Wang CH, Jin Z, Zhang W *et al.* Activity diagram synthesis using labelled graphs and the genetic algorithm. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(6): 1388–1406 Nov. 2021. DOI 10.1007/s11390-020-0293-9

Activity Diagram Synthesis Using Labelled Graphs and the Genetic Algorithm

Chun-Hui Wang[1,2,3], *Member, CCF*, Zhi Jin[1,2,*], *Fellow, CCF, Senior Member, IEEE*, Wei Zhang[1,2]
Didar Zowghi[4], Hai-Yan Zhao[1,2], *Senior Member, CCF, Member, ACM, IEEE*, and Wen-Pin Jiao[1,2]

[1] *Key Laboratory of High Confidence Software Technology (Ministry of Education), Peking University Beijing 100871, China*

[2] *Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China*

[3] *School of Computer Science, Inner Mongolia Normal University, Hohhot 010022, China*

[4] *Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney 2007, Australia*

E-mail: {ciecwch, zhijin, zhangw.sei}@pku.edu.cn; Didar.Zowghi@uts.edu.au; {zhhy, jwp}@sei.pku.edu.cn

Received January 17, 2020; accepted June 9, 2020.

Abstract Many applications need to meet diverse requirements of a large-scale distributed user group. That challenges the current requirements engineering techniques. Crowd-based requirements engineering was proposed as an umbrella term for dealing with the requirements development in the context of the large-scale user group. However, there are still many issues. Among others, a key issue is how to merge these requirements to produce the synthesized requirements description when a set of requirements descriptions from different participants are received. Appropriate techniques are needed for supporting the requirements synthesis. Diagrams are widely used in industry to represent requirements. This paper chooses the activity diagrams and proposes a novel approach for the activity diagram synthesis which adopts the genetic algorithm to repeatedly modify a population of individual solutions toward an optimal solution. As a result, it can automatically generate a resulting diagram which combines the commonalities as many as possible while leveraging the variabilities of a set of input diagrams. The approach is featured by: 1) the labelled graph proposed as the representation of the candidate solutions during the iterative evolution; 2) the generalized entropy proposed and defined as the measurement of the solutions; 3) the genetic algorithm designed for sorting out the high-quality solution. Four cases of different scales are used to evaluate the effectiveness of the approach. The experimental results show that not only the approach gets high precision and recall but also the resulting diagram satisfies the properties of minimization and information preservation and can support the requirements traceability.

Keywords crowd-based requirements engineering, requirements synthesis, activity diagram, genetic algorithm

1 Introduction

Many applications need to satisfy the requirements that are of considerable diversity [1, 2]. For example, an online payment system serves for various users who may have different expectations on the payment ways and the transaction procedures. For collecting diverse requirements, it is needed to have considerable amount of distributed participants. However, it has been recognized that the traditional face-to-face requirements engineering approach has limitations in involving a large number of distributed participants [3]. Currently, there are some efforts paying attention to involving many participants. That leads to the setting of the crowd-based approach [1] which uses the crowd sourcing model to support requirements elicitation from a broader participant group [2, 4, 5]. The crowd-based requirements engineering (Crowd RE) [3, 6] has the expectation that a certain number of requirements providers can serve as rich requirements sources [3, 7, 8].

Regular Paper

A preliminary version of the paper was published in the Proceedings of APRES 2017.

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61620106007, 61751210 and 61690200.

*Corresponding Author

In Crowd RE, one of the key tasks is merging the diverse requirements to develop the synthesized requirements (we call it the requirements synthesis). At present, the requirements synthesis mainly relies on manual efforts with occasional support of text mining techniques [9]. There is still no effective technique to synthesize the fragmented requirements in an automatic way. The requirements synthesis remains to be a burden for requirements engineers and it is time-consuming and error-prone. For example, a team of three analysts took 130 man-hours to merge 25% of four pairs of variants (the number of nodes in them is 696, 100, 679, 389, respectively) of an end-to-end process model [10].

The requirements synthesis relies on the requirements representation to a large extent. At present, user stories [7, 11], scenario descriptions [12], and business workflows [13], etc. are widely adopted in Crowd RE. As the activity diagram is a general representation of them [14], it is suitable for requirements synthesis for Crowd-based RE.

There are some efforts on model merging which are related. Many of them are focusing on dealing with two models. The compare-match-compose process, i.e., to compare any pair of nodes from the two models, find the node-to-node correspondences and then compose the two models based on the correspondences, is often adopted [10, 15]. To deal with multiple models, this process needs to be extended to a step-by-step iterative binary model composition. The resulting model generated by the iterative strategy may rely on the order of the binary composition [16, 17].

There are existing studies dealing with multiple models. For example in [16], the models are representing different opinions and the node-to-node correspondences among the models are identified to build a global view which can exhibit the inconsistencies among the models. That is not for merging the models.

[17] designs a structure, i.e., *tuple*, as a multi-arity correspondence relation for grouping the nodes with similar descriptions from different models, rather than only focusing on pairs of nodes. Then the problem of model merging is reduced to a weighted set packing problem by viewing the n-way merging as to assigning the correspondence nodes into a same tuple. The weight of a tuple depends on the number of the shared descriptors of the nodes in the tuple. The result is the tuple-set with the largest weighted-sum. This method can cope with multiple models and at the same time takes the descriptors of the nodes into account. But the structure, e.g., the links between nodes, has been ignored while it is obviously another important aspect of the model semantics.

In this paper, we model the diagram synthesis also as an optimization problem. We notice that the node descriptors, the links between nodes and the edge descriptors are equally important for the semantics of the diagrams in the viewpoint of requirements. Moreover, often, similar descriptors in natural language may have different meanings and different descriptors may be similar, depending on the context. Thus, instead of considering only how well the matching of the nodes is, we consider both the matching of the descriptions of nodes/edges and the structure of diagrams. This asks for a global measurement of the entire diagram, rather than only the comparison among nodes.

Furthermore, we argue that for requirements synthesis, apart from identifying and merging similar parts of the input diagrams, the resulting diagram which will serve as the system requirements should satisfy the following constraints.

- *Minimization.* The resulting diagram consists of as few nodes/edges as possible for ensuring non-redundancy of the system requirements [17].
- *Information Preservation.* The resulting diagram preserves the information of all the input diagrams [16].
- *Traceability.* Any element, the node or the edge, in the resulting diagram can be traced back to its source, i.e., one of the input diagrams [10].

Our setting is as follows. There are n participants involved in requirements collection. Focusing on a feature request, they express their own viewpoints about the features by inputting activity diagrams, which we call the input diagrams. Each of the diagrams is a directed graph with the actions or the conditions as the nodes and the action flows or the condition flows as the edges. The requirements synthesis is merging the input diagrams to produce a resulting diagram. We consider merging multiple diagrams simultaneously and reduce the diagram synthesis problem as a combinatorial optimization problem.

The genetic algorithm (GA) is good at solving optimization and has been applied successfully to many software engineering problems [18], e.g., finding the smallest set of test cases [19], finding the best allocation of resources [20], etc.. These studies evidence that the genetic algorithm can help to solve the software development problems. This paper aims to use the genetic algorithm to solve the problem of multiple diagram synthesis. This idea originates from previous

work [21], but this paper has made important extensions at the following key points.

• The features of diagram synthesis are systematically characterized and a novel labelled graph is proposed as the general representation for multiple diagram synthesis. This labelled graph explicitly captures the necessary features of the requirements synthesis.

• Based on the labelled graph representation, the genetic algorithm is redesigned, which makes the algorithm more general and the performance much higher.

• More case studies and experiments are included to verify the effect of the proposed approach.

As any genetic algorithm is featured by 1) the "genetic representation" of the "individuals" (i.e., the candidate solutions), 2) the "fitness function", and 3) the "genetic operators", the paper is characterized by the following three contributions.

• A labelled graph is proposed as the genetic representation for capturing the features of the diagram synthesis. In the labelled graph, each node/edge has a label which retains all the information about the corresponding node/edge in the input diagrams. With these, the labelled graph can answer the questions about the diagram synthesis: what the nodes/edges are, how they are described, why they are needed, who needs them, etc..

• A generalized entropy is given on the basis of the genetic representation. The generalized entropy originates from the information entropy [22] and extends the information entropy by taking into account the similarity of the nodes/edges, the structure of the labelled graphs, and the number of nodes and edges. A fitness function is defined based on the generalized entropy and serves to measure the quality of the candidate solutions.

• New genetic operators for the crossover and the mutation are defined over the labelled graph based genetic representation. That enables the evolution of the candidate solutions until reaching one of the termination conditions.

The effectiveness of the proposed approach is evaluated through experiments on four separate cases. The results show that our approach is able to find the resulting diagram with a good precision and recall within an acceptable time. And it is scalable to different sizes of the input diagrams.

The rest of the paper is organized as follows. We first provide the framework of our approach in Section 2, including the notations that are used in the rest of the paper. We detail our approach in Section 3 and present experimental evaluations of our approach in Section 4. Section 5 discusses some properties and the validity of the approach. We present related work in Section 6 and conclude our paper in Section 7.

2 Overview

This section presents the framework of the proposed diagram synthesis approach and gives the notations that will be used in the rest of the paper.

