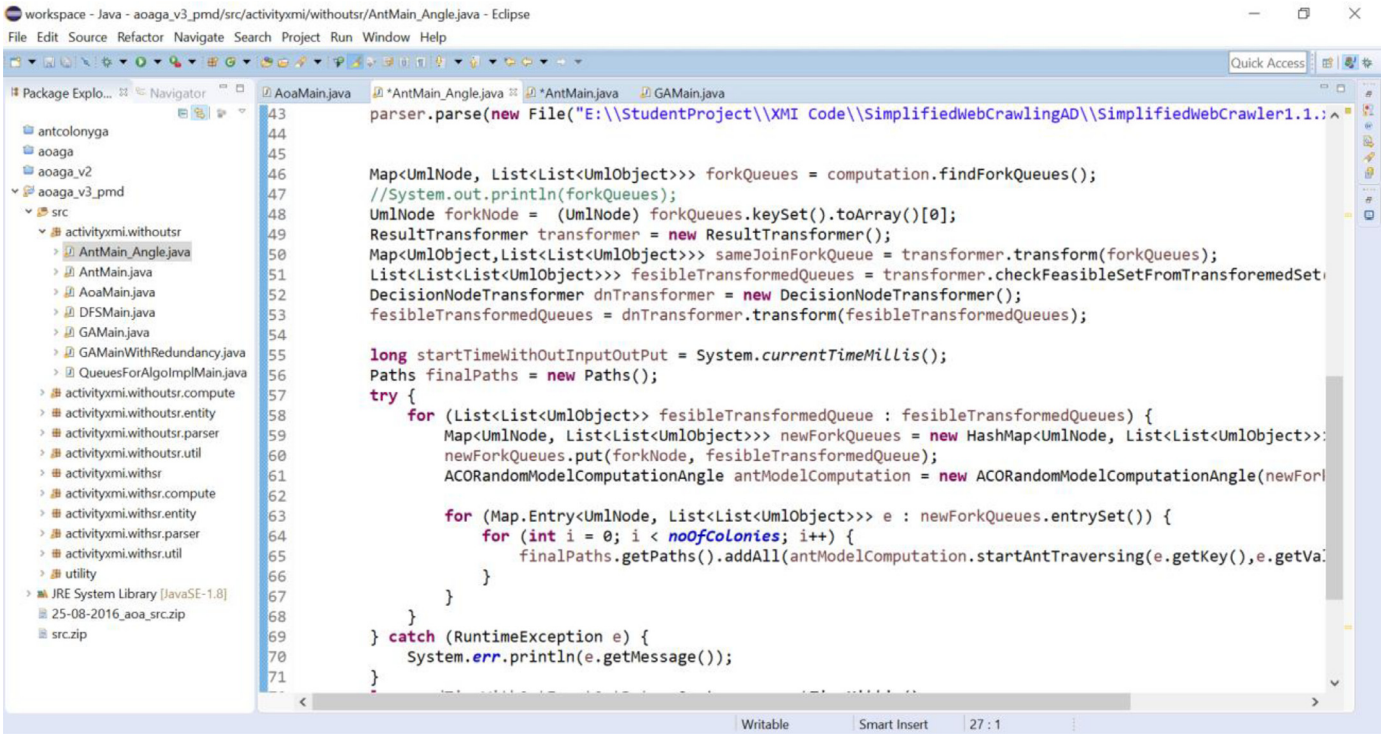

Fig. 19. Implementation Snapshot Depicting Eclipse code window for the deployment and development of Orientation based Ant Colony Algorithm for generating test scenarios from the XMI of UML AD.

