Why Evolutionary Ontologies are not Genetic Programming

Diana Contraș
Dept. of Automation
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
diana.contras@profinfo.edu.ro

Oliviu Matei
Dept. of Electrical Engineering
TUC, North University Centre of Baia Mare
Baia Mare, Romania
oliviu.matei@cunbm.utcluj.ro

***Abstract*—Recently, the concept of evolutionary ontologies (EO) has been introduced. Ever since there are debates whether or not EO's are genetic algorithms or genetic programming (GP). This article makes a comparison between genetic programming (GP) and evolutionary ontologies (EO). Between the two there are significant differences, which make them completely distinct, although for some specific representations of the ontologic individuals, such as RDF, there are some similarities. However, we prove that there is a large gap between GP's and EO's, which impose EO's as a new, completely different field of evolutionary computation.**

Keywords—computational intelligence; evolutionary computation; genetic algorithms; genetic programming; optimization

I. INTRODUCTION

Evolutionary computation is a field of artificial intelligence which deals with evolutionary principles such as natural selection and genetic inheritance in order to solve continuous optimization and combinatorial optimization problems.

The major classical domains for evolutionary computations are: genetic algorithms (GA), evolutionary strategies (ES), genetic programming (GP).

Genetic programming (GP) introduced by Koza [25] is aimed at designing, in the evolutionary manner, calculation methods (such as programs, algorithms) or structures (such as circuits, decision tree). Their principles are based on genetic algorithms, but the individuals are programs that evolve, rather than numbers of binary strings. Koza implemented GP using LISP language [25], which is a functional programming language with a tree-like structure of functions. That is why the classical approaches to GP are tree-based. But, again, the GP is not about evolving trees, but programs.

In [32] Matei et al. have introduced the term of evolutionary ontologies (EO). They are evolutionary algorithms which manipulate ontologies as individuals. The use of ontologies is wide as there is knowledge available for many fields, such as biology [27], medicine [31], product development [29, 39, 40, 43], design [2, 9], geosciences [20, 30], management [6, 36], semantic web [14], even for traffic safety [7], financial decision [15] or more specific areas, such as the design of data summarization engine [50] and the architecture of the enterprise [21]. The evolutionary computation has the strength of exploring all those ontologies in a wise way.

Ever since, there is a debate whether or not evolutionary ontologies are in fact genetic programming. In this article we will show several reasons for which EO's are a different field of genetic computation than GP, although they share some commons aspects.

The rest of the paper is organized as follows. In section II an overview of genetic programming is presented, then, in section III we describe the evolutionary ontologies and, finally, in section IV, the major differences between the two concepts are detailed, while the conclusions are drawn in section V.

II. GENETIC PROGRAMMING

Genetic programming addresses one of the central goals of computer science, namely automated programming, whose goal is to create, in an automated way, a computer program that enables the computer to solve the problem. As Arthur Samuel stated in [45], the goal of automatic programming concerns: "How can computers be made to do what needs to be done, without being told exactly how to do it?"

A. GP individuals

Cramer [11] and Koza [25], suggested that tree structure should be used as the representation of an individual. Cramer published the first method like this and used subtree crossover. Other innovative implementations followed evolving in LISP and PROLOG ([12, 17]). Koza, however, was the first that recognized the importance of the method and demonstrated its feasibility for automatic programming in general.

Graphs are the newest arrival of the fundamental program structures [28, 49]. Teller and Veloso [48] proposed the name PADO for a graph-based GP system. Graphs are capable of representing very complex program structures compactly. A graph structure is no more than nodes connected by edges. One may think of an edge as a pointer between two nodes indicating the direction of the flow of program control. PADO does not just permit loops and recursion, but it embraces them. This is not a trivial point: other GP systems have experimented with loops and recursion only gingerly because of the great difficulties they cause.

B. GP selection

The most commonly used selection method in genetic algorithms is roulette wheel selection, but in GP, according to [25] there are notably alternative methods such as tournament selection and rank selection. According to [44], in GP tournament selection is preferred to other types of selections because the rank that is used to select the winner is ordinal which means that is not depending on the total fitness value of the population.

However, according to [52], there are two situations that should be considered in terms of tournament selection in GP. The first one is called multi-sampled, when the some individuals in population are chosen many times to form the tournament. The second one is called not-sampled, when some individuals are never chosen as part of the tournament. In [52] it is shown that different sampling replacement strategies have not important impact on selection pressure and using them cannot adjust the selection pressure in dynamic evolution. So the solution is to develop automatic and dynamic selection pressure tuning methods instead of alternative sampling replacement strategies.