2.1 Framework

The framework is given in Fig.1. It is a process of accepting a set of input diagrams and producing a resulting diagram which combines the commonalities as many as possible while leveraging the variabilities. The four steps of the process are:

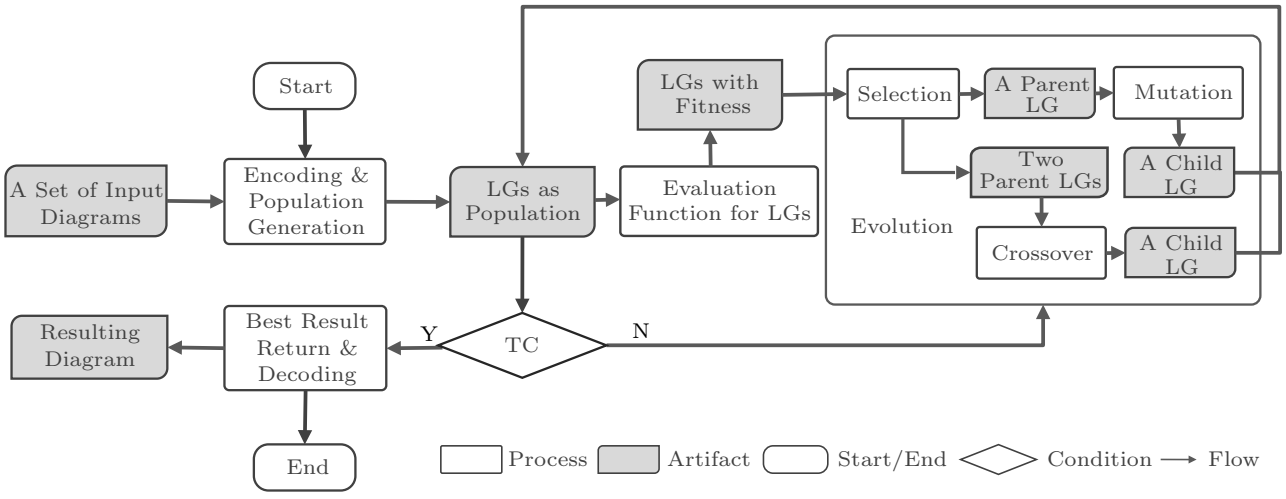

Fig.1. Process model of the diagram synthesis based on the genetic algorithm.

1) *Initiation*: encoding the set of input diagrams to obtain a set of labelled graphs and constructing the initial population;

2) *Evaluation*: evaluating each individual in current population with the fitness function to assign each individual a ranking score;

3) *Termination or Evolution*: terminating the process if the termination conditions hold (go to step 4); otherwise, producing the population of next generation through selection, crossover and/or mutation and undergoing next evolution (go to step 2);

4) *Decoding*: selecting the best individual in current population as the solution and decoding the labelled graph to obtain the resulting diagram.

2.2 Working Example

Before presenting the details of the approach, we first give a working example. It contains three activity diagrams collected from a requirements modeling task that is expressed by a user story "As a customer, I want to find a book and add it to my cart". Three activity diagrams are provided by three different providers and are used to express their expected workflows for the user story requirements. Fig.2 gives the three activity diagrams and shows the similar nodes which are aligned (connected with dashed lines), in which all nodes have been numbered in the form of "participant_number.node_number". Each node has a narrative description and a type. Each edge connects two nodes, and is also typed. The edge of a decision flow has a narrative description.

Obviously, among the diagrams, there are the commonalities and also the variabilities.

2.3 Notations

The notations that will be used in the rest of the paper are presented here.

Node/Edge Type. The nodes or the edges in the input models are typed:

• the types of a node: $T_V = \{s, e, sa, ua, d\}$, in which s stands for the "start" node, e stands for the "end" node, sa stands for the "system action" node, ua stands for the "user action" node, and d stands for the "decision" node;

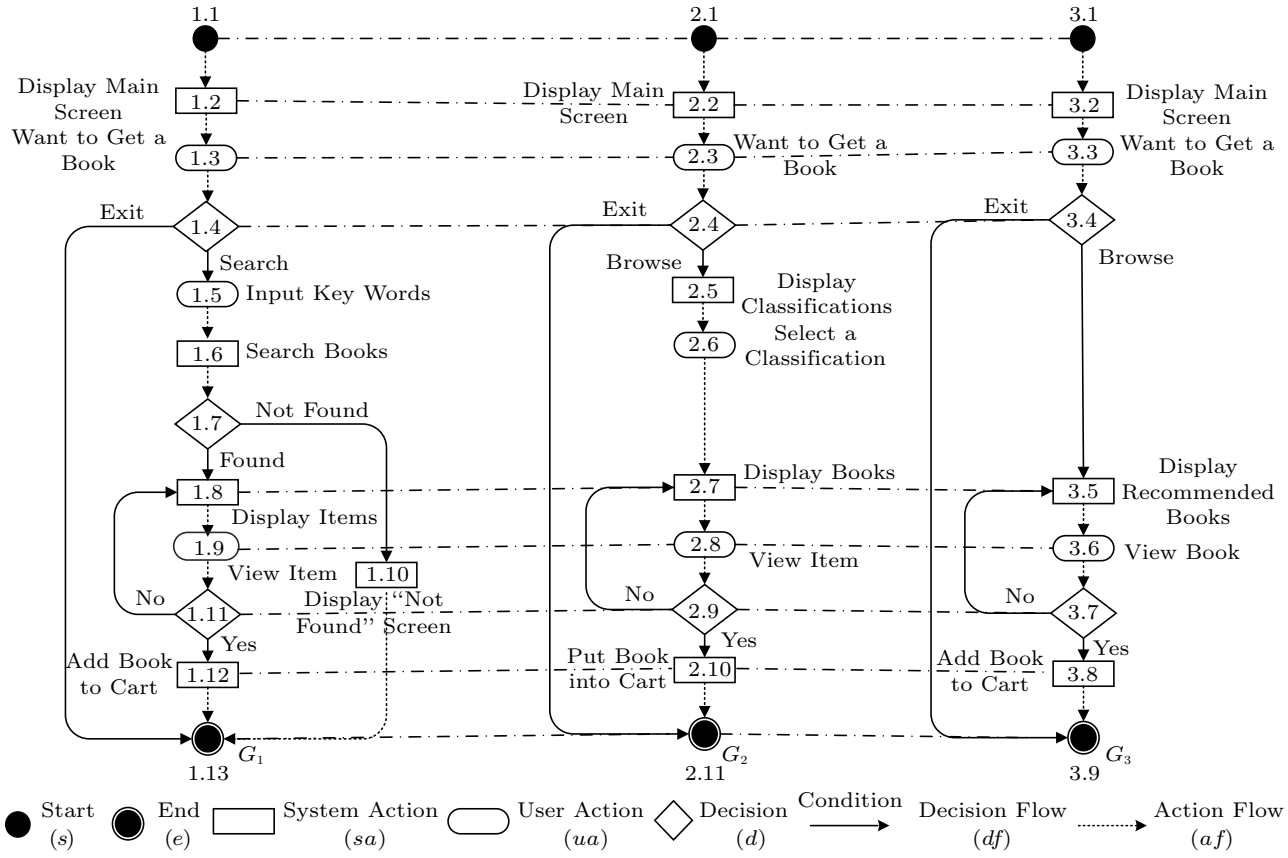

Fig.2. Three activity diagrams about an online shopping system. The similar nodes are aligned.

• the types of an edge: $T_E = \{af, df\}$, where *df* stands for the "decision flow" edge and *af* stands for the "action flow" edge.

Input Diagram. The input diagram from participant u is $G_u = (A_u, F_u)$, in which A_u is the set of typed nodes and F_u is the set of typed edges.

• For any $n \in A_u$, $n = \langle s(n), type, des\rangle$, in which $type \in T_V$, des is a narrative description, and $s(n)$ is in the form of $participant\_number.node\_number$, which gives a unique serial number to each node.

• For any $f \in F_u$, $f = \langle s(n_1) \to s(n_2), type, des\rangle$, in which $n_1, n_2 \in A_u$, $type \in T_E$ and des is a narrative description.

Labelled Graph (*LG*). A labelled graph corresponds to a set of input diagrams and represents a way of merging these input diagrams. It consists of the graphic elements, i.e., nodes and edges, and the labeling elements. A labeling function decides the links between each graphic element and its labeling element. Formally, let the set of input diagrams be

$$GS = \{G_{u_1}, G_{u_2}, ..., G_{u_N}\}.$$

A labelled graph corresponding to GS is defined as

$$LG(GS) = \{\mathcal{V}, \mathcal{E}, L_V, L_E\},$$

in which,

• $\mathcal{V}$ is a finite set of nodes;

• $\mathcal{E}$: $\mathcal{V} \times \mathcal{V}$ is a finite set of ordered pairs of nodes;

• L_V: $\mathcal{V} \to A^+_{u_1} \times A^+_{u_2} \times \cdots \times A^+_{u_N}$ is a labeling function of nodes, where,

1) $A^+_{u_i} = A_{u_i} \cup \{\tau\}$ $(1 \leqslant i \leqslant N)$, and τ is an empty action with *null* type, and

2) for any $v \in \mathcal{V}$, $L_V(v) \neq \tau^N$;

• L_E: $\mathcal{E} \to F^+_{u_1} \times F^+_{u_2} \times \cdots \times F^+_{u_N}$ is a labeling function of edges, where,

1) $F^+_{u_i} = F_{u_i} \cup \{\zeta\}$, and ζ is an empty edge with *null* type, and

2) for any $e \in \mathcal{E}$, $L_E(e) \neq \zeta^N$, and

3) for any $e = (n_1, n_2) \in \mathcal{E}$, $n_1, n_2 \in \mathcal{V}$, where $a \in L_V(n_1), b \in L_V(n_2)$, if $(a, b) \in L_E(e)$, then $\exists i (1 \leqslant i \leqslant N) \cdot (a, b) \in F_{u_i}$.

Obviously, a labelled graph has the following characteristics:

• each element (i.e., the node or the edge) in the labelled graph contains at least one source element from one of the input diagrams; and

• any element in input diagrams only appears once in the labelled graph.

That is, the labelled graph records the nodes and edges in all input diagrams and preserves the places of all the nodes and edges.

Type-Restricted Labelled Graph (*TLG*). A type-restricted labelled graph is a labelled graph in which each element consists of the nodes with the same type and the edges with the same type. We assume that the nodes/edges with type τ or ζ are of *null* type, which can match with any other types.