Annexure B

Figs. 20 and 21

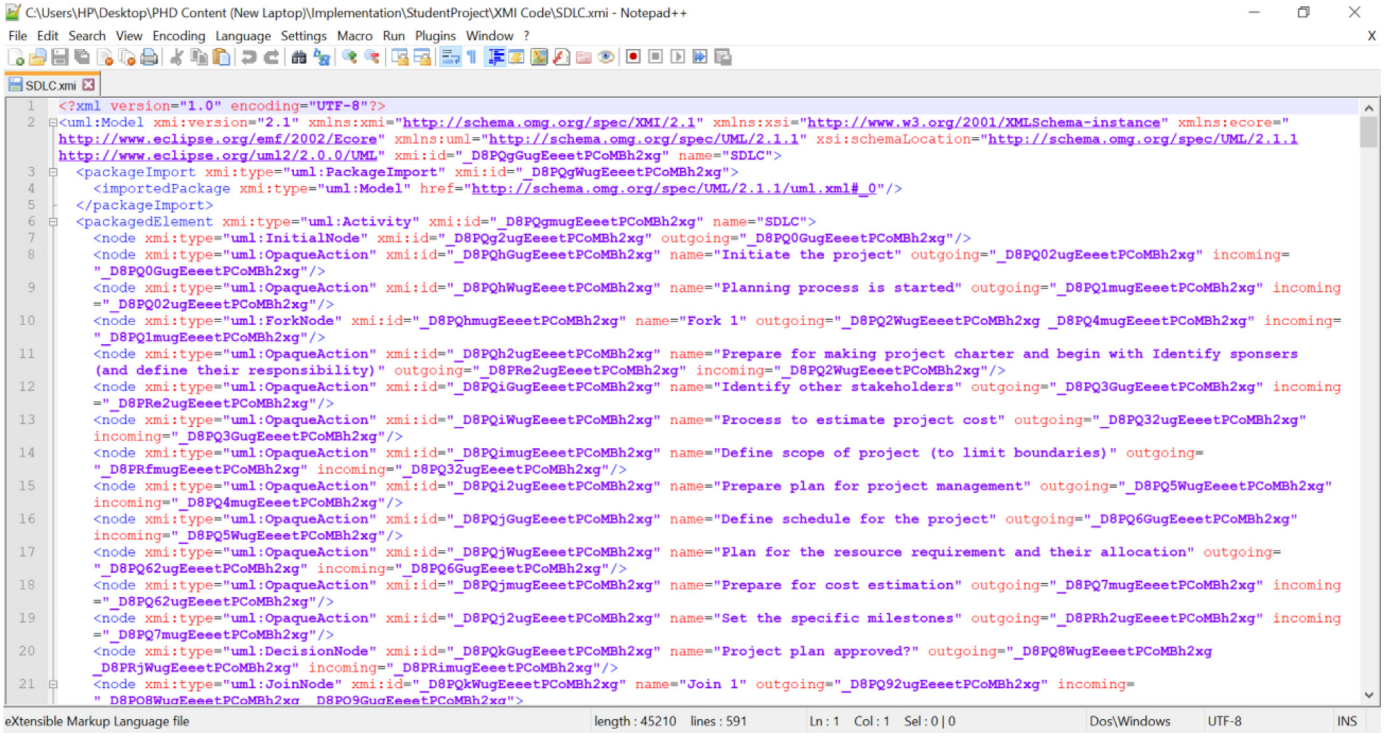

Fig. 20. Snapshot Depicting XMI of UML AD generated using IBM RSA for the student project named SDLC.

