International Journal of Computer Applications (0975 – 8887)
Volume 114 – No. 6, March 2015

Analyze Effects of the Genetic Programming-based Emergence Engineering in Trustiness of Engineering Solutions

Babak Farhadi
Computer Engineering Faculty, Qazvin Islamic Azad University, Qazvin, Iran

Eslam Nazemi
Computer Engineering Faculty, Shahid Beheshti University, Tehran, Iran

ABSTRACT

In self-organization filed, Emergence Engineering is a new idea in software engineering scope which aims at setting up emergent phenomena in categories of individuals in order to extract those phenomena for engineering solutions. We inflict the needs of functional proportionate to a dynamic system and attend for it to assort. In this paper we analyze the effects of the clarification of the behavioral explanation in terms of trustiness of the solutions.

Keywords

Emergence engineering, genetic algorithm, self-organization, statute-based genetic programming.

1. INTRODUCTION

The last expansions in massively distributed systems show that common engineering approaches attain their restrictions when we have to tackle with a huge number of interacting, autonomous platforms. In this regards, swarms illustrate that huge collections of individuals may produce a useful group behavior whereas the individual behavior may be difficult to determine. On the other hand, the divide-and-conquer approach, realizing self-organization capabilities as found in nature have gained significant interest. Utilizing emergent phenomena into an engineering process is called Emergence Engineering, a new idea in software engineering scope.

In this paper, we discuss the contribution of the base language to the evolution success, and we comment on the overall practicability of creating solutions by emergence.

2. RELATED WORK

For agent systems, Genetic Programming is also a well-known approach in the context of foraging simulations [2] or rendezvous procedures [3]. Cramer was the first one to utilize genetic algorithms and tree-like structures to evolve programs [4]. These concepts all address either optimization problems or solutions for specific problem areas. There is currently only few related work concerning emergence engineering for agent societies. The most notable work here is ADELFE [5] which is an engineering approach explicitly exploiting emergence among a set of cooperative agents. We call ADELFE an online emergence engineering approach, due to the fact that the agents are situated in the real environment and need to self-organize to respond to changes in the environment.

3. FRAMEWORK OVERVIEW

In Genetic Programming, a crowd of individuals with a quite random genome is made. All individuals of the population are tested for compatibility according to the target functions. A latest election process filters out the individuals with low adaptability and allows those with ideal adaptability to enter the intercross pool with a higher probability. In the proliferation step, child is made by combining these solution candidates and integrated into the crowd. If the cancellation criterion is seen, the evolution stops here, otherwise the process is iterated, continuing by evaluating the new crowd. Our approach to Emergence Engineering is based on Genetic Programming which is a kind of evolutionary algorithms for breeding programs, algorithms, and same constructs.

3.1 Data flow Process

Data flow process consists of following steps. The story must be analyzed, resulting in a collection of needs. Appropriate target functions must be found which determine the compatibility of a solution. These functions should let for a piecemeal evaluation. The Genetic Algorithm repeatedly makes new versions of the individuals and elects the most appropriates ones. Sometimes, an individual is chosen out. If the Genetic algorithm converges, this individual will be most presumably better suited than any individual before. The individual is set in a component as its behavior and so deployed in the real environment.

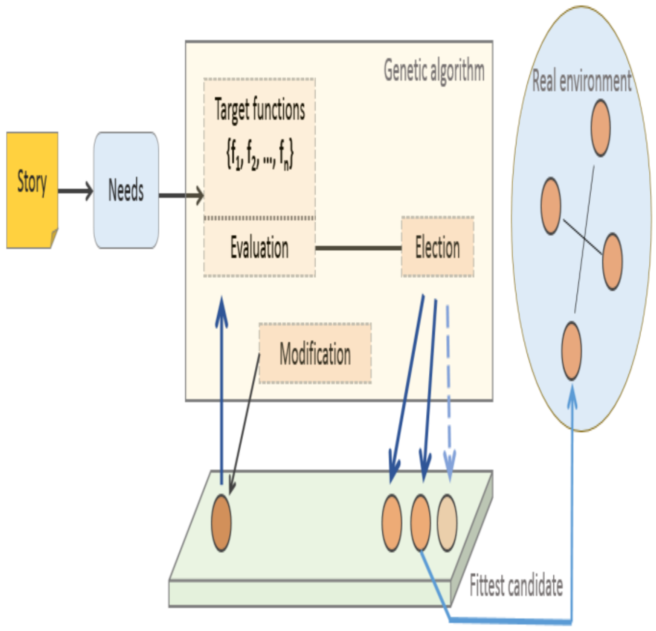


Fig 1: Data Flow Process.

