XML-based Genetic Programming Framework: Design Philosophy, Implementation and Applications

Ivan Tanev and Katsunori Shimohara

Department of Information Systems Design,
Faculty of Science and Engineering, Doshisha University,
1-3 Miyakodani, Tatara, Kyotanabe,Kyoto 610-0321, Japan

e-mail: {itanev, kshimoha}@mail.doshisha.ac.jp

Abstract

We present the design philosophy, the implementation and various applications of XML-based genetic programming (GP) framework (XGP). The key feature of XGP is the distinct representation of genetic programs as DOM-parse trees featuring corresponding flat XML-text. XGP contributes to the achievement of (i) fast prototyping of GP by using the standard built-in API of DOM-parsers for manipulating the genetic programs, (ii) human-readability and modifiability of the genetic representations (iii) generic support for the representation of grammar of strongly-typed GP using W3C-standardized XML-schema; and (iv) inherent inter-machine migratability of the text-based genetic representation (i.e., the XML text) in the distributed implementations of GP.

Keywords: genetic programming, strongly-typed genetic programming, genetic representation, XML, DOM.

1 Motivation

Developing controllers of mobile autonomous robots is often performed as a sequence of simulated off-line design (phylogenetic learning) of the robot's software model followed by on-line adaptation (ontogenetic learning) on the physical robot situated in real environment. Justification to incorporate off-line software simulation into the process of robots controller design comes from the facts that verification of robot behavior on physical robots is extremely time consuming and often dangerous for the robot and surrounding environment. These arguments in favor of software simulation of controller as a process, which precedes the physical design and online learning on real robot become even more relevant within the context of recently emerged trends in robotics such as investigating the social behavior, the role and value of communication in emergent coherence, cooperation and collaboration in the robots' societies situated in inherently competitive or cooperative environments. Simulating robot controllers as software agents and focusing on the model of their relevant features is viewed as a promising way to address the above mentioned shortcomings of direct verification of behavior on physical robots.

GP, which we propose as an approach for offline learning implies that the agent's code is automatically designed by computer system via simulated evolution employing selection and survival of the fittest in a way similar to the evolution of species in the nature. While for many tasks handcrafting the agent code can be seen as natural approach, it might be unfeasible for most of real-world problems due to their typically enormous complexity. Moreover, in many problems the challenge is to develop a solution, which is competitive or even better than human-designed one. Such a solution might be well beyond the abilities of human to handcraft it.

The software model of the evolvable robot's controller should fulfill the basic requirements of being adequate, fast running, and quickly developed. Considering the adequacy of the model as beyond the scope of this document, we intend to highlight the issues related to the efficiency of the system (in terms of reduced developing- and execution time) for evolving agent's behavior. The typically slow developing time of GP stems from the highly specific semantics of main attributes of GP (representation, initial population, genetic operations and fitness evaluation) and the lack of generic support to these attributes in 3G algorithmic languages and corresponding software engineering standards. *Developing time* of GP can be significantly reduced incorporating commodity-off-the-shelf software components and standards in software engineering of GP. The *runtime* of GP can be reduced as a cumulative result of reduced computational effort (the amount of individuals that should be processed in order to obtain a solution with specified probability) and increased computational performance (the amount of individuals evaluated per unit of time).

The *objective* of our research is to develop a genetic representation of the evolvable autonomous agents, which based on commodity-off-the-shelf software components and widely adopted industrial standards, would facilitate the achievement of easy and quick development phase of GP and would contribute to the achievement of better computational effort. The long-term aim is to employ such a representation in our research on the emergence and the survival value of social behavior and communication in multi-agent systems.

The remaining of the document is organized as follows. Section 2 introduces GP as algorithmic paradigm for offline learning of behavior of predator agents in predator-prey multi-agent systems (MAS). It also elaborates the proposed approach of representing evolvable agents (genetic programs) as DOM-parsing trees and discusses its design-time and runtime-related implications. Section 3 presents the result of verification of our approach on predator-prey pursuit problem. Conclusion is drawn in Section 4.