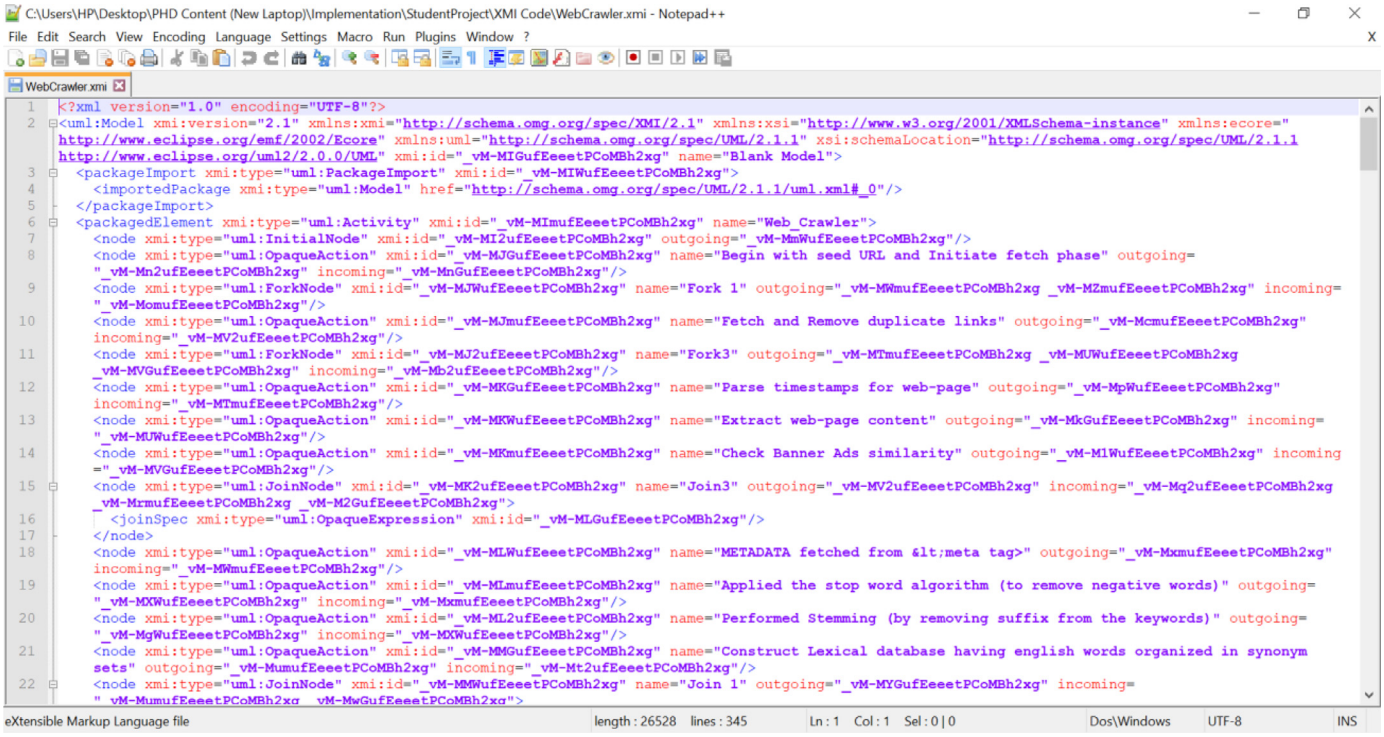

Fig. 21. Snapshot Depicting XMI of UML AD generated using IBM RSA for the student project named Web Crawling.

CRediT authorship contribution statement

Vinay Arora: Conceptualization, Methodology, Formal analysis, Validation, Software, Writing - review & editing. **Maninder Singh:** Investigation, Methodology, Software, Validation. **Rajesh Bhatia:** Conceptualization, Investigation, Validation, Writing - review & editing.

