Translation of the Mutation Operator from Genetic Algorithms to Evolutionary Ontologies

Diana Contraş
Dept. of Automation
Technical University of Cluj-Napoca
Romania

Oliviu Matei
Dept. of Electrical Engineering
Technical University of Cluj-Napoca
North University Centre of Baia Mare, Romania

***Abstract*—Recently introduced, evolutionary ontologies represent a new concept as a combination of genetic algorithms and ontologies. We have defined a new framework comprising a set of parameters required for any evolutionary algorithm, i.e. ontological space, representation of individuals, the main genetic operators such as selection, crossover, and mutation. Although a secondary operator, mutation proves its importance in creating and maintaining evolutionary ontologies diversity. Therefore, in this article, we widely debate the mutation topic in evolutionary ontologies, marking its usefulness in practice by experimental results. Also we introduce a new mutation operator, called relational mutation, concerning mutation of a relationship through its inverse.**

***Keywords*—Evolutionary ontologies; Genetic algorithms; Mutation; Ontology**

I. Introduction

Evolutionary ontologies (EO) reprezent a new concept, introduced very recently by Matei et al. in [1]. Vast and complex information used in artificial intelligence (AI) increasingly requires the use of ontologies as a manifestation of knowledge. But static information is of no use in the areas of AI, always subjected to change. Therefore evolutionary ontologies are the next natural step that is required to be made in AI.

EO are genetic algorithms using ontologies as individual items instead of classical data structures such as strings of bits, different values (real numbers, characters, objects) or programs.

Also in EO is followed the pattern of the evolutionary process: it is selected an initial population that undergoes genetic operators such as crossover and mutation, then the offsping are subjected again to selection for resuming the algorithm until the condition of the problem to be solved is fulfilled or the maximum number of epochs is reached.

Although originated in classical genetic operators, crossover and mutation that are used in evolutionary ontologyies are new operators designed to meet the needs of complex structuring of knowledge contained in ontologies.

The genetic operators of evolutionary ontologies were introduced in [1], but was only traced the outline. The involvement of mutation in evolutionary ontologies diversity requires the detailing of the operator, which is the purpose of this article.

The rest of the paper is organized as follows: section II contains references to the genetic mutation operator and to the ontologies, but we have to emphasize that the two concepts are first used together in this article. Section III introduces mutation operators for evolutionary ontologies, the absolute novelty being relational mutation. In section IV we demonstrate the utility of mutation operators in practice, on an application for automatically generated scenes. Finally, the conclusions are marked in section V.

II. Related Work

Imitating the biological model, genetic mutation operator changes the offspring resulting from crossover of the chromosomes [2]. Often, genetic algorithms tend to block during searching the solution at a local optimum. The purpose of the mutation is to prevent this by not allowing the population to become too similar [3].

Mutation operator does not apply to all chromosomes, but a certain percentage of the population, called mutation rate [4]. Sahoo et al. [5] state that mutation rate generally ranges between 0.05 and 0.02, but that sometimes is calculated as $1/n$, where *n* is the number of chromosomal genes. Mutation rate is an important parameter of genetic algorithms, because, as Cao et al. show in [6] and Maaita et al. in [7], too many or too few mutations adversely affect convergence.

Mutation operator depends on the chromosome coding technique [8]. For the binary representation is used, in particular, *bit-flipping* mutation [9], which consists in changing the values of some genes from 0 to 1 or vice versa [10].

For the integer encoding is used the *random resetting* mutation operator, where the value of a gene is replaced by a value chosen at random from an allowed range [11] or *creep* mutation, where a gene value is increased or decreased by a small value [12].

Permutation encoding involves chromosome representation as a string of integer or real values [13]. The most suitable mutation operators for this type of encoding are *swap* mutation and *inversion* mutation [14]. Under the swap mutation, two randomly chosen genes interchange their values [15]. Regarding inversion mutation, two random points are selected within the chromosome and all the genes between the two points are reversed [15].

Uniform and *non-uniform* mutation are two types of mutations used for real encoding of the data. The positions of the genes uniform mutated are chosen with equal probability [16], and the values of these genes are replaced with uniform random numbers [17]. Introduced by Michalewicz and presented in [18], non-uniform mutation, is to explore solutions space uniformly at first, and finally to become local search.