Formally, a labelled graph

$$LG(GS) = \{\mathcal{V}, \mathcal{E}, L_V, L_E\}$$

is type-restricted if the following conditions hold:

• for any $v \in \mathcal{V}$, if $n_1(= \langle \_, t_1, \_\rangle), n_2(= \langle \_, t_2, \_\rangle) \in L_V(v)$ are two non-τ nodes (for simplicity, we only highlight the to-be-explained field, and let the other field be "_"), then $t_1 = t_2$; and

• for any $e \in \mathcal{E}$, if $f_1(= \langle \_, t_1, \_\rangle), f_2(= \langle \_, t_2, \_\rangle) \in L_E(e)$ are two non-ζ edges, then $t_1 = t_2$.

Semantic-Restricted Labelled Graph (*SLG*). A labelled graph

$$LG(GS) = \{\mathcal{V}, \mathcal{E}, L_V, L_E\}$$

is semantic-restricted if the following conditions hold:

• for any $v \in \mathcal{V}$, if $n_1(= \langle \_, t_1, a_1\rangle), n_2(= \langle \_, t_1, a_2\rangle) \in L_V(v)$ are two non-τ nodes, then narrative descriptions a_1 and a_2 are sufficiently similar; and

• for any $e \in \mathcal{E}$, if $f_1(= \langle \_, t_1, a_1\rangle), f_2(= \langle \_, t_1, a_2\rangle) \in L_E(e)$ are two non-ζ edges, then narrative descriptions a_1 and a_2 are sufficiently similar.

Obviously, the labelled graph that is both type-restricted and semantic-restricted is the representation of the solution domain. Fig.3 shows a labelled graph corresponding to the working example which is type-restricted. The detailed information about the labelled nodes and the labelled edges is given in Table 1 and Table 2 respectively.

The labelled graph plays an important role in the genetic algorithm for requirement synthesis. Referring back to Fig.1, the encoding step accepts a set of input diagrams and generates the initial population in which each individual is a labelled graph. The evaluation step evaluates the labelled graphs and ranks the individuals and the evolution step evolves the population by performing genetic operators on the labelled graphs. And finally, the decoding step transforms the labelled graph into the resulting diagram.

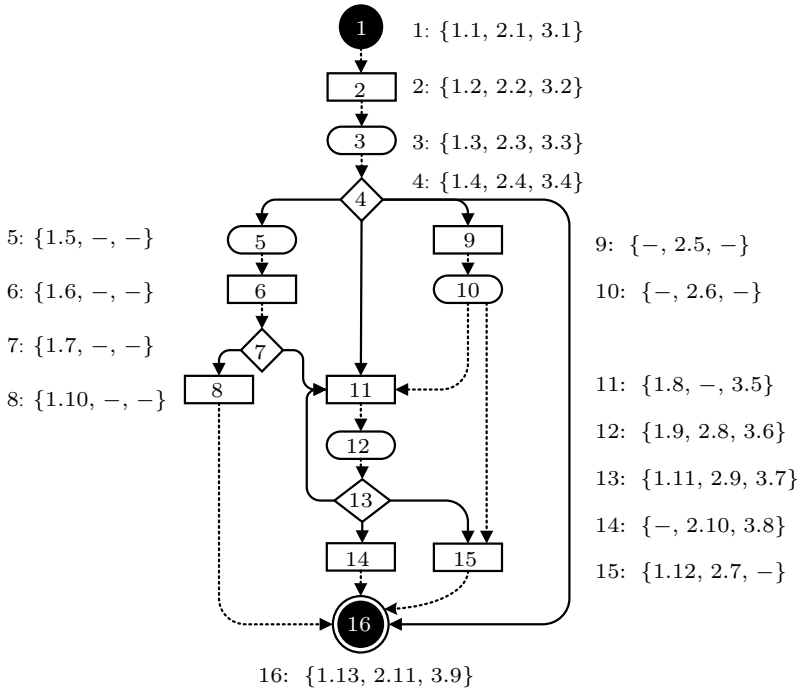

Fig.3. TLG for the three input diagrams in Fig.2.

Table 1. Detailed Information About Labelled Nodes in Fig.3

Node	Type	Description in G_1	Description in G_2	Description in G_3
1	*s*	Start	Start	Start
2	*sa*	Display main screen	Display main screen	Display main screen
3	*ua*	Want to get a book	Want to get a book	Want to get a book
4	*d*	Null	Null	Null
5	*ua*	Input key words		
6	*sa*	Search book		
7	*d*	Null		
8	*sa*	Display “not found” screen		
9	*sa*		Display classifications	
10	*ua*		Select a classification	
11	*sa*	Display items		Display recommended books
12	*ua*	View item	View item	View book
13	*d*	Null	Null	Null
14	*sa*		Put book into cart	Add book to cart
15	*sa*	Add book to cart	Display books	
16	*e*	End	End	End

Note: The blank in each row means there is no such node.

3 Genetic Algorithm

We use a genetic algorithm for searching the huge space of the synthesis of multiple diagrams to find a high-quality solution.

3.1 Overview

Before presenting the algorithm, we first give some parameters that are used by the genetic algorithm:

• the population size (K): it tells how many individuals there are in each generation's population, where each individual is an LG;

• the mutation probability (*mutationRate*): it tells how often the individuals will be mutated to prevent falling into local extreme;

• the maximum generation number (*maxGeneration*): it tells how many generations to run at most to allow the algorithm to return a solution;

Table 2. Detailed Information About Labelled Edges in Fig.3

Edge	Type	Edge from G_1	Edge from G_2	Edge from G_3
$1 \to 2$	af	1.1 → 1.2	2.1 → 2.2	3.1 → 3.2
$2 \to 3$	af	1.2 → 1.3	2.2 → 2.3	3.2 → 3.3
$3 \to 4$	af	1.3 → 1.4	2.3 → 2.4	3.3 → 3.4
$4 \to 5$	df	1.4 → 1.5 (search)		
$4 \to 9$	df		2.4 → 2.5 (browse)	
$4 \to 11$	df			3.4 → 3.5 (browse)
$4 \to 16$	df	1.4 → 1.13 (exit)	2.4 → 2.11 (exit)	3.4 → 3.9 (exit)
$5 \to 6$	af	1.6 → 1.7		
$6 \to 7$	af	1.7 → 1.8		
$7 \to 8$	df	1.7 → 1.10 (not found)		
$7 \to 11$	df	1.7 → 1.8 (found)		
$8 \to 16$	af	1.10 → 1.13		
$9 \to 10$	af		2.5 → 2.6	
$10 \to 15$	af		2.6 → 2.7	
$11 \to 12$	af	1.8 → 1.9		3.5 → 3.6
$12 \to 13$	af	1.9 → 1.11	2.8 → 2.9	3.6 → 3.7
$13 \to 11$	df	1.11 → 1.8 (no)		3.7 → 3.5 (no)
$13 \to 14$	df		2.9 → 2.10 (yes)	3.7 → 3.8 (yes)
$13 \to 15$	df	1.11 → 1.12 (yes)	2.9 → 2.7 (no)	
$14 \to 16$	af		2.10 → 2.11	3.8 → 3.9
$15 \to 16$	af	1.12 → 1.13	2.7 → 2.11	

Note: The blank in each row means there is no such edge.

• the minimum fitness (*minEntropy*): it tells an ideal fitness of the final solution; when finding such an individual, the algorithm returns the result;

• the successive iteration number (*plateauSize*) for deciding if the algorithm reaches a plateau: it defines the number of the successive iterations in which the best individuals do no longer produce better results.

Second, we define the stopping criteria. They are: 1) there is a solution that has *minEntropy* as its fitness value; 2) the maximum generation number (i.e., *maxGenertion*) is reached; 3) a plateau has reached (i.e., *plateauSize*). The algorithm will stop when one of the above three stopping criteria is satisfied.

With the parameters and the stopping criteria, the outline of the genetic algorithm is given as follows and Algorithm 1 shows the detailed process:

1) generating the initial population via encoding the set of the input diagrams (referring to Subsection 3.2);

2) evaluating each of the individuals in current population using the fitness function (referring to Subsection 3.3);

3) detecting if any of the stopping criteria is satisfied; if yes, terminating the process and returning the best individual in current population;

4) applying the genetic operators (referring to Subsection 3.4) and going back to step 2, if the algorithm does not terminated.

3.2 LGs As Initial Population

Given $GS = \{G_i = \langle A_i, F_i\rangle | 1 \leqslant i \leqslant m\}$ be a set of m input diagrams. From GS, we construct the initial population which contains K LGs as the individuals. The construction process is as follows.

1) Constructing type-restricted labelled graph nodes

• dividing $A_1 \cup A_2 \cdots \cup A_m$ into $|T_v|$ subsets N_t ($t \in T_v$) so that $N_t = \{\langle \_, t, \_\rangle | \langle \_, t, \_\rangle \in \bigcup_{i=1}^{m} A_i\}$; and

• for each N_t ($t \in T_v$), constructing a set of the "valid labelled nodes" so that $N(N_t) = \{V_t | (V_t \in \mathbb{P}(N_t) \wedge (\forall \langle i.s(n), \_, \_\rangle, \langle j.s(n'), \_, \_\rangle \in V_t, \text{where } i \neq j)\}(i, j \in [1, m])$.

2) Generating the individuals:

• an initial population $\mathcal{M}$ is a set of K labelled graphs. Each labeled graph consists of two sets:

- a set of valid labelled nodes $VS = \{V^1, \cdots, V^x\}$, in which $V^l \in \bigcup_{t \in T_v} N(N_t)$ where $1 \leqslant l \leqslant x$, $\bigcup_{i=1}^{n} V^i = \bigcup_{i=1}^{m} A_i$, and $\nexists n, n' \in VS$ so that $n = n'$;
- a set of labelled edges $E = \{E^1, ..., E^y\}$, in which E^i links V^s to V^t ($V^s, V^t \in VS$), $\forall \langle s(n_1) \to, s(n_2), \_, \_\rangle \in E^i$ where ($\langle s(n_1), \_, \_\rangle \in V^s$, $\langle s(n_2), \_, \_\rangle \in V^t$).

As the edges are defined by the nodes, we use the set of nodes to represent an individual.