C. GP crossover

The crossover operator combines the genetic material of the two parents by swapping a part of one parent with a part of the other. We will discuss tree, linear and graph crossover separately.

The tree-based crossover, proceeds by the following steps:

- Choose two individuals as parents, based on mating selection policy. Usually, this policy is similar to the one described for genetic algorithms. However, one can use any kind of selection.
- Select a random subtree in each parent. The selection of subtrees can be biased so that subtrees constituting terminals are selected with lower probability than other subtrees.
- Swap the selected subtrees between the two parents. The resulting individuals are the children.

The tree-based crossover swaps subtrees [37]; linear crossover (applicable for linear individuals) swaps segments of code between the parents [38].

The steps in linear crossover are:

- Choose two individuals as parents, based on mating selection policy. Usually, this policy is similar to the one described for genetic algorithms. However, one can use any kind of selection. This steps is the case as for tree-based crossover.
- Select a random sequence of instructions in each parent.
- Swap the selected sequence between the two parents. The resulting individuals are the children.

Graph crossover is somehow more complicated. The following procedure is employed by Teller [48]:

- Choose two individuals as parents, based on mating selection policy. Usually, this policy is similar to the one described for genetic algorithms. However, one can use any kind of selection. This steps is the case as for tree-based crossover.
- Divide each graph into two node sets: label all edges (pointers, arcs): as internal if they connect nodes within a fragment; as external otherwise and label nodes in each fragment as output if they are the source of an external edge and as input if they are the destination of an external edge.
- Swap the selected fragments between the parents.
- Recombine edges so that all external edges in the fragments now belonging together point to randomly selected input nodes of the other fragments.

With this method, all edges are assured to have connections in the new individual and valid graphs have been generated.

D. GP mutation

According to [42] in GP mutation has been studied since the early 80. However Koza [25] has not used mutation, wanting to prove that it is not necessary in GP. In [42] is stated that nowadays GP mutation is used mainly in modeling applications. Mutation operators are different depending on the method of representation – tree, linear or graph.

In case of tree representation are several types of mutation operators [42]:

- Subtree mutation: a subtree is selected randomly and is replaced with another randomly created subtree [25].
- Size-fair subtree mutation: was introduced by Langdon [26]. Using this mutation operator, the new subtree is, on average, the same size as the subtree to be replaced.
- Node replacement mutation: it is also known as point mutation. A node of the tree is selected randomly and is replaced with another node. In [33] is stated that in order to remain the tree legal the created node has to have the same number of parameters as the node that is replaced.
- Hoist mutation: introduced by Kinnear in [24] determine the random choice of a non-terminal node and converting the node and its subtree into the main tree.
- Shrink mutation: a subtree is selected randomly and is replaced with a randomly created terminal, according to [5]. This kind of operator is used in order to reduce program size.
- Permutation mutation: a function node is selected randomly then its arguments are randomly permuted [25].

For the tree individuals, the mutation operator randomly selects a node in the tree and replaces the select subtree with a

new randomly generated subtree. The new subtree is created in the same way as the programs in the initial population.

In linear GP, mutation is a little different [8]. The mutation operator selects an instruction from an individual. Then, it makes one or more changes in that instruction.

Suppose we have the instruction $r_0 + r_1 = r_2$ has been chosen for mutation. The following changes can occur: any of the operands can be changed to another randomly chosen operand or the operator can be replaced by another valid operator. Some mutation for the above instruction are $r_1 = r_1 + r_2$ or $r_0 = r_2 + r_2$ or $r_0 = r_1$ OR r_2.

In [41] Parallel Distributed Genetic Programming (PDGP), a new form of genetic programming, was introduced, for the programs with parallelism. In PDGP the programs are represented as graphs and the mutation operator was adapted to this representation, resulting two forms of mutation: global mutation and link mutation. The first type of mutation generates a random subgraph that is inserted into the parent graph. The second type of mutation, change a randomly chosen link of a randomly chosen node from the graph.

III. Evolutionary Ontologies

An ontological evolutionary algorithm is a genetic algorithm in which the individuals are ontologies rather than any other data structure. The solution space, the restriction and bounderies of this evolution is defined by means of an ontological space (called onto-space).