If we discuss coding tree, we meet *subtree mutation*, where a randomly selected subtree is replaced by another randomly created subtree (the process is broadly described in [19]) or *point* mutation, which changes a randomly selected node with a another node [20].

In addition to ordinary mutation operators, various types of mutation have been researched, such as discrete [21], [22], model-based [23] and fuzzy [24].

Although, so far, the mutation was considered a secondary genetic operator, Lehre and Yao proved in [25] that there is a close link between mutation and selection, which influences the running time of genetic algorithms. Moreover, mutation is such an important genetic operator, that significant research has been done for implementing it in different programming languages, such as C++ [26]. Also, mutation was used with particle swarm optimization technique [27], whose role is to optimize the difficult problems of mathematics in continuous or binary space [28]. Lastly, the importance of mutation operator is emphasized in practical applications of genetic algorithms, like the ones regarding the alignment of multiple molecular sequences [29], the detecting of human faces in an image with complex background [30] and the modeling of neural image compression [31].

The multitude of information requires the use of new data structures, enabling knowledge sharing and reuse, therefore, in recent years there has been an increase in the use of ontologies [32]. An ontology can be seen as a set of related concepts [33].

Matei et al. used, for the first time, the ontologies in the context of genetic algorithms, introducing the concept of evolutionary ontologies in [1]. Our goal was to enrich the ontology knowledge already held, this being achieved by applying genetic operators on ontologies. Due to the complex nature of mutation operator, we believe that the general framework introduced in [1] is not enough, therefore we detail mutation in this article.

III. Mutation in evolutionary ontologies

In [1] was defined the operating framework of ontologies under the aegis of the evolutionary algorithms. The ontological space (onto-space) is an ontology that includes all the elements of a domain of knowledge at information level with all their characteristics and relationships. Formally, an onto-space is defined as follows:

$$OS = (C, P, I) \tag{1}$$

where C is the set of classes, P is the set of properties and I is the set of instances.

An individual of the evolutionary ontologies is a subset of the ontology, meaning

$$Ch = (SC, SP, SI) \tag{2}$$

where $SC \subset C$ is a subset of classes in OS, $SP \subset P$ is a subset of properties in OS and $SI \subset I$ is a subset of instances in OS.

A population consists of (μ) such individuals, but not necessarily cover the whole onto-space, therefore

$$\bigcup_{i=1}^{\mu} Ch_i \subset OS \tag{3}$$

Due to the complex nature of individuals, is required the instantiation mutation, resulting three types of mutation operators, namely class mutation, instance mutation and property mutation.

A. Class mutation

The set of classes of onto-space, C, can be seen as a reunion of all ontology classes

$$C = \bigcup_{i=1}^{n_c} C_i \tag{4}$$

where n_c is the number of classes in the ontology.

In an ontology, therefore also in onto-space, classes are organized hierarchically [34], a class whether or not containing one or more subclasses.

Any class or subclass C_i may have one or more subclasses, which in turn can have subclasses, going as deep as much require the domain of knowledge that is represented.

$$C_i = \bigcup_{j=1}^{n_{c_i}} C_{i.j} \tag{5}$$

where n_{c_i} is the number of subclasses of class C_i.

Any class or subclass C_i can be regarded as a set of instances, composed by its own instances and all instances of all its subclasses.

$$C_i = \bigcup_{k_1=1}^{n_{inst_{c_i}}} I_{i.k_1} \bigcup_{k_2=1}^{n_{c_i}} \bigcup_{k_3=1}^{n_{inst_{k_2}}} I_{i.k_2.k_3} \tag{6}$$

where $n_{inst_{c_i}}$ is the number of the own instances of class C_i.

Mutation of a class C_i takes place by replacing all its own instances with the instances of a randomly chosen subclass of class C_i. In other words, the instances $\bigcup_{k_1=1}^{n_{inst_{c_i}}} I_{i.k_1}$ will be replaced by the instances $\bigcup_{k_3=1}^{n_{inst_{q_1}}} I_{i.q_1.k_3}$, where q_1 is a random number with values in the range $[1, n_{c_i}]$ representing the serial number of the selected subclass.

B. Instance mutation

The set of instances of onto-space, I, is the reunion of all ontology instances

$$I = \bigcup_{i=1}^{n_{inst}} I_i \tag{7}$$

where n_{inst} is the number of instances in the ontology.