Algorithm 1. Genetic Algorithm for Diagrams Synthesis

Input: $GS = \{G_1, G_2, ..., G_m\}$, a set of input diagrams
Output: RD, the resulting diagram
1: $\mathcal{M} = createInitialPopulation(GS, K)$ // refer to Subsection 3.2
2: **for** all $M_i \in \mathcal{M}$ $(1 \leqslant i \leqslant K)$ **do**
3: $e_i = calculateEntropy(M_i)$; // refer to Subsection 3.3
4: **end for**
5: $\mathcal{M}' = \{< M_i, e_i > | 1 \leqslant i \leqslant K\}$;
6: **while** not(any of the stopping criteria) **do**
7: $C \leftarrow \emptyset$;
8: **while** $|C| < K$ **do**
9: M', $M'' = selection(\mathcal{M}')$; $M = crossover(M', M'')$; // refer to Subsection 3.4
10: $C \leftarrow C \cup \{M\}$;
11: **end while**
12: **for** $i = 1$ to $mutationRate \times K$ **do**
13: $M = selection(\mathcal{M}')$; $M' = mutation(M)$; // refer to Subsection 3.4
14: $C \leftarrow \{C - \{M\}\} \cup \{M'\}$;
15: **end for**
16: $\mathcal{M} \leftarrow C$;
17: **for** all $M_i \in \mathcal{M}$ $(i = 1$ to $K)$ **do**
18: $e_i = calculateEntropy(M_i)$;
19: **end for**
20: $\mathcal{M}' = \{< M_i, e_i > | 1 \leqslant i \leqslant K\}$;
21: **end while**
22: Let M be the individual with minimum entropy in $\mathcal{M}$;
23: $RA = constructSynthesizedDiagram(M)$;
24: **return** RA

3.3 Fitness Function for LG

A fitness function is a particular type of objective function that is used to measure the quality of the solution. In our case, the design of the fitness function follows the criteria mentioned in Section 1. The basic idea is to view the descriptions of the elements and the flows in diagrams as random variables and to treat the requirements synthesis as the resolution of the inconsistency to find an optimal solution that can cover all submitted requirements descriptions.

Entropy refers to disorder or uncertainty and is a measurement which tells the degree of disorder or uncertainty in a system. Specifically, information entropy [22] is the average rate at which information is produced by a stochastic source of data. When multiple participants give their certain requests and the descriptions are synthesized according to certain criteria, the amount of information conveyed by such an event defined in this way becomes a random variable whose expected value is the information entropy.

Generally, the measure of the information entropy associated with each possible data value is the negative logarithm of the probability mass function for the value. But it does not consider the similarity between different values. When synthesizing the activity diagrams, it is necessary to take into account the similarity of the narrative descriptive statements annotated with the nodes/edges. We propose to use the generalized entropy to measure each possible element in the input diagrams, shown in (1), in which, X is the random variable with possible values $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$. $p : \mathcal{X} \to [0..1]$ is the possibility mass function of X. $s : \mathcal{X} \times \mathcal{X} \to [0..1]$ is a similarity function.

$$\mathcal{H}(X) = -\sum_{k=1}^{n} p(x_k) \log \left(\sum_{t=1}^{n} p(x_t) s(x_k, x_t) \right). \quad (1)$$

Accordingly, we design the fitness function as follows. Firstly, for an individual $M = (\mathcal{V}, \mathcal{E}, L_V, L_E)$ in a population $\mathcal{M}$, its entropy includes the entropies of all nodes as shown in (2), where $\mathcal{H}^n(v)$ is the node entropy, and $f(v)$ is an adjustment function. Each node has a set of incoming/outgoing edges as its context and is annotated with a set of descriptive statements. The entropy of a node is computed by (3), where $\mathcal{H}^n_d(v)$ is the entropy in the aspect of the node descriptions, $\mathcal{H}^n_c(v)$ is the context entropy, and α and $(1-\alpha)$ are the weights of these two aspects respectively.

$$\mathcal{H}^g(LG(GS)) = |\mathcal{E}| \sum_{v \in \mathcal{V}} (\mathcal{H}^n(v) + f(v)), \quad (2)$$

$$\mathcal{H}^n(v) = \alpha \mathcal{H}^n_d(v) + (1 - \alpha) \mathcal{H}^n_c(v). \quad (3)$$

Secondly, the context entropy consists of three parts for measuring uncertainty of the incoming-edges, the outgoing-edges and the edge-descriptions respectively. The incoming edges and the outgoing edges connected

to node v are grouped by their types. (4) is for computing the context entropy, where $\mathcal{H}_t^e(e^i)$ is for the incoming edges with type t, $\mathcal{H}_t^e(e^o)$ is for the outgoing edges with t, and $\mathcal{H}_d^e(e)$ is for the descriptions of edge e.

$$\mathcal{H}_c^n(v) = \sum_{t\in T_E} \mathcal{H}_t^e(e^i) + \sum_{t\in T_E} \mathcal{H}_t^e(e^o) + \sum_{t\in T_E} \mathcal{H}_d^e(e). \quad (4)$$

Finally, the node/edge description entropy and the incoming/outgoing edge entropy are given referring to (1). (5) shows the description entropy of the node/edge, where X represents a random variable of the node/edge description, and $s_d(x_k, x_t)$ represents the similarity between two descriptions x_k and x_t.

$$\mathcal{H}_d^{(n/e)}(X) = -\sum_{x_k\in X} p(x_k) \log \left(\sum_{x_t\in X} p(x_t) s_d(x_k, x_t) \right). \quad (5)$$

In order to calculate the similarity between two descriptions x_k, x_t, we use a word-to-word similarity matrix based approach [23]. This approach represents two descriptions with two sets of words, and some stop words are removed. (6) is used to calculate the similarity of the two sets of words, i.e., to accumulate the similarities of all word pairs with the largest similarity and average the accumulated result as the similarity of two sets of words. The similarity $s_w(w, w')$ of a pair of words w, w' equals the average value of their N-gram [24] similarity and WordNet① similarity.

$$s_d(x_k, x_t) = \left(\sum_{w\in x_k} \max_{w'\in x_t} (s_w(w, w')) + \sum_{w\in x_t} \max_{w'\in x_k} (s_w(w, w')) \right) /2. \quad (6)$$

For a set of incoming (or outgoing) edges E of type t and connected with node v, (7) is used to measure the entropy of the incoming (or outgoing) edge. In (7), $x_i \in \mathbb{P}(E)(x_i \in X)$, and $s_e(x_k, x_t)$ is the similarity between two edge sets x_k and x_t. We use the Jaccard similarity [25] method to calculate the similarity of two edge sets.

$$\mathcal{H}_t^e(X) = -\sum_{x_k\in X} p(x_k) \log \left(\sum_{x_t\in X} p(x_t) s_e(x_k, x_t) \right). \quad (7)$$

Fig.4 shows the details of a node numbered 13 in the labelled graph that is described in Fig.3. The node consists of three nodes from the three input diagrams in Fig.2 and connects with four edges e_1, e_2, e_3 and e_4. These labelled edges are classified into two types (*incoming af* and *outgoing df*), in which e_1 belongs to the *incoming af* type and other edges belong to the *outgoing df* type. Entropy of each edge type is calculated with (7) and shown in Table 3. Description entropy of each labelled edge is calculated with (5) and shown in Table 4. For the valid labelled node 13, node-description entropy $\mathcal{H}_d^n(v) = 0$, incoming-edge entropy $\mathcal{H}_{af}^e(e^i) =$

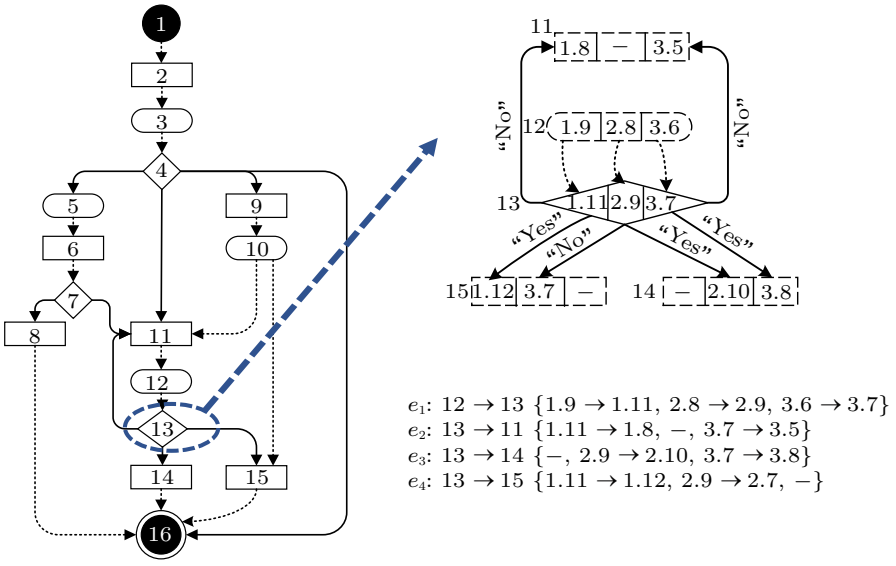

Fig.4. Labelled graph of the three diagrams in Fig.3.

①http://wordnet.princeton.edu, June 2020.