According to [32], the onto-space is an ontology describing a domain specific knowledge, containing all the concepts along with their allowed and denied relationships. The onto-space defines the degrees of freedom as well as the boundaries of the solution space to be searched by the evolutionary process. Quite often, the solution space is infinite and special algorithms are needed for exploring it efficiently. An onto-space would be the ontology about all electronic appliances and the solution required to be found is a possible arrangement of a kitchen given some restrictions.

Formally, an onto-space is $OS = (C, P, I)$, where C is the set of classes, P is the set of properties and I is the set of instances. Within the ontology OS, there are two disjunctive sub-ontologies $OS_e = (C_e, P_e, I_e)$ and $OS_f = (C_f, P_f, I_f)$, with $OS_e \cup OS_f = OS$. OS_e is the sub-ontology which will undergo the evolutionary process and OS_f is the fixed sub-ontology, e.g. which will not change under the evolutionary process.

A. EO individuals

An individual is a subset of the ontology, represented as $Ch = (SC, SP, SI)$ where SC is a subset of classes in OS, SP is a subset of properties in OS and SI is a subset of instances in OS. Furthermore, an individual of EO consists of an evolving part Ch_e, which will be subject to genetic operators, and a fixed part Ch_f. The two parts hold the following relations: $Ch_e \cup Ch_f = Ch$ and $Ch_e \cap Ch_f = \varnothing$. Moreover: $Ch_e \subset OS_e$, $Ch_f \subset OS_f$, $Ch \subset OS$.

A population consists of a given (μ) such individuals does not necessarily cover the entire onto-space, therefore

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

B. EO selection

Evolutionary ontologies are permissive, so selection operators can be folded easily on their technique. Roulette wheel selection, where the participants at the evolutionary process are chosen based on the fitness function (meaning the higher the fitness function is, the better the chance that a chromosome to be elected), is a good option for EO.

The deterministic selection, is also suitable for EO. In the case of (μ,λ)-selection, μ parents produce λ ($\lambda > \mu$) offspring and only the offspring undergo selection, while in the case of ($\mu+\lambda$)-selection both parents and offspring are involved in the evolutionary process.

In EO, both, Monte Carlo and deterministic techniques are used, depending on the specific problem to be solved. Thus, if the fitness function has subjective nature, it requires the use of Monte Carlo techniques, such as in the case of an application for automatic generation of scenes, where the grade that reflect user satisfaction is the fitness function. But if it is an application for automatic generation of products, where cost or time of production are objective parameters, deterministic technique must be used.

C. EO crossover

In the case of EO, we can talk about three distinctive crossover operators.

Class crossover: in an ontology classes are organized hierarchically. Two parents are randomly selected as two groups of related classes and subclasses, a cutting point is chosen, it changes between the two parent classes to the point of cutting and the two resulted groups are the offspring. In doing so is likely the ontology to become inconsistent. In such cases the repair operator will be used to validate the ontology.

Instance crossover: in an ontology for each class can be established as many individuals as wanted. The individuals are not independent, but related through object properties. Two such groups of related instances are selected as two parents. A cutting point is chosen and the instances that follow the cutting point are interchanged between parents, resulting two offspring. In case of inconsistencies is recommended the repair operator.

Relational crossover: in an ontology, there are two types of properties: the object properties and the data properties. Regarding object properties crossover, are elected two object properties P_1 and P_2 as parents, $P_1, P_2 \subset P$. Each property has a domain and a range from among the classes $C_{11}P_1C_{12}$ and $C_{21}P_2C_{22}$, with $C_{11}, C_{12}, C_{21}, C_{22} \subset C$ and it materializes in relating the instances: $I_{11}P_1I_{12}$ and $I_{21}P_2I_{22}$ where $I_{11} \in C_{11}$, $I_{12} \in C_{12}$, $I_{21} \in C_{21}$ and $I_{22} \in C_{22}$. After crossover are obtained two offspring, as follow $I_{21}P_1I_{12}$ and $I_{11}P_2I_{22}$ or $I_{11}P_1I_{22}$ and $I_{21}P_2I_{12}$.

As in the class crossover case, the result may be inappropriate. The repair operator will remove the inconsistency.