The instances are grouped together in classes, as can be seen in (6). We consider an instance $I_{i.q_1}$, where $q_1 \in$

$[1, n_{inst_{c_i}}]$, of a class C_i. Mutation of the instance $I_{i.q_1}$ means replacing it with another instance of class C_i, namely $I_{i.q_2}$, where $q_2 \in [1, n_{inst_{c_i}}]$, $q_2 \neq q_1$ and $I_{i.q_2} \notin \bigcup_{k_2=1}^{n_{c_i}} \bigcup_{k_3=1}^{n_{inst_{k_2}}} I_{i.k_2.k_3}$.

C. Data Property Mutation

In an ontology, thus also in onto-space, are two types of properties: data properties and object properties [35]. We note with P_{d_i} a data property, $\forall i \in [1, n_{p_d}]$, where n_{p_d} is the number of data properties in OS and with P_{o_j} an object property, $\forall j \in [1, n_{p_o}]$, where n_{p_o} is the number of object properties in OS. Therefore, the set of onto-space properties is defined as follows:

$$P = \bigcup_{i=1}^{n_{p_d}} P_{d_i} \cup \bigcup_{j=1}^{n_{p_o}} P_{o_j} \tag{8}$$

The mutation may be applied differently, depending on the type of property.

A data property is characterized by a data type, that indicates the kind of value for the corresponding instance, which have such property data. Typically, the data types most commonly used for data properties are numerical types and boolean.

We consider a data property P_{d_i}, with $i \in [1, n_{p_d}]$. If I_j, with $j \in [1, n_{inst_i}]$ is one of the instances that have this data property, and the value corresponding to I_j for property P_{d_i} is val, with $val \in [val_1, val_k]$, then applying mutation on data property means changing the val value. Changing the val value depends on the data type of the property P_{d_i}, as follows:

1) If data property P_{d_i} has an integer data type, then there are two possibilities:
   a) the val value will be changed with val', another value from the range $[val_1, val_k]$, ie
   $$val = val', \text{ where } val' \in [val_1, val_k] \tag{9}$$
   b) the val value will be changed by adding a value (positive or negative), which is part of a symmetrical distribution to zero, ie
   $$val = val + \alpha, \text{ where } \alpha > 0 \text{ or } \alpha < 0 \text{ such that } val + \alpha \in [val_1, val_k] \tag{10}$$
2) If data property P_{d_i} has a real data type, then there are two possibilities:
   a) first possibility is similar to the first possibility for integer values, hence (9) takes place also in this case
   b) the val value will be changed by adding a value (positive or negative), based on non-uniform mutation, ie
   $$val = val + \alpha \cdot \mid \beta - val \mid \cdot r \cdot (1 - \frac{g}{G})^b$$
   where
   r is a uniformly distributed random number [0, 1],
   g is the current generation,
   G is the maximum number of generations,
   b is the degree of non-uniformity usually of value 5, moreover
   when $\alpha > 0$, $\beta = val_k$
   and when $\alpha < 0$, $\beta = val_1$ (11)
3) If data property P_{d_i} has a boolean data type, then mutation acts as a switch by changing the value val with its complement, ie
   $$val = val', \text{ where } val' = \text{FALSE if } val = \text{TRUE or } val' = \text{TRUE if } val = \text{FALSE} \tag{12}$$

D. Object Property Mutation

An object property is designed to establish relations between the classes of the onto-space and thus between their instances. Each object property has behavior of a binary relation [36], so each object property is defined on a domain and has values in a range.

We consider an object property P_{o_j}, with $j \in [1, n_{p_o}]$, which has the domain, respectively the range one class or a class reunion.

$$(P_{o_j}, \bigcup_{k_1=1}^{n_{dom_j}} C_{k_1}, \bigcup_{k_2=1}^{n_{r_j}} C_{k_2}) \tag{13}$$

where n_{dom_j} is the number of domain classes and n_{r_j} is the number of range classes for object property P_{o_j} .

According to (6), an instance of the domain is $I_{i_1.k_1}$, where $k_1 \in [1, n_{inst_{c_{i_1}}}]$ or $I_{i_1.k_2.k_3}$, where $k_2 \in [1, n_{c_{i_1}}]$ and $k_3 \in [1, n_{inst_{k_2}}]$ and an instance of the range is $I_{i_2.k_1}$, where $k_1 \in [1, n_{inst_{c_{i_2}}}]$ or $I_{i_2.k_2.k_3}$, where $k_2 \in [1, n_{c_{i_2}}]$ and $k_3 \in [1, n_{inst_{k_2}}]$.