To decrease complexity for the evolution process, we let only one kind of agent behavior evolve, which must be set in all participating agents. The suitable portion of the behavior may be elected by means of a state variable. As we have only one kind of individual, the election pressure on the individuals is distributed to the rule set of the individuals. As the set of rules apply some effect on the behavior of the agent, the individual behavior contributes to the totally behavior which can be measured by the compatibility function. Therefore, within our framework, we can testify emergence in distinct levels. The behavior of the crowd someway emerges from the behavior of the individuals, and the behavior of the individuals emerges from the rule set.

3.2 Expressiveness of the Behavioral Language

Unlike [1] and [6], our approach uses a new behavioral description which indicates an unordered set of rules instead of conducted instructions that we'll named this statute-based Genetic Programming (SBGP). SBGP is potent to declare many of the constructs known from high-level programming languages, but it lacks important capabilities similar indexed memory access. Therefore, to develop the clarity, we made new primitives for indirect memory access, using the notation [a(t)](s), which Implies for the value of the a(t)th symbol (at time step s = t or s = t +1) in the ordered list of all symbols. It (new form of Statute-based Genetic Programming named e SBGP) allows the evolution of list-sorting algorithms and in addition, it lets for conditions with more than two expressions, which enables the process to make more complex rules without the need for intermediary variables.

4. EXPRIMENTS

We evaluated experiments to specify requisites for an ideal emergence engineering. We divided them in two procedure.

4.1 Election Procedure

In this challenge, one agent is elected as "commander" and all agents accordant to this choice is the distributed nature. It must be ensured that all agents get a steadfast viewpoint of the election. In here, an agent may send a message containing a single number stored in the variable "out" with the command "send". If a message is received, its contents appear in the symbol "inbound" and the variable "inboundMsg" is set to 1. The agents have unique identifiers with variable of "ID" and two multi-purpose variables "a" and "b". We expect that identifier of the elected agent to be stored in variable "a" after about 2500 simulated time slices.

The target functions are determined as follows: f1 counts the number of different IDs found when comparing all the values stored in the "a" variables of the agents after the simulation and fines values that don't mark valid IDs. f2 defined the behavior size in terms of the number of rules, f3 counts the

time units used for active computations (penalizing useless computation when the node could sleep instead), and f4 counts the number of messages exchanged. For the best overall compatibility, all four functions must yield minimal values.

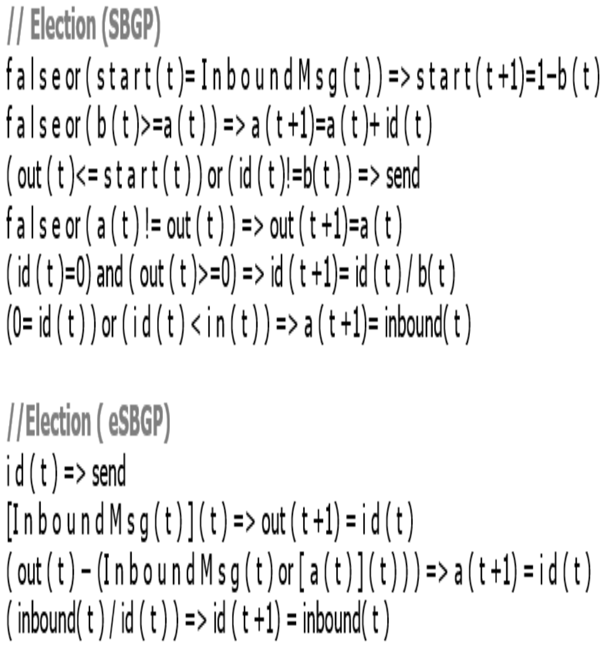
Some of the outcomes such as Message Extinction are well-known algorithms and other are incomprehensible. We evaluated the reliability of multiple solutions delivered by these two approaches by the fraction of procedures where the election process proceeds correctly. Programs generated with SBGP are reliable in 53% of the procedures and those from eSBGP achieve correct behavior in 97% of the network simulations. This seems to be a mention that the clarity of the language can help to find better solutions.

4.2 Critical Procedure

Code that accesses a shared resource is called critical procedure (CP). Processes running simultaneously on different nodes have to decide whether they are allowed to access the CP or whether they have to wait, using message exchange to coordinate themselves.

The first target function f1 evaluates the number of contraventions of them mutual exclusion criterion. To increase pressure, we sum up the square of the number of nodes inside the CP for each time step. The second target function f2 represents the number of times each process could enter the CP at least in the constant time range of the simulation. We add a value commensurate to the total number of accesses of the CP. Finally, f3 counts the number of rules and exerts pressure to drop dispensable rules. The target function f1 and f3 are subject to minimization, f2 is to be maximized.