In an ontology each data property (DP) has a domain from among the classes and a range from different datatypes like int, integer, double, float etc. For data property crossover are elected two classes with several data properties as parents: $C_1(DP_{11}, DP_{12}, \ldots, DP_{1k}, \ldots, DP_{1n})$ and $C_2(DP_{21}, DP_{22}, \ldots, DP_{2k}, \ldots, DP_{2n})$. A cuting point is selected and the result of the crossover will be the same classes with modified properties: $C_1(DP_{21}, DP_{22}, \ldots, DP_{2k}, \ldots, DP_{1n})$ and $C_2(DP_{11}, DP_{12}, \ldots, DP_{1k}, \ldots, DP_{2n})$. The repair operator will be also applied if appropriate result is not obtained.

D. EO mutation

The mutation operator is applied for each individual with a probability pm and requires different treatment for classes, for instances, respectively for properties.

Instance mutation means replacing a randomly selected instance I_1 belonging to a class C_1 ($I_1 \in C_1$) with another instance from the same class C_1 (but not from the subclasses of the class C_1). This operator preserves the number of ontological instances in an individual.

A class mutation means replacing all the instances in a class C_1 by other instances belonging to a random subclass of the class C_1 in the onto-space.

The property mutations may be approached separately for data properties, respectively for object properties. Mutation of data properties for an instance, means the replacement value of the instance with another value of the same data type as the initial value. Mutation of an object property means to replace with its inverse, if it exists, or with another suitable object-property which is currently out the OS, or to replace one of the instances it relates to another instance of the same class or its subclasses as the initial instance.

E. Repair operator

The individuals of EO are complex data structures, not simple as binary strings. Due to their complexity, inconsistencies may arise after the evolutionary process. That is why Matei et al. [32] introduced a new operator, called repair, which is a deterministic operator. It can be applied on the population each time another genetic operator is used or only once, after crossover and mutation. Repairing an individual means adjusting its instances and properties so that they respect all the rules defined in the onto-space.

IV. Genetic Programming vs Evolutionary Ontologies

Evolutionary computation works with numbers rather than symbols, therefore is a sub-symbolic field of artificial intelligence [23]. From this point of view, we can affirm that evolutionary ontologies are symbolic simply because ontologies are symbolic intelligence [1, 22]. This is not the first attempt to create a bridge between the two domains: symbolic and subsymbolic. Thus, Goertzel presented in [19] a detailed design for incorporation of a subsymbolic system and a symbolic system into a integrative cognitive architecture, as a hybridization approach, and Andrews et al. offer a survey on the artificial neural networks in [4]. However, it is for the first time when Matei et al. [32] apply the genetic principles to ontologies. Evolutionary ontologies combine mathematical algorithms and ontologies, therefore they are not pure symbolic because they evolve using mathematical principles, which is never the case of other symbolic fields, such as knowledge-based systems [3, 18] and intelligent agents [47, 51].

All the elements of the evolutionary ontologies, meaning classes, instances, relations, properties participate at the evolutionary process. Thus we can conclude that the individuals of EO are ontologies themselves. In EO classes and instances receive improved properties and relations as the result of their participation in the evolutionary act. On the other hand, the individuals of GP are source codes or more simply programs. Evolving programs is a difficult job that does not generally lead to some spectacular results. Therefore in GP is used the representation of the programs as trees (e.g. in LISP). Thus the evolution in GP does not imply evolutionary trees, but evolving programs thereby the result is a program that contains new and/or modified code sequences representing the optimal solution.

Four types of crossover operator are found in EO: class crossover, instance crossover and two types of crossover based on ontology properties. The four types of operators behave differently depending on individuals used as operands. Class crossover operator generates a new structure of evolutionary ontology classes. Instance crossover results in modification of genetic individual instances. Finally, properties crossover determine new relations or new properties in the onto-space. In GP three types of crossover operator are mentioned: tree-based crossover, linear crossover and graph crossover. The tree-based crossover operator generates mutual exchange of subtrees. If the tree is the representation of a program then the crossover operator means joining some sequences of source code to sequences from other source code. In the case of linear crossover the exchange is made between sequences of instructions. The last type of crossover operator in GP, graph crossover, interchange subgraphs such that the resulted graphs to respect the relation between nodes and edges.