For the sake of simplicity, we note, the P_{o_j} domain with D_j, the P_{o_j} range with R_j, an instance of the domain with I_{i_1}, an instance of the range with I_{i_2}, the results being the same regardless of chosen notations. Thus

$$I_{i_1} P_{o_j} I_{i_2} \tag{14}$$

where $I_{i_1} \in D_j$ and $I_{i_2} \in R_j$.

The inverse of the P_{o_j}, if any, is $P_{o_j}^{-}$ with the domain R_j and the range D_j, therefore

$$I_{i_2} P_{o_j}^{-} I_{i_1} \tag{15}$$

Applying mutation operator on the object property P_{o_j} means one of the following:

1) Replacing the object property P_{o_j} with its inverse $P_{o_j}^{-}$, if it exists, very suitable for spatial relations,

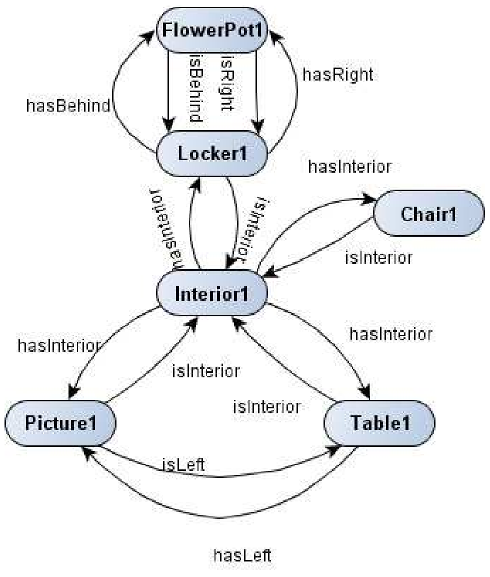

Fig. 1: Scene before relational mutation

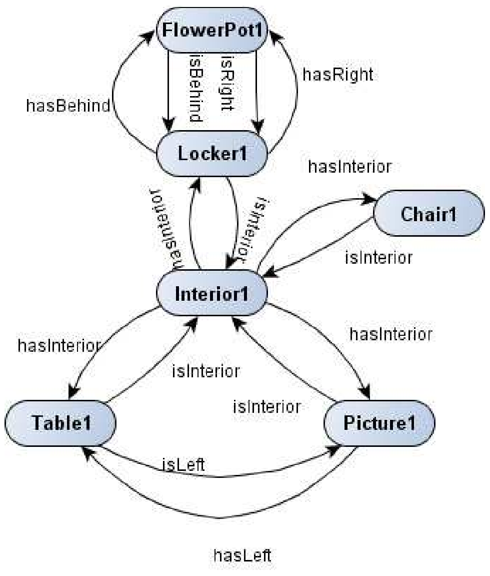

Fig. 2: Scene after relational mutation

getting a mirror effect, as you can see in Fig. 1 and Fig. 2.
We call this type of mutation, **relational** mutation, but it does not apply in all conditions. The relations (14) and (15) occur before mutation. After mutation:

$$I_{i_1} P_{o_j}^{-} I_{i_2}$$
$$I_{i_2} P_{o_j} I_{i_1} \qquad (16)$$

with $I_{i_1} \in R_j$ and $I_{i_2} \in D_j$. So I_{i_1} and I_{i_2} belong both to the domain and range, when applying relational mutation. For relational mutation to be valid, must occur:

$D_j = R_j$, in which case relations P_{o_j} and $P_{o_j}^{-}$swap

or

$D_j \sqcap R_j \neq \perp$, in which case the relational mutation applies only when relations link instances of intersection. (17)

Of course that relational mutation may also occur in other particular cases, which must be checked manually to match the onto-space rules, but those shown in (17) may be automatically applied as imposed by evolutionary ontologies.

2) Changing the object property P_{o_j} with another object property P_o, which is not part of the onto-space, but suitable for domain knowledge, so as to have the same domain and range as the object property P_{o_j}.

$$P_{o_j} = P_o, \text{ so that } P_o \notin OS, \text{ but}$$
$$(P_o, \bigcup_{k_1=1}^{n_{dom_j}} C_{k_1}, \bigcup_{k_2=1}^{n_{co_j}} C_{k_2}) \qquad (18)$$

If the object property P_{o_j} has inverse, we identify two situations:

a) The new object property P_o has inverse, in which case the inverse of object property P_o will substitute the inverse of object property P_{o_j};

b) The new object property P_o has no inverse, in which case the inverse of object property P_{o_j} will be deleted from the onto-space.