2 Approach

2.1 Algorithmic Paradigm

We consider a set of stimulus-response rules as a natural way to model the reactive behavior of autonomous agents which in general can be evolved using artificial neural networks, genetic algorithms, and GP. GP is a domain-independent problem solving approach in which a population of computer programs (individuals) is evolved to solve problems [2]. The simulated evolution in GP is based on the Darwinian principle of reproduction and survival of the fittest. In GP individuals are represented as parsing trees whose nodes are functions, variables or constants. The nodes that have sub-trees are non-terminals - they represent functions where the sub-trees represent the arguments to function of that node. Variables and constants are terminals – they take no arguments and they always are leaves in the parsing tree. The set of terminals for evolving agent's behavior in predator-prey MAS [3] includes the relevant stimuli such as perceptions (e.g. distance and visible angle to various objects in the world), its own state, etc.; and the response (actions) which the agent is able to perform. The most relevant functions (non-terminals) are the arithmetical and logical operators (>, <, =, +, - , etc.), and the `IF-THEN` function, establishing the relationship between certain stimulus and corresponding response. A sample stimulus-response rule is shown in Figure 1. It expresses a reactive behavior of turning to the bearing (angle) of the peer agent (`Peer_a`) plus `10` (degrees) as a result of stimulus of distance to that agent (`Peer_d`) being less than `20` (mm). A parsing tree of rule, depicted in Figure 1 is shown in Figure 2.

```
IF (Peer_d<20) THEN Turn(Peer_a+10)
```

Figure 1. Sample stimulus-response rule governing the agent behavior.

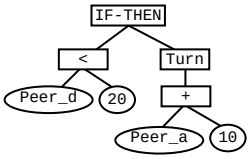

Figure 2. Sample stimulus-response rule represented as a parsing tree in GP.

Parsing-tree representation and its flat equivalents (LISP S-expression, postfix or prefix notations) are typically maintained and manipulated by GP-systems in a customized way. Therefore, an eventual tailoring of the available general purpose GP-systems for the task of evolving autonomous agents would be a time consuming approach. Design of GP-system from scratch would be (if not the only feasible) at least faster and more flexible way, providing that the implementation of main attributes of GP is based on widely adopted industrial standards and off-the-shelf technologies.

2.2 Genetic Representation

Inspired by flexibility and recently emerged widespread adoption of document object model (DOM) and extensible markup language (XML), we propose an approach of representing genetic program as a DOM-parsing tree featuring corresponding flat XML text. Our additional inspiration comes from the fact that despite of the reported use of DOM/XML for representing computer architectures, source codes, and agents' communication languages we are not aware about any attempts to employ this technology for representing evolvable structures such as genetic programs in generic, standard, and portable way. Our approach implies performing genetic operations on DOM-parsing tree using off-the shelf, platform- and language neutral DOM-parsers, and using XML-text representation as a format, feasible for migration among the computational nodes in eventual distributed GP.

The DOM-parsing tree of the sample rule considered earlier (Figure 1) looks in exactly the same way as parsing tree in canonical representation of genetic programs (Figure 2). However the flat representation of the rule is XML (Figure 3), rather than LISP S-expression.

```
<GP>
  <IF-THEN>
    <LE>
      <PERC>Peer_d</PERC>
      <PERC>20</PERC>
    </LE>
    <TURN>
      <PLUS>
        <PERC>Peer_a</PERC>
        <PERC>10</PERC>
      </PLUS>
    </TURN>
  </IF-THEN>
</GP>
```

Figure 3. XML representation of stimulus-response rule in GP.