The SBGP process didn't converge to a special set of proportional solutions, even after more than 3200 generations. We evaluated the behavior of the best individuals in 170 random network configurations. For SBGP, we found a solution avoiding collisions in 94.2% of all given environments. In 48.4% of the environments, the solution was also fair, allowing more than one process to enter the CP. Our tests also demonstrated that lower rates of collisions correlate with lower compatibility. Involving more agents obviously increases the chances of triggering violations.



```
// Election (SBGP)
falseor(start(t)=InboundMsg(t))=>start(t+1)=1-b(t)
falseor(b(t)>=a(t))=>a(t+1)=a(t)+id(t)
(out(t)<=start(t))or(id(t)!=b(t))=>send
falseor(a(t)!=out(t))=>out(t+1)=a(t)
(id(t)=0)and(out(t)>=0)=>id(t+1)=id(t)/b(t)
(0=id(t))or(id(t)<in(t))=>a(t+1)=inbound(t)

//Election ( eSBGP)
id(t)=>send
[InboundMsg(t)](t)=>out(t+1)=id(t)
(out(t)-(InboundMsg(t)or[a(t)](t)))=>a(t+1)=id(t)
(inbound(t)/id(t))=>id(t+1)=inbound(t)
```

Fig 2: SBGP and eSBGP in solutions.

5. EVALUATION

In some procedures like election, the engineering process may benefit from the higher expressiveness, while in others, it fails to create better solutions. An important key to a productive usage of this engineering process is to find out early which procedure is well suited, and which is not. Analyzing our experiments, we found special cases for the result of the evolution as any-agent and all-agent behaviors. The former one refers to the observation that eventually, one of the agent instances has got some property or expresses some behavior. The latter case refers to the situation where eventually, the whole collective adopts the same properties and behaviors.

In order to explore this approach, we have applied Genetic Programming to an interesting aspect of distributed systems, the load balancing problem. In load balancing scenarios, tasks continuously enter a system consisting of multiple workstations. Each task needs a different amount of time to finish. The target is to reduce the overall waiting time by distributing the workload equally between the stations. Traditional forms of load balancing are performed by a central instance assigning the tasks to the different processors; modern methods rely on decentralized cooperation of the workstations. In our version of the load balancing scenario, we pack each task into a single agent who has to decide itself on which station it wants to run. The goal of this agent society is to evenly distribute its workload on all computers in a grid.

Load balancing and election seem to be suitable problems for an emergence engineering approach. We believe that this is due to the fact that both problems require all-agent behaviors. For load balancing, each agent flights as soon as there is a host with lower load, while for election, all agents eventually share the same property at the end, namely knowing the ID of the winner.

CP is more than an election problem, due to the fact that we also want to reach compatibility. More specifically, the CP problem requires an iterative election, and it requires the behavior of the group to change in order to achieve compatibility. This is neither an any-agent nor an all-agent behavior. The agents somehow need to learn that some agent was already allowed to enter the CP, and so it does not qualify to enter again for some time. While it is not impossible that agents develop learning behavior with the evolutionary process, it is obviously fairly unlikely. Hence, for the applicability of this approach, the analysis must take care whether the group behavior implies learning capabilities or not.

6. CONCLUSION

Emergence engineering may sound like a contradiction, but as we showed in this article, a viable approach exists that exploits emergent phenomena for the creation of agent behavior. Yet it is profoundly different from the classical idea of software engineering. For an emergent process to occur and to reach a reasonable result, the targets must be carefully formulated, and the capabilities of the agents must be defined, but specific expectations should be avoided.

We've presented some simple criteria which may indicate whether our approach is likely to produce suitable solutions for a problem. If we have considered a problem to be suitable for our approach, we must find appropriate objective functions for the evolutionary algorithms to measure the fitness of individuals. Finally, we have to decide on the expressiveness of the behavioral language and the capabilities of the agents. We found that increasing the expressiveness of the implementation language of the agent behavior need not necessarily yield better solutions. Although we increased the scope of the possible agent behavior, some problems seem to trap the evolution in sub-optimal areas. We believe that defining some more procedures will provide us with more experience on the behavior of the evolution process and whether it may be required to introduce additional modifications.

Our research is still in an early state where we need to conduct more experiments to find out characteristics of problems which may be suitably handled by this approach. However, we must face the fact that generating programs in this way is extremely time-consuming, literally taking days to weeks until a solution is found.

7. REFERENCES

[1] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: a survey," Genetic Programming and Evolvable Machines, vol. 11, pp. 365-396, 2010.

[2] F. H. Bennett III, "Emergence of a multi-agent architecture and new tactics for the ant colony food foraging problem using genetic programming," in Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, 1996, pp. 430-439.

[3] H. Iba, "Emergent cooperation for multiple agents using genetic programming," in Parallel Problem Solving from Nature—PPSN IV, ed: Springer, 1996, pp. 32-41.

[4] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in Proceedings of the First International Conference on Genetic Algorithms, 1985, pp. 183-187.

[5] C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard, "ADELFE: a methodology for adaptive multi-agent systems engineering," in Engineering Societies in the Agents World III, ed: Springer, 2003, pp. 156-169.

[6] M. Zapf and T. Weise, "Offline emergence engineering for agent societies," in Proc. of the Fifth European Workshop on Multi-Agent Systems EUMAS, 2007.