References

[1] D. Kundu, M. Sarma, D. Samanta, A UML model-based approach to detect infeasible paths, J. Syst. Softw. 107 (2015) 71–92.
[2] V. Panthi, D.P. Mohapatra, Firefly optimization technique based test scenario generation and prioritization, J. Appl. Res. Technol. 16 (2018) 466–483.
[3] A. Nayak, D. Samanta, Synthesis of test scenarios using UML activity diagrams, Softw. Syst. Model. 10 (2011) 63–89.
[4] V. Arora, R. Bhatia, M. Singh, A systematic review of approaches for testing concurrent programs, Concurr. Comput. 28 (2016) 1572–1611.
[5] E.G. Talbi, Metaheuristics from Design to Implementation, John Wiley & Sons, 2009.
[6] X.S. Yang, Nature-inspired Optimization Algorithms, Elsevier, 2014.
[7] UML activity diagram repository, (https://github.com/UMLADRepo/AD-Repository)
[8] The lindholmen dataset, (http://oss.models-db.com/Downloads/lindholmendb_v2/UMLFiles_list_V2.0.csv)
[9] R. Hebig, T.H. Quang, M.R.V. Chaudron, G. Robles, M.A. Fernandez, The quest for open source projects that use UML: mining github, in: Proceedings of ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, 2016, pp. 173–183.
[10] M. Shirole, R. Kumar, UML behavioral model based test case generation: a survey, ACM SIGSOFT Softw. Eng. Notes 38 (2013) 1–13.
[11] A. Jyoti, V. Arora, Visualization of Deadlock and Wait-Notify Anomaly in Multithreaded Programs Master Thesis, Computer Science & Engineering, Thapar University, 2014.
[12] V. Verma, V. Arora, Code and Model Based Test Sequence Generation for Multithreaded Programs Master Thesis, Computer Science & Engineering, Thapar University, 2014.
[13] A. Jyoti, V. Arora, Debugging and visualization techniques for multithreaded programs: a survey, in: Proceedings of IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE), Jaipur, India, 2014, pp. 1–6.
[14] S.A. Asadollah, D. Sundmark, S. Eldh, H. Hansson, W. Afzal, 10 Years of research on debugging concurrent and multicore software: a systematic mapping study, Softw. Qual. J. (2016) 1–34.
[15] R.S. Pressman, Software Engineering, A Practitioner's Approach, 7th ed., McGraw-Hill, New York, NY, 2010.
[16] H. Li, C.P. Lam, in: Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams, Testing of Communicating Systems, Lecture Notes in Computer Science, 3502, Springer, Berlin Heidelberg, 2005, pp. 69–80.
[17] C.P. Lam, Computational intelligence for functional testing, in: F. Meziane, S. Vadera (Eds.), Artificial Intelligence Applications For Improved Software Engineering Development: New Prospects, IGI Global, 2010.
[18] U. Farooq, C.P. Lam, H. Li, Towards automated test sequence generation, in: Proceedings of 19th Australian Conference on Software Engineering, Perth, WA, Australia, 2008, pp. 441–450.
[19] M. Shirole, M. Kommuri, R. Kumar, Transition sequence exploration of uml activity diagram using evolutionary algorithm, in: Proceedings of 5th India Software Engineering Conference, Kanpur, India, ACM, 2012, pp. 97–100.
[20] A. Mishra, D.P. Mohapatra, Generation and Prioritization of test sequences using UML activity diagram, National Institute of Technology, 2014.
[21] F. Sayyari, S. Emadi, Automated generation of software testing path based on ant colony, in: 2nd International Congress on Technology, Communication and Knowledge (ICTCK), Mashhad, Iran, Islamic Azad University, 2015, pp. 435–440.
[22] V. Panthi, D.P. Mohapatra, ACO based embedded system testing using UML activity diagram, in: 2016 IEEE Region 10 Conference (TENCON), Singapore, IEEE, 2016, pp. 237–242.
[23] A.A. Kyaw, M.M. Min, Test path optimization algorithm compared with ga based approach, in: T. Zin, J.W. Lin, J.S. Pan, P. Tin, M. Yokota (Eds.), Genetic and Evolutionary Computing, Springer, 2016, pp. 455–463.
[24] G. Bhattacharjee, S. Dash, Test path prioritization from uml activity diagram using a hybridized approach, Int. J. Knowl.-Based Organ. (IJKBO) 8 (2018) 83–96.
[25] V. Arora, R. Bhatia, M. Singh, Synthesizing test scenarios in UML activity diagram using a bio-inspired approach, computer languages, Syst. Struct. 50 (2017) 1–19.
[26] D. Kundu, D. Samanta, A novel approach to generate test cases from UML activity diagrams, J. Object Technol. 8 (2009) 65–83.
[27] P.R. Srivastava, K. Baby, G. Raghurama, An approach of optimal path generation using ant colony optimization, in: 2009 IEEE Region 10 Conference, Singapore, 2009, pp. 1–6.
[28] J.L. Deneubourg, J.M. Pasteels, J.C. Verhaeghe, Probabilistic behaviour in ants: a strategy of errors? J. Theor. Biol. 105 (1983) 259–271.
[29] M. Dorigo, V. Maniezzo, A. Colorni, The Ant system: An autocatalytic Optimizing Process, Dipartimento di Elettronica, Politecnico di Milano, 1991 Technical Report 91-016 revised.
[30] S. Christodoulou, Scheduling resource-constrained projects with ant colony optimization artificial agents, J. Comput. Civil Eng. 24 (2009) 45–55.
[31] M.D. Toksari, A hybrid algorithm of ant colony optimization (ACO) and iterated local search (ILS) for estimating electricity domestic consumption: case of Turkey, Int. J. Electr. Power. Energy Syst. 78 (2016) 776–782.
[32] A. Prakasam, N. Savarimuthu, Metaheuristic algorithms and probabilistic behaviour: a comprehensive analysis of Ant colony optimization and its variants, Artif. Intell. Rev. 45 (2016) 97.
[33] T. Stützle, M. López-Ibánez, P. Pellegrini, M. Maur, M. Antonio Montes de Oca, M. Birattari, M. Dorigo, Parameter adaptation in ant colony optimization, in: Y. Hamadi,