2.3 Design-time Implications

In contrast to the typical approaches to manipulate parsing trees using custom code and representations, the proposed XML-based genetic programming (XGP) offers benefits of requiring *minimum programming efforts* and allowing developers to use the software platform, developing language, and/or programming paradigm which better fits the aims of concrete implementation of GP. These benefits are result of:

- Use of API of DOM-parsers: parsing tree of genetic program is manipulated using built-in API of DOM-parsers,
- Platform neutrality of parsers: DOM-parsers are available for virtually any of widely used software platforms (e.g., as Java classes), facilitating the portability of GP across different software platforms,
- Language neutrality of parsers: DOM-parsers are also available as language-neutral components (e.g. Microsoft COM), offering the same programming model of parsing trees regardless of the language employed to develop the code of GP that manipulates them, and
- Paradigm neutrality of parsers: DOM-parsers are available for programming paradigms, such as database stored procedures, web-client and web-server-side scripts, etc.

2.4 Run-time Implications

The potential strength of GP to automatically evolve a set of stimulus-response rules featuring arbitrary complexity without the need to a priory specify the extent of

such complexity might imply an enormous computational effort caused by the need to discover a huge search space while looking for optimal solution to the problem. A well-known way to limit the search space, and consequently, to reduce the computational effort of GP is to impose a restriction on the syntax of evolved genetic programs based on their a priory known, domain specific semantics. The approach is known as strongly typed genetic programming (STGP) and its advantage over canonical GP in achieving better computational effort is well proven. Our objective is in XGP to realize a generic support of STGP.

Considering the same sample rule as shown in Figure 3, and multimodality of perception information, it is noticeable that all nodes (i.e. both the functions and their operands) are associated with *data types* such as distance (`Peer_d, 20`), angle (`Peer_a, 10`), Boolean (`Peer_d<20`), etc. An eventual arbitrary creation or modification of such genetic program semantically would make little sense: e.g. it is unfeasible to maintain a rule with stimulus-related part featuring arithmetical expression involving operands of different data types (e.g. distance, angle and Booleans) at least because physically they are of different dimensions. In addition, there is clear possibility in maintaining introns if such an expression compares perception variable with constant beyond the range of corresponding sensor (e.g. `Peer_d>1000`, in case that sensor range is `400`). Analogically, the semantics of action `Turn()` implies a parameter of corresponding data type - an angle. However data types are not explicitly specified in considered so far DOM/XML representation of genetic programs in XGP, and consequently, not evident for the routines that create and alter it. To address the issue, in XGP we explicitly introduced the notion of type (in a way similar to STGP) represented as XML-tag for all the entities the genetic programs are composed of. In addition, we established a set of rules (i.e. grammar) describing the allowed relationship between types in semantically meaningful genetic program. Routines that create and alter genetic programs (e.g. creation of initial population and mutation) refer to the type of entity they are going to currently alter and impose the corresponding constraints to the sub-tree structures or values being generated. In crossover only nodes (with corresponding sub-trees) of the same type can be swapped.

In XGP we represent the grammar of STGP as a *XML schema* [1][2], which is an official World Wide Web Consortium (W3C) recommended, standard way to define the relationship among the entities in XML-document (i.e. genetic program). Since the syntax of schema conforms to the XML-standard, the routines those create and alter the parsing tree of genetic program access the schema via API of DOM-parsers in a way, identical to the way of accessing DOM/XML-based representation of genetic program.