Four types of mutation operator are also encountered in EO depending on which element from the onto-space they are applied. Class mutation generates a new class structure by replacing all instances of a class with the instances of a random subclass of that class. Instance mutation signify the replacement of an instance with another instance from the same class as the initial instance, i.e the introduction of new instances into genetic individuals. Data property mutation means the change of the initial value depending on the type of data. The main contribution of object property mutation is the mirror effect, by replacing a relation with its inverse. In GP are identified three types of mutation operator: for tree representation, for linear representation and for graph representation. They are different types of operators with different results. In the tree representation we witness to modifying a node or subtree. Regarding linear representation, the mutation determine the change of an operator or an

operand from an instruction. For graph representation, two forms of mutation are identified which generate the change of a subgraph or of a node's link.

The selection operator used in EO is based on the models used in evolutionary strategies, namely (μ,λ)-selection or (μ+λ)-selection or on the model used in genetic algorithms, as Monte Carlo technique, unlike GP where is used mainly tournament selection.

The repair operator is mandatory in evolutionary ontologies, to remove inconsistencies generated by crossover and mutation. In GP repair operator is mentioned [13, 16, 46], but it is not imposed.

Other operators outside the standard ones were introduced in GP. Thus, new operators called geometric semantic operators were presented in [35]. Another new operator in GP called forking was introduced in [34]. EO hold classical genetic operators: selection, crossover, mutation, that have been adapted to the ontological character of the individuals or are completely new operators (as relational crossover and relational mutation). Further we consider as future work in EO the implementation of new operators due to the complexity of the genetic individuals.

V. CONCLUSIONS

The gap between GP and EO, shown in Table I is so large that make the evolutionary ontologies a distinct domain.

TABLE I. THE DIFFERENCES BETWEEN GP AND EO

Aspect	GP instantiation of the aspect	EO instantiation of the aspect
Inteligence level	Subsymbolic, entirely mathematical algorithms	Symbolic concepts evolved with subsymbolic algorithms
Solution space	The set of programs, defined depending on the problem at hand	An ontological space containing the possible instances and their relations
The individuals	Programs	Ontologies
Crossover	Three basic types tree-based crossover, linear crossover, graph crossover	Actually we have four different operators, one for classes, one for instances and two for properties
Mutation	Three basic types for tree representation, for linear representation, for graph representation	Like in the case of crossover, there are four different mutations for classes, instances, data properties and object properties
Repair operator	Needed sometimes, although it did not exist in the early stages of GP	Absolutely needed as the individuals may contain very complex internal and external relations
Selection	Tournament selection	Deterministic or Monte Carlo based
New operators	Geometric semantic operators Forking operator	New specific operators may be defined because ontologies imply different concepts and princples

In this article we show that between evolutionary ontologies and genetic programming there are significant differences, although they are both fields of genetic computation and share common aspects:

- both are fields of evolutionary computation;
- their algorithms are very similar, implying individuals which evolve undergoing some genetic operators, out of which three are classical: crossover, mutation and selection;
- both have the purpose of optimization;
- in certain cases, the representation of individuals is tree-based, therefore some operators (crossover and mutation) may be similar, however, not identical.

As future work, evolutionary computation and ontologies may be used together, especially in the context of multi-agent systems, which are complicated and hard to create, according to [10].

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] D. Aerts, and M. Czachor, "Quantum aspects of semantic analysis and symbolic artificial intelligence," arXiv preprint quant-ph/0309022, 2003.

[2] Y. Afacan, and H. Demirkan, "An ontology-based universal design knowledge support system," Knowledge-based systems, vol. 24, no. 4, pp. 530-541, 2011.

[3] R. Akerkar, and P. Sajja, Knowledge-based systems, Jones & Bartlett Publishers, Sudbury, MA, pp. 244-335, 2010.

[4] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks" Knowledge-based systems, vol. 8, no. 6, pp. 373-389, 1995.

[5] P. J. Angeline, "An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover," Proceedings of the 1st annual conference on genetic programming, Cambridge, Massachusetts, USA: July 28-31, pp. 21-29, 1996.

[6] D. Apostolou, G. Mentzas, and A. Abecker, "Managing knowledge at multiple organizational levels using faceted ontologies," Journal of Computer Information Systems, vol. 49, no. 2, pp. 32-49, 2008.

[7] J. Barrachina, P. Garrido, M. Fogue, F. J. Martinez, J. C. Cano, C. T. Calafate, and P. Manzoni, "VEACON: A Vehicular Accident Ontology designed to improve safety on the roads," Journal of Network and Computer Applications, vol. 35, no. 6, pp. 1891-1900, 2012.