3) Replacing one of the instances I_{i_1} or I_{i_2} with another instance I'_{i_1}, respectively I'_{i_2}, belonging to the domain, respectively to the range.

starting from $I_{i_1} P_{o_j} I_{i_2}$, is obtained by the application of mutation

$$I'_{i_1} P_{o_j} I_{i_2}, \text{ where } I'_{i_1} \in \bigcup_{k_1=1}^{n_{dom_j}} C_{k_1} \text{ or}$$
$$I_{i_1} P_{o_j} I'_{i_2} \text{ where } I'_{i_2} \in \bigcup_{k_2=1}^{n_{co_j}} C_{k_2} \qquad (19)$$

The third situation may seem similar to instance mutation. However, is another type of mutation because in the case of instance mutation, an instance I_i is replaced by another instance of the same class as I_i, while in the case of object property mutation, an instance is replaced by another instance of a class reunion, which is the domain or the range of the object property.

IV. Experimental results

The concept of *evolutionary ontologies* was applied on user centered automatically generated scenes application, like in all previous similar experiments, reported in [37], [38]. This way, the methodology is consistent throughout all the experiments. Cycle of the algorithm takes place in the following stages: it sets the initial population as a list of 10 randomly generated scenes. For each scene it selects a frame. We have considered

TABLE I: The number of differences between epochs without, respectively with mutation

Epoch	Without mutation			With mutation			Improvements		
	Median	Min.	Max.	Median	Min.	Max.	Median	Min.	Max.
5	11	9	13	13	10	16	18,18%	11,11%	23,08%
10	26	22	28	30	25	33	15,38%	13,64%	17,86%
15	34	30	38	39	34	45	14,71%	13,33%	18,42%
20	38	32	40	41	35	45	7,89%	9,38%	12,50%
25	36	32	38	39	35	43	8,33%	9,38%	13,16%
30	25	23	27	29	27	30	16,00%	17,39%	11,11%
35	26	24	27	30	29	31	15,38%	20,83%	14,81%

TABLE II: The number of differences between epochs when using a simple mutation, respectively applying also the inverse operator

Epoch	Without inverse			With inverse			Improvements		
	Median	Min.	Max.	Median	Min.	Max.	Median	Min.	Max.
5	13	10	16	14	10	16	7,69%	0,00%	0,00%
10	30	25	33	30	27	33	0,00%	8,00%	0,00%
15	39	34	45	40	34	45	2,56%	0,00%	0,00%
20	41	35	45	42	37	45	2,44%	5,71%	0,00%
25	39	35	43	39	36	44	0,00%	2,86%	2,33%
30	29	27	30	30	30	31	3,45%	11,11%	3,33%
35	30	29	31	31	31	32	3,33%	6,90%	3,23%

three types of frames sufficient for the power of example: land, water and interior. Every user (the algorithm is executed by 10 different users) gives each scene a grade of 1-10, according to his/her preferences. Then it selects scenes involved in the evolutionary process, by Monte Carlo technique, where the grades of users are the fitness function values. The selected scenes are subjected to crossover and mutation operators, whereupon the algorithm repeats until at least one scene is marked with 10 or the maximum threshold of 35 epochs is reached.

The first set of experiments were designed having in mind the importance of the mutation operator in the economy of evolutionary ontologies. The results are summarized in Table I. Two successive lines of the table, in column *Epoch*, denote the epochs range for which we have recorded values.

From Table I it is noticeable that the mutation improves the diversification with 7% to 24%, which is significant comparing with the complexity of the individuals (which is the number of components of a scene and the relationships between them).

A next set of experiments targeted the influence of the completely new type of mutation, namely *relational mutation*. The results are summarized in Table II. The header of Table II has the same meaning as the header of Table I.

The inversion itself brings interesting diversification, counting for up to almost 8%, which is significant for a simple operator, taking into account that it can be applied only for invertible relations. A graphical representation of the results reported in Table II is depicted in figure 3.