E. Monfroy, F. Saubion (Eds.), Autonomous Search, Springer, Berlin Heidelberg, 2012, pp. 191–215.
[34] H. Wang, Z. Shi, S. Li, Multicast routing for delay variation bound using a modified ant colony algorithm, J. Netw. Comput. Appl. 32 (2009) 258–272.
[35] S. Garnier, A. Guérécheau, M. Combe, V. Fourcassié, G. Theraulaz, Path selection and foraging efficiency in Argentine ant transport networks, Behav. Ecol. Sociobiol. (Print) 63 (2009) 1167–1179.
[36] C. Zhang, Z. Zhen, D. Wang, M. Li, UAV path planning method based on ant colony optimization, in: IEEE Chinese Control and Decision Conference (CCDC), Xuzhou, China, 2010, pp. 3790–3792.
[37] E. Sun, C. Wang, F. Tian, A survey on multi-path routing protocols in wireless multimedia sensor networks, Indonesian J. Electr. Eng. Comput. Sci. 12 (2014) 6978–6983.
[38] K. Sundareswaran, K. Jayant, T.N. Shanavas, Inverter harmonic elimination through a colony of continuously exploring ants, IEEE Trans. Ind. Electron. 54 (2007) 2558–2565.
[39] J. Branke, M. Guntsch, Solving the probabilistic TSP with ant colony optimization, J. Math. Modell. Algorithms 3 (2004) 403–425.
[40] N. Veček, M. Mernik, B. Filipič, M. Črepinšek, Parameter tuning with chess rating system (CRS-Tuning) for meta-heuristic algorithms, Inf. Sci. (Ny) 372 (2016) 446–469.
[41] M. Črepinšek, S.H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (2014) 161–170.
[42] A. Arcuri, G. Fraser, On parameter tuning in search based software engineering, in: Lecture Notes in Computer Science, 6956, Springer, Berlin Heidelberg, 2011, pp. 33–47.
[43] R. Karimpour, G. Ruhe, Evolutionary robust optimization for software product line scoping: an explorative study, computer languages, Syst. Struct. 47 (2017) 189–210.
[44] M. Dorigo, T. Stützle, Ant Colony Optimization, The MIT Press, Cambridge, Massachusetts, 2004.
[45] M. Mernik, S.H. Liu, D. Karaboga, M. Črepinšek, On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation, Inf. Sci. (Ny) 291 (2015) 115–127.
[46] J.K.C. Amarjeet, Harmony search based remodularization for object-oriented software systems, computer languages, Syst. Struct. 47 (2017) 153–169.
[47] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (2011) 3–18.
[48] H. Garg, A hybrid PSO-GA algorithm for constrained optimization problems, Appl. Math. Comput. 274 (2016) 292–305.