[8] M. Brameier, and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," Evolutionary Computation, IEEE Transactions on, vol. 5, no. 1, pp. 17-26, 2001.

[9] D. Contras, and A. Pintescu, "Defining spatial relations in a specific ontology for automated scene creation," Carpathian Journal of Electronic and Computer Engineering, vol. 6, no. 1, pp. 50-55, 2013.

[10] C. Costea, "Distributed Constraint Optimization in Microgrid Operations", Carpathian Journal of Electronic and Computer Engineering, vol. 3, no. 1, pp. 13-16, 2010.

[11] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," Proceedings of the First International Conference on Genetic Algorithms, Pittsburgh, Pennsylvania, USA: July 1985, pp. 183-187.

[12] D. Dickmanns, J. Schmidhuber, and A. Winklhofer, "Der genetische algorithmus: Eine implementierung in prolog," Fortgeschrittenenpraktikum, Institut fur Informatik, Lehrstuhl Prof. Radig, Technische Universit at Munchen, 1987.

[13] J. H. Drake, M. Hyde, K. Ibrahim, and E. Ozcan, "A genetic programming hyper-heuristic for the multidimensional knapsack problem," Kybernetes, vol. 43, no. 9/10, pp. 1500-1511, 2014.

[14] H. J. Du, D. H. Shin, and K. H. Lee, "A sophisticated approach to semantic web services discovery," Journal of Computer Information Systems, vol. 48, no. 3, pp. 44-60, 2008.

[15] J. Du, and L. Zhou, "Improving financial data quality using ontologies," Decision Support Systems, vol. 54, no. 1, pp. 76-86, 2012.

[16] A. A. Freitas, "A genetic programming framework for two data mining tasks: classification and generalized rule induction," Genetic Programming 1997: Proc 2nd Annual Conf. Morgan Kaufmann, Stanford University, CA, USA: July 1997, pp. 96-101.

[17] C. Fujiki, "Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma," Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms, Cambridge, Massachusetts, USA: July 28-31, 1987, pp. 236-240.

[18] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, F. N. Noy, and S. W. Tu, "The evolution of Protégé: an environment for knowledge-based systems development," International Journal of Human-computer studies, vol. 58, no. 1, pp. 89-123, 2003.

[19] B. Goertzel, Perception processing for general intelligence: Bridging the symbolic/subsymbolic gap, Artificial General Intelligence. Springer Berlin Heidelberg, pp. 79-88, 2012.

[20] C. T. Jung, C. H. Sun, and M. Yuan, "An ontology-enabled framework for a geospatial problem-solving environment," Computers, Environment and Urban Systems, vol. 38, pp. 45-57, 2013.

[21] L. A. Kappelman, and J. A. Zachman, "The enterprise and its architecture: ontology & challenges," Journal of Computer Information Systems, vol. 53, no. 4, pp. 87-95, 2013.

[22] P. D. Karp, "Pathway databases: a case study in computational symbolic theories," Science, vol. 293, no. 5537, pp. 2040-2044, 2001.

[23] T. D. Kelley, "Symbolic and sub-symbolic representations in computational models of human cognition what can be learned from biology?," Theory & Psychology, vol. 13, no. 6, pp. 847-860, 2003.

[24] K. E. Kinnear Jr, "Evolving a sort: Lessons in genetic programming," Neural Networks IEEE International Conference on. IEEE, San Francisco, California, USA: March 28-April 1, 1993, pp. 881-888.

[25] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection, (Vol. 1), MIT press, Cambridge, Massachusetts, 1992, pp. 71-78, pp. 91, pp. 108, pp. 604.

[26] W. B. Langdon, "The evolution of size in variable length representations," Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on. IEEE, Anchorage, AK, USA: May 4-9, 1998, pp. 633-638.

[27] S. E. Lewis, "Gene Ontology: looking backwards and forwards," Genome Biology, vol. 6, no. 1, pp. 103, 2005.

[28] X. Li, and K. Hirasawa, "Continuous probabilistic model building genetic network programming using reinforcement learning," Applied Soft Computing, vol. 27, pp. 457-467, 2015.

[29] S. C. J. Lim, Y. Liu, and W. B. Lee, "A methodology for building a semantically annotated multi-faceted ontology for product family modelling," Advanced Engineering Informatics, vol. 25, no. 2, pp. 147-161, 2011.