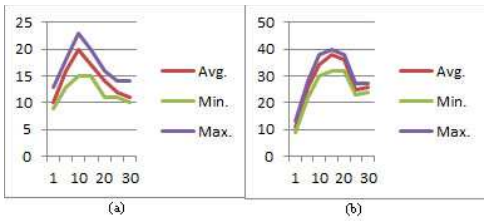

Fig. 3: The median, minimum and maximum number of differences in the two cases: (a) without inverse; (b) with inverse as mutation

V. Conclusion

This article puts a good mathematical foundation to the mutation operator in the context of evolutionary ontologies. Actually, we define this operator at the level of:

- classes;
- instances;
- data properties;
- object properties (class relationships).

The most complex mutation operators are at the level of object properties, where we define three specific operators:

- replacing the whole relation with its inverse when this is possible;
- replacing the property itself;
- changing relations between the instances.

The complete novelty of the article is the application of the inverse relationships as relational mutation in the context of evolutionary ontologies. This type of mutation has proved highly efficient in application for automated generation of scenes, causing the innovations in scenes due to its mirror effect.

As future work we intend to use relational mutation in an application for automatic design of industrial products, where its role is to maintain the diversity of relationships between components of products.

Acknowledgment

The research leading to these results has received funding from the European Communitys Seventh Framework Programme under grant agreement No609143 Project ProSEco.

References

[1] O. Matei, D. Contraş, and P. Pop, "Applying evolutionary computation for evolving ontologies," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 1520–1527.

[2] A. K. M. Masum, M. Shahjalal, F. Faruque, and I. Sarker, "Solving the vehicle routing problem using genetic algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 7, pp. 126–131, 2011.

[3] N. H. Moin, S. Salhi, and N. Aziz, "An efficient hybrid genetic algorithm for the multi-product multi-period inventory routing problem," *International Journal of Production Economics*, vol. 133, no. 1, pp. 334–343, 2011.

[4] D. B. Conkey, A. N. Brown, A. M. Caravaca-Aguirre, and R. Piestun, "Genetic algorithm optimization for focusing through turbid media in noisy environments," *Optics express*, vol. 20, no. 5, pp. 4840–4849, 2012.

[5] L. Sahoo, A. K. Bhunia, and P. K. Kapur, "Genetic algorithm based multi-objective reliability optimization in interval environment," *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 152–160, 2012.

[6] K. Cao, B. Huang, S. Wang, and H. Lin, "Sustainable land use optimization using boundary-based fast genetic algorithm," *Computers, Environment and Urban Systems*, vol. 36, no. 3, pp. 257–269, 2012.

[7] A. A. Maaita, J. Zraqou, F. Hamad, and H. A. Al-Sewadi, "A generic adaptive multi-gene-set genetic algorithm (amga)," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 5, pp. 12–18, 2015.

[8] K. El-Naggar and T. H. Abdelhamid, "Selective harmonic elimination of new family of multilevel inverters using genetic algorithms," *Energy Conversion and Management*, vol. 49, no. 1, pp. 89–95, 2008.

[9] W. Jia, D. Zhao, T. Shen, C. Su, C. Hu, and Y. Zhao, "A new optimized ga-rbf neural network algorithm," *Computational intelligence and neuroscience*, vol. 2014, 2014.

[10] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

[11] F. Corut Ergin, E. Kaldırım, A. Yayımlı, and A. Ş. Uyar, "Ensuring resilience in optical wdm networks with nature-inspired heuristics," *Journal of Optical Communications and Networking*, vol. 2, no. 8, pp. 642–652, 2010.

[12] E. N. Chatzi, B. Hiriyur, H. Waisman, and A. W. Smyth, "Experimental application and enhancement of the xfem–ga algorithm for the detection of flaws in structures," *Computers & Structures*, vol. 89, no. 7, pp. 556–570, 2011.

[13] M. K. Rafsanjani and S. Eskandari, "The effect of a new generation based sequential selection operator on the performance of genetic algorithm," *Indian Journal of Science and Technology*, vol. 5, no. 12, pp. 3758–3761, 2012.

[14] S. Molla-Alizadeh-Zavardehi, A. Mahmoodirad, and M. Rahimian, "Step fixed charge transportation problems via genetic algorithm," *Indian Journal of Science and Technology*, vol. 7, no. 7, pp. 949–954, 2014.