Table 3. Entropy of Each Edge Type for Node 13 in Fig.4

Edge Type	Edge Set Provided by Input Models	Probability	Similarity	Entropy
Incoming af	$G_1, G_2, G_3 : \{e_1\}$	$p(\{e_1\})$=1	–	$\mathcal{H}^e_{af}(e^i) = -1 \times \log(1) = 0$
Outgoing df	$G_1 : \{e_2, e_4\}$; $G_2 : \{e_3, e_4\}$; $G_3 : \{e_2, e_3\}$	$p(\{e_2, e_4\}) = 1/3$; $p(\{e_3, e_4\}) = 1/3$; $p(\{e_2, e_3\}) = 1/3$	$s(\{e_2, e_4\}, \{e_3, e_4\}) = 1/3$; $s(\{e_2, e_4\}, \{e_2, e_3\}) = 1/3$; $s(\{e_3, e_4\}, \{e_2, e_3\}) = 1/3$	$\mathcal{H}^e_{df}(e^o) = -1/3 \times \log(1/3 \times 1/3 + 1/3 \times 1/3 + 1/3 \times 1) - 1/3 \times \log(1/3 \times 1/3 + 1/3 \times 1/3 + 1/3 \times 1) - 1/3 \times \log(1/3 \times 1/3 + 1/3 \times 1/3 + 1/3 \times 1) = 0.26$

Table 4. Description Entropy of Each Labelled Edge for Node 13 in Fig.4

Edge	Description	Probability	Similarity	Entropy
e_1	Null	$p(\text{null}) = 1$	$s(\text{null}, \text{null}) = 1$	$\mathcal{H}^e_d(e_1) = -1 \times \log(1) = 0$
e_2	G_1: "no"; G_2: "no"; G_3: null	$p(\text{"no"}) = 2/3$; $p(\text{null}) = 1/3$	$s(\text{"no"}, \text{"no"}) = 1$; $s(\text{"no"}, \text{null}) = 0$; $s(\text{null}, \text{null}) = 1$	$\mathcal{H}^e_d(e_2) = -2/3 \times \log(2/3 \times 0 + 1/3 \times 1) - 1/3 \times \log(1/3 \times 0 + 2/3 \times 1) = 0.26$
e_3	G_1: null ; G_2: "yes"; G_3: "yes"	$p(\text{null}) = 1/3$; $p(\text{"yes"}) = 2/3$	$s(\text{"yes"}, \text{"yes"}) = 1$; $s(\text{"yes"}, \text{null}) = 0$; $s(\text{null}, \text{null}) = 1$	$\mathcal{H}^e_d(e_3) = -1/3 \times \log(1/3 \times 0 + 2/3 \times 1) - 2/3 \times \log(2/3 \times 0 + 1/3 \times 1) = 0.26$
e_4	G_1: "yes"; G_2: "no"; G_3: null	$p(\text{null}) = 1/3$; $p(\text{"yes"}) = 1/3$; $p(\text{"no"}) = 1/3$	$s(\text{"yes"}, \text{"no"}) = 0.23$; $s(\text{"yes"}, \text{null}) = 0$; $s(\text{"no"}, \text{null}) = 0$; $s(\text{null}, \text{null}) = 1$	$\mathcal{H}^e_d(e_4) = -1/3 \times \log(1/3 \times 0.23 + 1/3 \times 0 + 1/3 \times 1) - 1/3 \times \log(1/3 \times 0.23 + 1/3 \times 0 + 1/3 \times 1) - 1/3 \times \log(1/3 \times 0 + 1/3 \times 0 + 1/3 \times 1) = 0.417$

0, outgoing-edge entropy $\mathcal{H}^e_{ef}(e^o) = 0.26$, and edge-description entropy $\mathcal{H}^e_d(e) = 0.26 + 0.26 + 0.42 = 0.94$. As the result, the node 13's entropy is $\mathcal{H}^n(v) = 1.2$.

3.4 Genetic Operators for New LGs

Genetic operators are used to create new LGs for population evolution. There are three operators, i.e., the crossover, the mutation and the selection, which work in conjunction with one another. Among them, the selection is simply for selecting proper individuals from current population. The Roulette-wheel selection is used for this purpose [26].

The crossover operator selects two individuals from current population as parents and produces a child. The selection operator needs to be conducted $2 \times K$ times for getting K pairs of individuals for producing K individuals for next generation.

For conducting the crossover, we adopt an approach proposed in [27]. After selecting two individuals M_1 and M_n from current population, a shortest path P from M_1 to M_n is found first. Each node on P is a labelled graph, and two adjacent nodes M_i and M_{i+1} on P differ by a "transposition", i.e., swapping two nodes $\langle k.s(n_1), t, \_\rangle$ and $\langle k.s(n_2), t, \_\rangle$ between two valid labeled nodes V and V' in M_i. The midpoint $M_{\lfloor n/2 \rfloor}$ on P is obtained as the child individual of M_1 and M_n. That is, $M_{\lfloor n/2 \rfloor}$ is expected to share approximately half of M_1 and M_n.

The mutation operator is for preserving the diversity of individuals. Given the mutation probability *mutationRate*, the mutation operator selects *mutation Rate* $\times K$ individuals and alters one or more gene values to produce new individuals.

In more detail, we select individual M from current population. Two types of mutation operations are randomly performed to get an individual M': 1) swapping the nodes $\langle k.s(n_1), t, \_\rangle$ and $\langle k.s(n_2), t, \_\rangle$ between two valid labelled nodes V^i and V^j in M and obtaining M', where $\langle k.s(n_1), t, \_\rangle \in V^i, \langle k.s(n_2), t, \_\rangle \in V^j$, $i \neq j$; 2) removing node $\langle k.s(n), t, \_\rangle$ from V ($V \in M$), and inserting a new element $V' = \{\langle k.s(n_1), t, \_\rangle\}$, and obtaining a new individual M', where $M' = (M - \{V\}) \cup \{(V - \{\langle i.s(n), t, \_\rangle\}) \cup V'\}$.

4 Evaluation

We conduct experimental studies on four cases to evaluate the effectiveness of our approach for requirements synthesis.

4.1 Experimental Setting and Research Questions

We implement the proposed approach by Java②. The tool can run on the platform of Intel® Core™

②https://github.com/ciecwch/ActivitySynthesis, June 2020.

CPU, 3.40 GHz, 16 GB memory, running Microsoft Windows 10. The followings are three research questions (RQs).

• *RQ*1. How long does it take for the synthesis algorithm to generate the resulting diagram from the input diagram sets with different sizes of the input diagrams?

• *RQ*2. How is the synthesis effect of the resulting diagram?

• *RQ*3. Can a solution be obtained for each execution?

Four case studies are used to quantitatively assess the performance of our approach. They are selected from four different merging tasks and have different numbers of the input diagrams and different numbers of nodes/edges in the input diagrams. The four merging tasks are as follows.

1) *WE* (*Working Example*). As a customer, I want to find a book and add it to my cart.

2) *MS* (*Mobile Shopping App*). As a registered member, I want to buy something in a mobile mall.

3) *ATM* (*Automated Terminal Machine*). As a card holder, I want to withdraw money at the ATM.

4) *RA* (*Web-Based Reservation Application*). As a registered member, I want to order a meal on a reservation booking application.

In these four cases, the input diagrams of one case (ATM) come from a use-case specification, and the input diagrams of the other cases come from three student groups. For the ATM case, we get nine scenarios from the two use cases of "withdrawn fund" and "validate PIN" in a bank ATM use-case specification[③] and convert these nine scenarios into nine input diagrams. In order to obtain the input diagram sets of the other three experimental cases, we recruit 12 experienced students in software development, and divided them into three groups of 3, 4 and 12 to provide needs. We collect three sets of scenario documents from the three groups. In these scenarios, each scenario represents a sequence of actions and interactions between actors and the system according to the scenario format in UML use case description[④]. To get an input diagram from a scenario, we use a semi-automatically approach based on a tagging scenario. The approach mainly includes two steps: the first step is to tag the types of nodes and edges in the scenario description; the second step is to automatically convert the scenario into an input diagram. Tables 5–8 give the features of the activity diagrams in the four cases, respectively.

Table 5. Features of the Activity Diagrams in WE Case

ID	SB	BB	AB	#node	#edge
1	✓		✓	13	15
2		✓	✓	11	12
3		✓	✓	9	10

Note: SB: search book; BB: browse book; AB: add a book. #: number of.

Table 6. Features of the Activity Diagrams in MS Case

ID	R	L	BG	SG	AC	DG	CO	MO	DO	PO	#node	#edge
1	✓	✓	✓	✓			✓	✓	✓	✓	57	64
2	✓	✓	✓	✓	✓	✓					51	55
3	✓	✓	✓				✓		✓	✓	57	63
4	✓	✓	✓	✓	✓	✓					57	65

Note: R: register; L: login; BG: browse good; SG: search good; AG: adding good to the shopping cart; DG: delete good; CO: create order; MO: modify order; DO: delete order; PO: pay for order; #: number of.

Table 7. Features of the Activity Diagrams in ATM Case

ID	WF	VP	#node	#edge
1	✓		17	16
2	✓		16	15
3	✓		10	9
4	✓		12	11
5	✓		14	13
6		✓	18	17
7		✓	9	8
8		✓	14	13
9		✓	16	15

Note: WF: withdrawn fund; VP: validate PIN.

Table 8. Features of the Activity Diagrams in RA Case

ID	CR	CD	AC	EC	CP	CO	PO	#node	#edge
1		✓	✓	✓				17	19
2		✓	✓					12	15
3	✓	✓	✓			✓	✓	15	17
4		✓	✓		✓	✓		22	28
5		✓	✓		✓	✓	✓	11	13
6	✓	✓	✓			✓	✓	14	17
7	✓	✓	✓	✓		✓		18	22
8	✓	✓	✓			✓		19	24
9		✓	✓					17	22
10	✓	✓	✓	✓	✓	✓		23	29
11		✓	✓	✓		✓		16	19
12		✓	✓			✓		22	27

Note: CR: choose restaurant; CD: choose dish; AC: add dish to cart; EC: edit cart; CP: choose preference dish; CO: create order; PO: pay for order.

③http://www.cs.fsu.edu/baker/swel/restricted/templates/rr631gv1_stuwrk_withdraw_cash_use_case_spec.pdf, June 2020.

④https://courses.cs.washington.edu/courses/cse403/15sp/lectures/L4.pdf, June 2020.

For the above four cases, we use the diagram synthesis approach proposed in this paper to get a resulting diagram. The parameters of this approach are given in Table 9, where $graphNum$ represents the number of the input diagrams.