[30] X. Ma, E. J. M. Carranza, C. Wu, and F. D. van der Meer, "Ontology-aided annotation, visualization, and generalization of geological time-scale information from online geological map services," Computers & geosciences, vol. 40, pp. 107-119, 2012.

[31] O. Matei, "Defining an Ontology for Diagnosis Based on Radiographic Images," Carpathian Journal of Electronic and Computer Engineering, vol. 1, no. 1, 2008.

[32] O. Matei, D. Contras, and P. Pop, "Applying evolutionary computation for evolving ontologies," Evolutionary Computation (CEC), 2014 IEEE Congress on. IEEE, Beijing, China: July 6-11, 2014, pp. 1520-1527.

[33] B. McKay, M. J. Willis, and G. W. Barton, "Using a tree structured genetic algorithm to perform symbolic regression," Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414). IET, Sheffield, UK: 12-14 Sep. 1995, pp. 487-492.

[34] A. Meier, M. Gonter, and R. Kruse, "Accelerating convergence in cartesian genetic programming by using a new genetic operator," Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, Amsterdam, Netherlands: July 06 - 10, 2013, pp. 981-988.

[35] A. Moraglio, K. Krawiec, and C, G. Johnson, "Geometric semantic genetic programming," Parallel Problem Solving from Nature-PPSN XII. Springer Berlin Heidelberg, 2012, pp. 21-31.

[36] D. L. Nastasie, and A. Koronios, "A Multidimensional Perspective on the Diffusion of Ontologies," Journal of Computer Information Systems, vol. 51, no. 2, pp. 49-59, 2010.

[37] A. C. N. Ngomo, and K. Lyko, "Eagle: Efficient active learning of link specifications using genetic programming," The Semantic Web: Research and Applications. Springer Berlin Heidelberg, 2012, pp. 149-163.

[38] M. Oltean, "Evolving evolutionary algorithms using linear genetic programming," Evolutionary Computation, vol. 13, no. 3, pp. 387-410, 2005.

[39] A. Petrovan, M. Lobontiu, and S. Ravai Nagy, "Broadening the Use of Product Development Ontology for One-off Products," Applied Mechanics and Materials, vol. 371, pp. 878-882, 2013.

[40] A. Petrovan, M. Lobontiu, G. Lobontiu, and S. Ravai-Nagy, "Overview on Equipment Development Ontology," Applied Mechanics and Materials, vol. 657, pp. 1066-1070, 2014.

[41] R. Poli, "Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming," ICGA, pp. 346-353, 1997.

[42] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, A field guide to genetic programming. Lulu. com, 2008, pp. 42-44.

[43] K. Rahmani, and V. Thomson, "Ontology based interface design and control methodology for collaborative product development," Computer-Aided Design, vol. 44, no. 5, pp. 432-444, 2012.

[44] R. Riolo, T. Soule, and B. Worzel, Genetic programming theory and practice IV, Vol. 4. Springer Science & Business Media, New York, 2007, pp.172-176.

[45] A. L. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of research and development, vol. 44, no. 1.2, pp. 206-226, 2000.

[46] L. Spector, and T. Helmuth, "Uniform linear transformation with repair and alternation in genetic programming," Genetic Programming Theory and Practice XI. Springer New York, 2014, pp. 137-153.

[47] K. Sycara, A. Pannu, M. Williamson, D. Zeng, and K. Decker, "Distributed intelligent agents," IEEE Intelligent Systems, vol. 11, no. 6, pp. 36-46, 1996.

[48] A. Teller, and M. Veloso, "PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System," No. CMU-CS-95-101. Carnegie-Mellon Univ. Pittsburgh PA Dept. of Computer Science, 1995.

[49] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," Genetic Programming and Evolvable Machines, vol. 12, no. 2, pp. 91-119, 2011.

[50] H. Wang, and S. Wang, "Ontology-based data summarization engine: a design methodology," Journal of Computer Information Systems, vol. 53, no. 1, pp. 48-56, 2012.

[51] M. Wooldridge, and N. R. Jennings, "Intelligent agents: Theory and practice," The knowledge engineering review, vol. 10, no. 2, pp. 115-152. 1995.

[52] H. Xie, and M. Zhang, Sampling Issues of Tournament Selection in Genetic Programming. School of Engineering and Computer Science, Victoria University of Wellington, 2009.