[15] B. Ferriman and C. Obimbo, "Solving for the rc4 stream cipher state register using a genetic algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 5, pp. 216–223, 2014.

[16] J. N. Zadeh, B. R. Wolfe, and N. A. Pierce, "Nucleic acid sequence design via efficient ensemble defect optimization," *Journal of computational chemistry*, vol. 32, no. 3, pp. 439–452, 2011.

[17] V. Meruane and W. Heylen, "An hybrid real genetic algorithm to detect structural damage using modal properties," *Mechanical Systems and Signal Processing*, vol. 25, no. 5, pp. 1559–1573, 2011.

[18] Z. Michalewicz and C. Z. Janikow, "Handling constraints in genetic algorithms." in *ICGA*, 1991, pp. 151–157.

[19] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[20] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," *Evolutionary Computation*, vol. 6, no. 3, pp. 231–252, 1998.

[21] Q. Fan and X. Yan, "Self-adaptive differential evolution algorithm with discrete mutation control parameters," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1551–1572, 2015.

[22] X. Zhang and S. Y. Yuen, "A directional mutation operator for differential evolution algorithms," *Applied Soft Computing*, vol. 30, pp. 529–548, 2015.

[23] B. K. Aichernig, E. Jöbstl, and S. Tiran, "Model-based mutation testing via symbolic refinement checking," *Science of Computer Programming*, vol. 97, pp. 383–404, 2015.

[24] M. Vannucci and V. Colla, "Fuzzy adaptation of crossover and mutation rates in genetic algorithms based on population performance," *Journal of Intelligent and Fuzzy Systems*, vol. 28, no. 4, pp. 1805–1818, 2015.

[25] P. K. Lehre and X. Yao, "On the impact of mutation-selection balance on the runtime of evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 2, pp. 225–241, 2012.

[26] P. Delgado-Pérez, I. Medina-Bulo, J. J. Domínguez-Jiménez, A. García-Domínguez, and F. Palomo-Lozano, "Class mutation operators for c++ object-oriented systems," *annals of telecommunications-annales des télécommunications*, pp. 1–12, 2014.

[27] Y. Jia and S. Chi, "Back-analysis of soil parameters of the malutang ii concrete face rockfill dam using parallel mutation particle swarm optimization," *Computers and Geotechnics*, vol. 65, pp. 87–96, 2015.

[28] J.-F. Chang, S.-C. Chu, J. F. Roddick, and J.-S. Pan, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science and Engineering*, vol. 21, no. 4, pp. 809–818, 2005.

[29] R. Gupta, P. Agarwal, and A. Soni, "Genetic algorithm based approach for obtaining alignment of multiple sequences," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 2, pp. 180–185, 2012.

[30] M. G. Moazzam, M. R. Parveen, and M. A.-A. Bhuiyan, "Human face detection under complex lighting conditions," *International Journal of Advanced Computer Science and Applications*, pp. 85–90, 1995.

[31] G. Rajput and M. Singh, "Modeling of neural image compression using ga and bp: a comparative approach," *International Journal of Advanced Computer Science and Applications*, pp. 26–34, 2011.

[32] K. Vanitha, K. Yasudha, K. Soujanya, M. S. Venkatesh, K. Ravindra, and S. V. Lakshmi, "The development process of the semantic web and web ontology," *IJACSA-International Journal of Advanced Computer Science and Applications*, vol. 2, no. 7, 2011.

[33] A. A. Felix, A. A. Taofiki, and S. Adetokunbo, "On algebraic spectrum of ontology evaluation," *IJACSA Editorial*, 2011.

[34] N. Manickasankari, D. Arivazhagan, and G. Vennila, "Ontology based semantic web technologies in e-learning environment using protégé," *Indian Journal of Science and Technology*, vol. 7, no. S6, pp. 64–67, 2014.

[35] S. Vigneshwari and M. Aramudhan, "Social information retrieval based on semantic annotation and hashing upon the multiple ontologies," *Indian Journal of Science and Technology*, vol. 8, no. 2, pp. 103–107, 2015.

[36] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[37] O. Matei, D. Contraş, and H. Vălean, "Relational crossover in evolutionary ontologies," in *10th International Conference on Soft Computing Models in Industrial and Environmental Applications*. Springer, 2015, pp. 165–175.

[38] O. Matei and D. Contras, "Advanced genetic operators in the context of evolutionary ontology," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 9–14.