We developed a set of formal rules for translating a BNF-defined grammar of STGP into corresponding XML-schema. Without touching the details of such rules, we present the resulting fragment of XML-schema (Figure 4) that corresponds to stimulus-related part of sample rule illustrated in Figure 3. Notice the definition of the sensory abilities – the *morphology* of the agent: the kind of perception information `(Wall_d, Peer_d)` and the range of corresponding sensor `(0..400)`. In a similar way XML schema defines the response abilities (actions) of the agents. Considering the communication as response for the speaker and stimulus for listener, the XML-schema offers a generic way to define the *communication abilities* of the agent. The section of XML-representation of strongly typed genetic program created applying the same fragment of XML schema is shown in Figure 5.

```
<xs:complexType name="IF-THEN"><xs:sequence>
  <xs:element name="COND-THEN" type="COND-THEN" />
  <xs:element name="THEN" type="THEN" />
</xs:sequence></xs:complexType>
<xs:complexType name="COND-THEN">
  <xs:choice>
  <xs:element name="COND_TDist" type="COND_TDist" />
  ...
</xs:choice></xs:complexType>
<xs:complexType name="COND_TDist">
  <xs:sequence>
  <xs:element name="VAR_TDist" type="VAR_TDist" />
  <xs:element name="OPER_TDist" type="OPER_TDist" />
  <xs:element name="CONST_TDist" type="CONST_TDist" />
</xs:sequence></xs:complexType>
<xs:simpleType name="VAR_TDist">
  <xs:restriction base="xs:string">
  <xs:enumeration value="Wall_d" />
  <xs:enumeration value="Peer_d" />
</xs:restriction></xs:simpleType>
<xs:simpleType name="OPER_TDist">
  <xs:restriction base="xs:string">
  <xs:enumeration value="GE" />
  <xs:enumeration value="LE" />
</xs:restriction></xs:simpleType>
<xs:simpleType name="CONST_TDist">
  <xs:restriction base="xs:integer">
  <xs:minInclusive value="0" />
  <xs:maxInclusive value="400" />
</xs:restriction></xs:simpleType>
```

Figure 4. XML schema of XGP, defining the grammar corresponding to the stimulus-related part of rule shown in Figure 1 and Figure 2.

```
<GP>
  <IF-THEN>
    <COND-THEN>
      <COND_TDist>
        <VAR_TDist>Peer_d</VAR_TDist>
        <OPER_TDist>LE</OPER_TDist>
        <CONST_TDist>20</CONST_TDist>
      </COND_TDist>
    </COND-THEN>
    <THEN>
    ...
    </THEN>
  </IF-THEN>
</GP>
```

Figure 5. XML-representation of strongly typed genetic program, created applying the fragment of XML schema shown in Figure 4.

3 Verification

In order to verify the feasibility of XGP and its design- and run-time implication we implemented a prototype of GP system for offline phylogenetic learning of agents in predator-prey pursuit problem. The problem comprises four predators (agents) whose goals are to capture a prey by surrounding it on all sides in a world (Figure 6) [3]. XGP runs on Windows OS system and employs Microsoft DOM parser – MSXML4.0. Considering the elaboration of the issues related to the emergence of surrounding behavior as irrelevant to the aims of this document and focusing on the verification of proposed approach as technology for representing genetic

programs, we would like to summarize the result as follows:

- Developing the prototype of XGP is significantly alleviated by use of DOM-parser. We measured about few [men x days] of development efforts (without considering the development of user interface), and
- XML schema in XGP offers generic way to impose the semantics constraints of genetic programs represented as DOM-parsing trees. The resulting computational effort for considered instance of predator-prey problem is relatively low and is in order of several thousands evaluations of genetic programs.

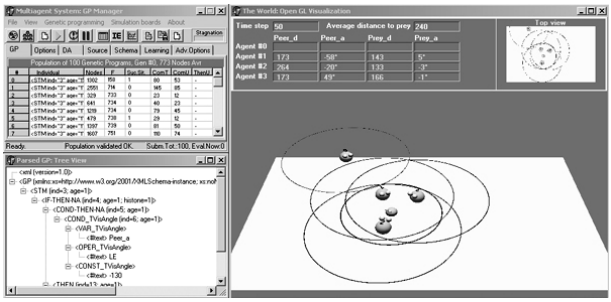

Figure 6. Snapshot of XML, employed for evolution of predator agents in predator-prey MAS

We observed similar design-, and run-rime implications of XGP, also applied for the evolution of snake-like robot (Figure 7) [4], Sony's Aibo quadruped robot (Figure 8), and computer-controlled scale car [5].