Table 9. Parameters of GA for Case Studies

Parameter	Value
K	$graphNum \times 50$
$maxGeneration$	$graphNum \times 100$
$mutationRate$	0.1
$platauSize$	100
α	5
$minEntropy$	0.3

4.2 Results and Analysis

We analyze the synthesis process and the final synthesis results of the four cases, and answer the three research questions. In fact, we execute the program for five times, calculate the average time for finding the final solution, and observe the fitness evolution towards the final solution, and then answer RQ1 (see Subsection 4.2.1). Next, to answer RQ2, we measure the synthesis quality of the final solutions with precision and recall (see Subsection 4.2.2). Finally, we analyze the entropy fed back by the five executions, and answer RQ3 (see Subsection 4.2.3).

4.2.1 Fitness Evolution

After each execution, a final solution will be obtained. Table 10 shows the related values for each case, where $\#TN/\#TE$ is the number of nodes/edges in the input diagrams, $\#AN/\#AE$ is the average number of nodes/edges in five resulting diagrams fed back by five executions, AF is the average fitness value of the five resulting diagrams, $\#AG$ is the average generation number of the five executions, and AT is the average time of the five executions.

The results show that the execution time is related to the size of the input diagrams (the number of the input diagrams and the number of nodes in all input diagrams). The larger the size of input diagrams, the longer the execution time of the process. In the WE case, the execution takes about 0.34 minutes for getting the resulting solution. However, in the RA case, the execution takes about 45 minutes. It is also acceptable for the requirements synthesis task with 12 input diagrams and 206 nodes to be synthesized.

In order to observe the changes of fitness value on each execution (fitness evolution), we randomly select one of the five executions, and record the fitness values of the best individual in each generation. Fig.5 shows the best fitness value in each generation, where the horizontal axis is the generation number, and the vertical axis is the fitness value of the best individual in each generation.

The results show that the process can obtain a solution in an acceptable time. The evolution generations are related to the size of the diagrams. The smaller the size, the fewer the iterations needed, and the larger the size, the more the iterations required.

4.2.2 Synthesis Quality Evaluation

In order to evaluate whether the final solution is good enough, we use "precision" and "recall" as the indicators to evaluate resulting diagram.

Precision is used to evaluate the correctness of the final solution. Supposing that the resulting diagram is $(\mathcal{V}, \mathcal{E}, L_V, L_E)$, precision is the proportion of correct valid labelled nodes CV in $\mathcal{V}$, calculated by (8). The correct valid labelled nodes need to be confirmed by requirements engineers.

$$precision = |CV|/|\mathcal{V}|. \tag{8}$$

Recall is used to measure the coverage. It is quantified with the ratio of the number of correctly labelled nodes to the number of expected valid labelled nodes EV, calculated by (9). The expected node is from an "expected output diagram" that is a synthesized diagram constructed manually by requirements engineers.

$$recall = |CN|/|EN|. \tag{9}$$

Table 11 shows the precision and the recall for each of the four cases. In Table 11, for each case, two final

Table 10. Features of the Resulting Diagram

Case	$\#TN$	$\#AN$	$\#TE$	$\#AE$	$\#AG$	AF	AT (m)
WE	33	22.0	37	30.00	45	1 009.1	0.34
MS	222	89.0	247	119.25	740	9 659.2	35.80
ATM	126	26.4	117	34.00	549	2 353.9	22.26
RA	206	32.0	252	111.60	1 200	18 592.0	44.53

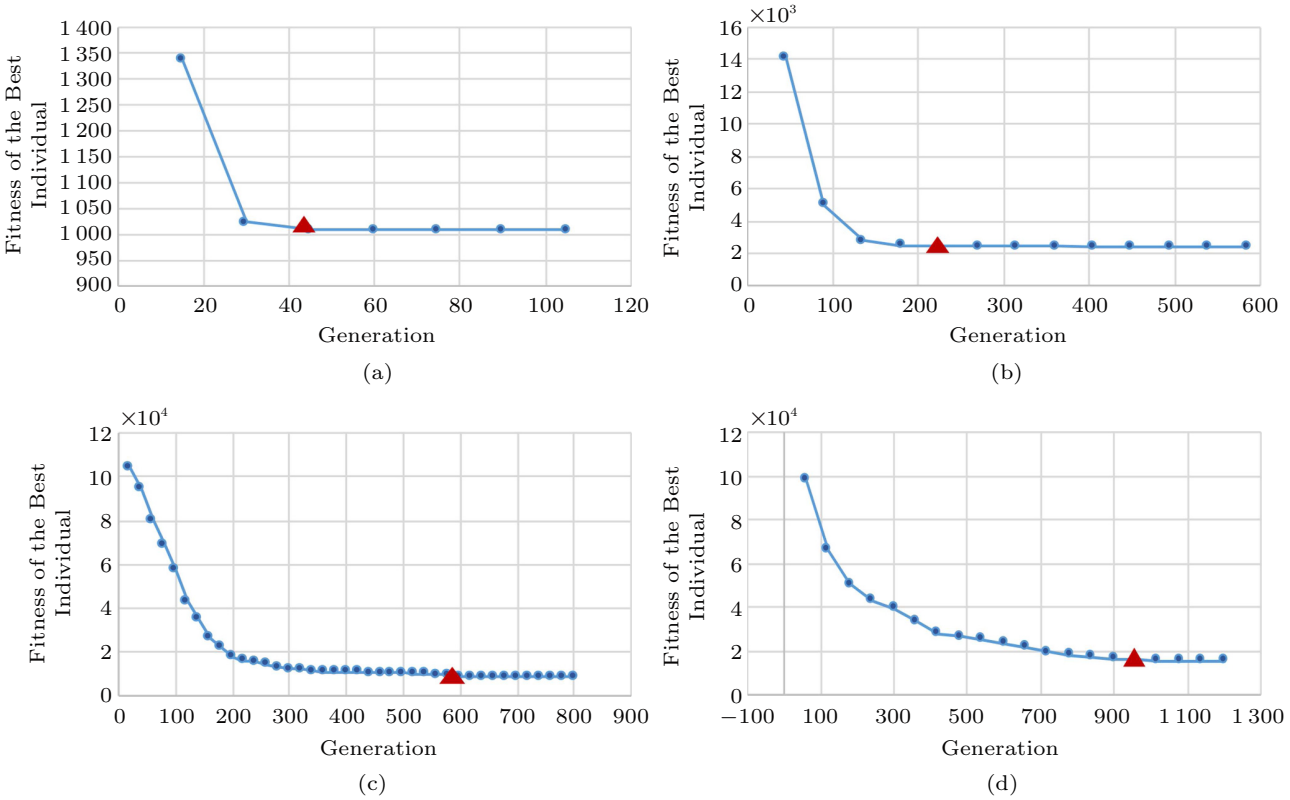

Fig.5. Evaluation of GA. (a) WE. (b) ATM. (c) MS. (d) RA.

Table 11. Precision, Recall and the Related Attributes for the Four Cases

Case	Solution	$\#AN$	$\#SN$	$\#CN$	$\#MN$	$\#EN$	$\#PN$	Precision (%)	Recall (%)
WE	M_1	12.7	22	22	0	0	22	100.00	100.00
	M_2	12.7	22	22	0	0	22	100.00	100.00
MS	M_1	55.5	76	68	4	0	74	89.47	91.89
	M_2	55.5	85	69	8	0	74	81.18	93.24
ATM	M_1	14.0	25	19	1	4	37	76.00	51.40
	M_2	14.0	32	29	0	3	37	90.60	78.30
RA	M_1	17.2	42	34	1	6	43	80.95	79.07
	M_2	17.2	35	27	0	8	43	77.14	62.79

solutions in the five executions are selected: one (M_1) has the minimum fitness value, the other (M_2) has the maximum fitness value. $\#AN$ is the average number of nodes in the set of input diagrams. $\#SN$ is the number of the nodes in the solution. $\#CN$ is the number of the correctly labelled nodes. $\#MN$ is the number of the missing valid labelled nodes, i.e., the node should be synthesized but not be synthesized. $\#EN$ is the number of the incorrect valid labelled nodes (some nodes are wrongly synthesized). $\#PN$ is the number of the perfect valid labelled nodes in the expected output diagram.

The results show that the precision and recall of the solution obtained by our approach is related to the total number of the nodes/edges in the input diagrams. The smaller the number, the better the synthesis quality. In general, the smaller the fitness value, the better the synthesis quality, but there will be a case of "over-synthesis", that is, some nodes with similar structures but different meaning may be merged together. For example, in the ATM case, M_2 has better precision and recall than M_1.

4.2.3 *Stability of the Approach*

To evaluate the stability of our approach, we analyze the fitness of five executions for each case. Fig.6

shows the distribution of the fitness value of final solutions from the five executions. The results show that the best solution of WE is achieved with each execution. As the total number of the nodes/edges in the input diagrams grows, it is difficult for the algorithm to get the optimal solution on every execution. However, it can be seen from Fig.6 that the fitness value of the solution returned by the algorithm is distributed over a small range, that is, the algorithm can obtain a similar solution on most executions, indicating that the process is stable.

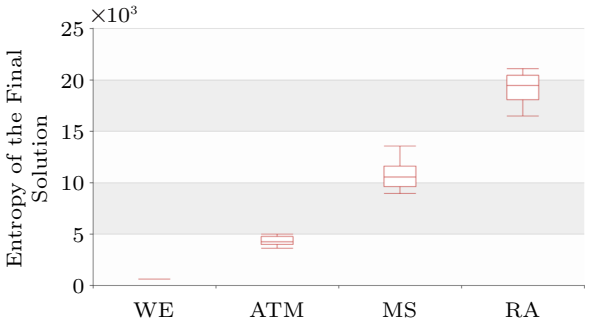

Fig.6. Stability of our approach on the four cases.

5 Discussions

In this section, we discuss the properties and the validity of the proposed approach.

5.1 Features of the Approach

As mentioned in Section 1, the synthesis of requirements needs to satisfy some properties, i.e., the information preservation, the traceability and the minimization.