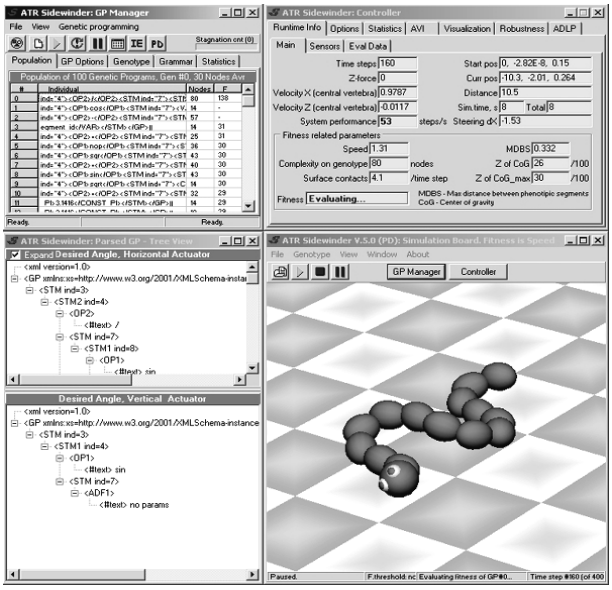

Figure 7. XGP applied for evolution of locomotion gaits of simulated snake-like robot.

4 Conclusion

We presented the result of our work on the role of genetic representation in facilitating quick design of efficiently running offline phylogenetic learning via GP. We proposed an XML-based genetic programming featuring a portable representation of evolvable agents (genetic programs) based on widely adopted DOM/XML standard. The manipulation of genetic programs implies the use of build-in API of off-the-shelf DOM-parsers. The approach features significant reduction of time consumption of usually slow process of software engineering of GP. In addition it offers a generic way to facilitate the reduction of computational effort via limitation of search space of GP by handling of only semantically correct genetic programs. Consistent with the concept of strongly typed GP, an approach of using W3C-recommended standard XML schema is developed as a generic way to represent and impose the grammar rules. The ideas laid in the foundation of the proposed approach are verified on the implementation of genetic programming for evolving social behavior of agents in predator prey pursuit problem. Due to the domain neutrality of GP, the approach can be applied for quick developing of efficiently running GP in various problem domains.

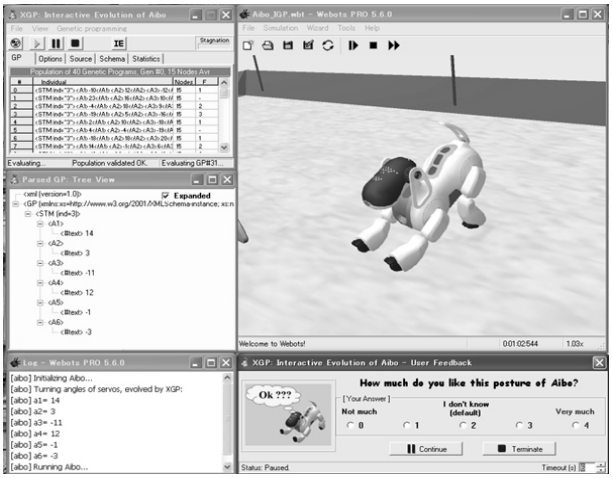

Figure 8. XGP applied for evolution of postures of the model of Sony's Aibo quadruped robot.

References

[1] D.Beech, M.Maloney, N.Mendelsohn, H. Thompson, XML Schema Part 1: Structures. W3C Recommendation, 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[2] J.R.Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, MA: MIT Press, 1992.

[3] I. Tanev, M. Brzozowski, and K. Shimohara, Evolution, Generality and Robustness of Emerged Social Behavior in Continuous Predators-prey Pursuit Problem, Genetic Programming and Evolvable Machines, Vol.6, Number 3, 2005, 301-318

[4] I. Tanev, T. Ray, and K. Shimohara, Exploring the Analogy in the Emergent Properties of Locomotion Gaits of Snakebot Adapted to Challenging Terrain and Partial Damage, Journal of ISCI, Vol.19, No.6, 2006, 220-232

[5] I. Tanev, and K. Shimohara, Evolution of Agent, Remotely Operating a Scale Model of a Car through a Latent Video Feedback, Journal of Intelligent Robotic Systems, No.52, Springer, 2008, 263-283