Information preservation means that all the inputted requirements descriptions collected from the participants need to be preserved as they are the participants' desires from different viewpoints. Information preservation is for ensuring the requirements completeness. Our approach proposes to serialize the input diagrams, as well as the nodes and the edges from any input diagrams. The approach gives each of the nodes a unique number (the edges are decided by nodes). We propose a structure of the labelled graph that records with each node the node numbers of the input diagrams which contribute to the node. And with the evolution process, the selecting operation makes choice but all the information with the labels is retained. In the whole process, any information proposed by the participants can be retrieved by encoding and selecting although the genetic algorithm is used to generate high-quality solutions based on an optimization objective. In this sense, the information proposed by the participants is preserved.

Requirements traceability means that any requirements description can have a trace indicating its source (sometimes in both a forward and a backward direction). It is valuable in requirements confirmation where it helps to identify who needs the requirements (as well as how to implement the requirements). In our approach, requirements traceability means that any node/edge in the resulting diagram can be traced back to its source in the input diagrams, and each node/edge in the input diagrams can trace to its provider. The traceability is also supported by a way of encoding the input diagrams. The best solution obtained from the genetic algorithm can tell who contributes to the resulting labelled graph, which part of the result comes from which input diagrams, i.e., the nodes (and the edges) in the resulting diagram can be traced back to their source.

Requirements synthesis integrates the commonalities of the input diagrams. Requirements minimization means that the resulting diagram needs to contain as few nodes/edges as possible for ensuring non-redundancy on a global view. In our approach, we propose a novel generalized entropy for the measure of the "disorder" of the intermediate labelled graphs. The labelled graph with the smaller entropy means to be of less disorder; thus more real commonalities are integrated with less confusion. The definition of the generalized entropy measurement shows that the entropy considers each of the nodes (with the type and the descriptors) as well as its contexts. This makes it a global measurement for ensuring the minimization of the solution.

5.2 Scalability

Table 10 shows that the execution time of our approach relates to the size of the diagrams. It performs very well when the total number of the nodes/edges is not very large (e.g., when the number of nodes/edges is 33/37 in WE). When the total number of the nodes exceeds 100, the execution time is in the range of tens of minutes. However, for example, user stories normally include a written sentence or two and a series of conversations about the desired functionality. The derived diagrams would not be too big. And requirements synthesis is not a time-critical activity and in this sense, the execution time is not very critical.

In crowd-based requirements engineering or Agile requirements, a large set of requirements descriptions may be obtained. In many cases, these requirements descriptions may be related to different functions and then can be grouped into several subsets in terms of the functions [28]. Each subset may contain less requirements descriptions, and then the algorithm can perform well on it. The resulting synthesized diagrams of the subsets will constitute the synthesized requirements.

5.3 Extendability

The proposed approach can be easily extended to accommodate other kinds of requirements diagrams as the labelled graph has sufficient expressiveness. For example, if the participants provide a set of statecharts, we can also represent the statecharts in labelled graphs but with different types (of the nodes and the edges). Then the algorithm can be used onto the labelled graphs corresponding to the input statecharts. However, for different input diagrams, the fitness function may need to be adjusted accordingly based on the diagram semantics.

5.4 Diversity

One of the good features of the proposed approach is allowing the diversity of the requirements. In crowd-based requirements engineering, the requirements come from different participants. They may use different terms to describe the requirements at different granularities. Some traditional requirements engineering approaches treat the differences as conflicts. But in many cases, these different requirements descriptions are not logical conflict but are just opinions from different angles. These different opinions may need to be reserved, especially in the crowd-based requirements engineering. Our approach uses the similarity to measure the degree of the commonality and combines the commonalities while leveraging the variability.

In fact, there will be some logical conflicts between the diverse requirements of a distributed user group. In this paper, we consider the logical conflicts as different opinions, and use the synthesis approach to combine the commonalities while leveraging the variability. For continuing this work, through the resulting diagram, we plan to analyze the differences to find conflicts.

5.5 Limitations and Threats to Validity

It is noticed that the parameter setting may affect the execution time and the quality of the resulting diagram. In our approach, the parameters include the mutation rate, the population size, the terminal condition, the coefficients (α) in fitness calculation, and so on. The parameter setting depends on the size of the diagrams to be synthesized. When the size is large, some parameter needs to increased, e.g., the population size, the maximum generation of GA execution, and the successive iteration number of the best solution. The coefficient α is another important parameter to synthesis process, with some of the experience we have summarized in our experiments, and we set the value between 5 and 10 so that GA can quickly get a good result.

Whether the activities described in different input activity diagrams use the same granularity affects the synthesis quality of the resulting diagram. In the four cases in this paper, the activity diagrams in each case have the same granularity. In order to help achieve consistence in granularity, we assign a feature request to participants and provide a document to suggest the participants write their needs in the format of scenarios. We also provide the writing style of "verb-object phrase" to express actions and interactions between them.

Our experiments are actually controlled experiments and involve a small number of students in the same lab. In order to minimize the possible biases and deviations, several strategies have been adopted, including: 1) asking each student to complete his/her task independently; 2) assigning two students to tag the submitted information for converting the scenario documents into the input diagrams; 3) assigning another student to calculate the precision and recall of the resulting diagram.

6 Related Work

Model merging is an interesting topic in the field of requirements modeling and can be a useful mechanism in crowd-based requirements engineering.

In Crowd RE, recent research has been directed towards involving users in evolving requirements for large software. Groen *et al.* [3] proposed a framework to support the communication of participants by eliciting and analyzing user feedback. The user feedback maybe includes the software problem and extension ideas (feature requires or new products). The analyzing process needs to filter out irrelevant data and automatically classify the remaining data. Lim and Anthony [29] utilized the social network to collect user feedback for supporting requirements elicitation. The user feedback

includes the user stories provided by stakeholders, a rated requirements list and recommended requirements. Breaux and Schaub [30] studied the feasibility of distributing requirements extraction to the crowd. They evaluated crowdsourcing by a manual requirements extraction task to a larger number of untrained workers.

Most existing approaches for analyzing user feedback predominantly use the manual approach or linguistic analysis techniques such as text mining [31]. Manual analysis cannot quickly respond to the processing of large amounts of data in Crowd RE. The linguistic analysis technique only is suitable for processing text data. And, model data plays an important role in requirements engineering. We propose an automated approach to synthesize the requirements gathered from multiple providers. We focus the model merging, but not text merging.

Various studies [32] have been done on merging model variants/versions into one merged model. Numerous approaches for model merging focus on merging two models [10, 15, 32]. Merging of multiple models normally is based on the binary strategy to step-by-step merge two models. The merged results using the binary merging strategy may be diverse with different input orders of multiple models [17].

Rubin and Chechik [17] proposed using tuples to represent the merging problem of class diagram. They designed an optimal approach to produce a maximal weight set of tuples. The measurement of weight is based on the calculation of the similarity about the properties (such as class names, attributes) in each tuple. Considering that the node descriptors, the links between nodes and the edge descriptors are equally important for the semantics of the activity diagrams, global measurement of nodes and edges is required in activity diagram synthesis. We reduce the activity diagram synthesis problem as a combinatorial optimization problem and propose a genetic algorithm for solving this problem.

Our approach is based on GA. In model merging, several GA-based approaches have been proposed to merge multiple version models. Assunção *et al.* [33] proposed a GA-based approach to deal with the merging of multiple UML class diagram variants. This approach identifies differences between models and uses these differences to measure the fitness for finding a global model with minimizing the difference value. Kessentini *et al.* [32] proposed a GA approach to merge models based on sequences of operations that originate different models. This approach is designed for finding a sequence of operations to generate a merged model in order to minimize the number of conflicts and maximize the number of successfully applied operations. Debreceni *et al.* [34] provided rule-based design space exploration where candidate models are generated to reach a conflict-free merged model to support collaborative model-driven engineering.

Most of above search-based merging approaches represent the model merging problem with a set of operations that generate multiple variants by operating on a base version. These approaches are more suitable for solving multi-version model merging problems, where the operation histories need to be recorded and the information of source input models are difficult to track. We use the labelled graph to represent the requirements synthesis, where the information of input diagrams is preserved and can be traced.

7 Conclusions

Many new applications, such as web-based applications, mobile applications, and so on, serve for a large number of distributed users. It is better to use the crowd-based approach to collect the requirements. Automated requirements synthesis is important. This paper proposed a novel approach to synthesize the activity diagram by using the genetic algorithm. A labelled graph was designed to represent a synthesis of a set of input activity diagrams. A fitness function based on generalized entropy was proposed to measure the quality of the labelled graph. The genetic operators (i.e., crossover and mutation) were designed to produce legal variant individuals to promote evolution.

We conducted experimental research for evaluating the proposed approach by analyzing four cases from different scenarios and with different scales. The results showed the competency of this approach in aspects of performance, quality, stability, and scalability. Moreover, the structured representation of the diagram synthesis brings great convenience in the management and evolution of the distributed graphical models.

There are several directions for continuing this work. First, we are interested in exploring additional heuristics that could further improve the speed of getting the final solution. For example, one could be to classify the activity diagrams according to their key words (such as terms, verbs and roles), synthesize a set of activity diagrams in a sub-cluster, and then synthesize the resulting diagrams of multiple sub-clusters. The other could be to add combination rules to generate high-quality

initial population. Second, we also plan to explore the usability of the resulting diagram on group decision for requirements negotiation, requirements prioritization, and so on. Further, through diagram synthesis, we plan to analyze the contribution of participants in a group and explore how to stimulate participants to produce high-quality requirements.

References

[1] Tuunanen T, Rossi M. Engineering a method for wide audience requirements elicitation and integrating it to software development. In *Proc. the 37th Hawaii International Conference on System Sciences*, January 2014. DOI: 10.1109/HICSS.2004.1265420.

[2] Snijders R, Atilla Ö, Dalpiaz F, Brinkkemper S. Crowd-centric requirements engineering: A method based on crowdsourcing and gamification. Technical Report, Utrecht University, 2015. http://www.cs.uu.nl/research/techreps/repo/CS-2015/2015-004.pdf, June 2020.

[3] Groen E C, Seyff N, Ali R *et al.* The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 2017, 34(2): 42-52. DOI: 10.1109/MS.2017.33.

[4] Adepetu A, Ahmed K A, Abd Y A, Zabbi A A, Svetinovic D. CrowdREquire: A requirements engineering crowdsourcing platform. In *Proc. the 2012 AAAI Spring Symposium on Wisdom of the Crowd*, March 2012, Article No. 1.

[5] Hosseini M, Shahri A, Phalp K, Taylor J, Ali R, Dalpiaz F. Configuring crowdsourcing for requirements elicitation. In *Proc. the 9th IEEE International Conference on Research Challenges in Information Science*, May 2015, pp.133-138. DOI: 10.1109/RCIS.2015.7128873.

[6] Pratyoush K S, Richa S. Crowdsourcing to elicit requirements for MyERP application. In *Proc. the 1st IEEE International Workshop on Crowd-Based Requirements Engineering*, August 2015, pp.31-35. DOI: 10.1109/CrowdRE.2015.7367586.

[7] Murukannaiah P K, Ajmeri N, Singh M P. Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in crowd RE. In *Proc. the 24th IEEE International Requirements Engineering Conference*, September 2016, pp.176-185. DOI: 10.1109/RE.2016.68.

[8] Groen E, Dörr J, Adam S. Towards crowd-based requirements engineering: A research preview. In *Proc. the 21st International Working Conference on Requirements Engineering: Foundation for Software Quality*, March 2015, pp.247-253. DOI: 10.1007/978-3-319-16101-3_16.

[9] Murukannaiah P K, Ajmeri N, Singh M P. Toward automating crowd RE. In *Proc. the 25th International Requirements Engineering Conference*, September 2017, pp.512-515. DOI: 10.1109/RE.2017.74.

[10] Rosa L M, Dumas M, Uba R, Dijkman R. Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering and Methodology*, 2013, 22(2): Article No. 11. DOI: 10.1145/2430545.2430547.

[11] Dalpiaz F, Brinkkemper S. Agile requirements engineering with user stories. In *Proc. the 26th International Requirements Engineering Conference*, August 2018, pp.506-507. DOI: 10.1109/RE.2018.00075.

[12] Sutcliffe A, Maiden N, Minocha S, Manuel D. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 1998, 24(12): 1072-1088. DOI: 10.1109/32.738340.

[13] Song X P, Hwong B, Matos G, Rudorfer A, Nelson C, Han M, Girenkov A. Understanding requirements for computer aided healthcare workflows: Experiences and challenges. In *Proc. the 28th International Conference on Software Engineering*, May 2006, pp.930-934. DOI: 10.1145/1134285.1134455.

[14] Ahmad T, Iqbal J, Ashraf A, Truscan D, Porres I. Model-based testing using UML activity diagrams: A systematic mapping study. *Computer Science Review*, 2019, 33: 98-112. DOI: 10.1016/j.cosrev.2019.07.001.

[15] Nejati S, Sabetzadeh M, Chechik M, Easterbrook S M, Zave P. Matching and merging of statecharts specifications. In *Proc. the 29th International Conference on Software Engineering*, May 2007, pp.54-64. DOI: 10.1109/ICSE.2007.50.

[16] Sabetzadeh M. Merging and consistency checking of distributed models [Ph.D. Thesis]. Department of Computer Science, The University of Toronto, 2008.

[17] Rubin J, Chechik M. N-way model merging. In *Proc. the 9th Joint Meeting on Foundations of Software Engineering*, August 2013, pp.301-311. DOI: 10.1145/2491411.2491446.

[18] Harman M, Mansouri S A, Zhang Y Y. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 2012, 45(1): Article No. 11. DOI: 10.1145/2379776.2379787.

[19] Alshahwan N, Harman M. Automated web application testing using search based software engineering. In *Proc. the 26th IEEE/ACM International Conference on Automated Software Engineering*, November 2011, pp.3-12. DOI: 10.1109/ASE.2011.6100082.

[20] Dai Y S, Xie M, Poh K, Yang B. Optimal testing-resource allocation with genetic algorithm for modular software systems. *Journal of Systems and Software*, 2003, 66(1): 47-55. DOI: 10.1016/S0164-1212(02)00062-6.

[21] Wang C H, Zhang W, Zhao H Y, Jin Z. Eliciting activity requirements from crowd using genetic algorithm. In *Proc. the 4th Asia Pacific Requirements Engineering Conference*, November 2017, pp.99-113. DOI: 10.1007/978-981-10-7796-8_8.

[22] Shannon C E. A mathematical theory of communication. *Bell System Technical Journal*, 1948, 27(3): 379-423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[23] Fernando S, Stevenson M. A semantic similarity approach to paraphrase detection. In *Proc. the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, December 2008, pp.45-52.

[24] Kondrak G. N-gram similarity and distance. In *Proc. the 12th International Conference on String Processing and Information Retrieval*, November 2005, pp.115-126. DOI: 10.1007/11575832_13.

[25] Niwattanakul S, Singthongchai J, Naenudorn E, Wanapu S. Using of Jaccard Coefficient for keywords similarity. In *Proc. the 2013 International MultiConference of Engineers and Computer Scientists*, March 2013.

[26] Simon D. Evolutionary Optimization Algorithms. John Wiley & Sons., 2013.

[27] Saraph V, Milenkovic T. MAGNA: Maximizing accuracy in global network alignment. *Bioinformatics*, 2014, 30(20): 2931-2940. DOI: 10.1093/bioinformatics/btu409.

[28] Arora C, Sabetzadeh M, Briand L C, Zimmer F. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 2017, 43(10): 918-945. DOI: 10.1109/TSE.2016.2635134.

[29] Lim S L, Finkelstein A. StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE Transactions on Software Engineering*, 2012, 38(3): 707-735. DOI: 10.1109/TSE.2011.36.

[30] Breaux T, Schaub F. Scaling requirements extraction to the crowd: Experiments with privacy policies. In *Proc. the 22nd International Requirements Engineering Conference*, August 2014, pp.163-172. DOI: 10.1109/RE.2014.6912258.

[31] Maalej W, Happel H, Rashid A. When users become collaborators: Towards continuous and context-aware user input. In *Proc. the 24th Annual ACM SIGPLAN Conference Companion on Object-Oriented Programming, Systems, Languages, and Applications*, October 2009, pp.981-990. DOI: 10.1145/1639950.1640068.

[32] Kessentini M, Werda W, Langer P, Wimmer M. Search-based model merging. In *Proc. the 15th Annual Conference on Genetic and Evolutionary Computation*, July 2013, pp.1453-1460. DOI: 10.1145/2463372.2463553.

[33] Assunção W K G, Vergilio S R, Lopez-Herrejon R E. Discovering software architectures with search-based merge of UML model variants. In *Proc. the 16th International Conference on Software Reuse*, May 2017, pp.95-111. DOI: 10.1007/978-3-319-56856-0_7.

[34] Debreceni C, Ráth I, Varró D, De Carlos X, Mendialdua X, Trujillo S. Automated model merge by design space exploration. In *Proc. the 19th International Conference on Fundamental Approaches to Software Engineering*, April 2016, pp.104-121. DOI: 10.1007/978-3-662-49665-7_7.

Chun-Hui Wang is a Ph.D. candidate of the School of Electronics Engineering and Computer Science, Peking University, Beijing. She is also an associate professor at the School of Computer Science, Inner Mongolia Normal University, Hohhot. Her research interests include requirements engineering and collective intelligence based software engineering. She is a member of CCF. She got the Best Paper Award of APREs 2017.

Zhi Jin received her Ph.D. degree in computer science from Changsha Institute of Technology, Changsha, in 1992. She is currently a professor of computer science at Peking University, Beijing, and the deputy director of Key Laboratory of High Confidence Software Technologies (Ministry of Education) at Peking University, Beijing. Her research interests include software engineering, requirements engineering, knowledge engineering, and machine learning. She has (co-)authored five books and has published more than 150 referred conference/journal papers. She is/was principle investigator of more than 15 national competitive grants, including the chief scientist of a national basic research project (973 project) of the Ministry of Science and Technology of China. She is currently a fellow of CCF, a senior member of IEEE, and the director of CCF Technical Committee of System Software.

Wei Zhang received his Ph.D. degree in computer Science from Peking University, Beijing, in 2006. Currently, he is an associate professor at School of Electronics Engineering and Computer Science, Peking University, Beijing. His research interests include software reuse, requirements engineering, collective intelligence based software engineering, and collective intelligence system design.

Didar Zowghi is professor of software engineering and the deputy dean of Graduate Research School at University of Technology Sydney (UTS), Australia, with B.Sc. (Hons), M.Sc. and Ph.D. degrees. Her research is focused on improving the software development process and the quality of its products. Her expertise is in requirements engineering and evidence-based research and has conducted a variety of empirical studies in many areas of software engineering. She is an associate editor of IEEE Software and Requirements Engineering Journal. She has published over 190 research articles in prestigious conferences and journals co-authored with 90 different researchers from 30 countries.

Hai-Yan Zhao received her Ph.D. degree in information engineering from the University of Tokyo, Tokyo, in 2003. She is now an associate professor of software engineering at Peking University, Beijing. Her research interests include requirements engineering, model-driven software engineering, and programming languages and systems. She is a senior member of CCF and a member of both ACM and IEEE.

Wen-Pin Jiao is a full professor with School of Electronics Engineering and Computer Science, Peking University, Beijing. He received his Ph.D. degree in computer science from Institute of Software, Chinese Academy of Sciences, Beijing, in 2000, and his M.S. and B.S. degrees in computer science from East China University of Science and Technology, Shanghai, in 1997 and 1991, respectively. Jiao's research interests include software engineering, multi-agent systems, and intelligent software technology.