Difference of tree maximum depth in Genetic Programming to generate competitive agents for RTS games

P. García-Sánchez, A. Fernández-Ares, A. M. Mora, P. A. Castillo and J.J. Merelo

Dept. of Computer Architecture and Technology and CITIC-UGR, University of Granada, Spain pgarcia@atc.ugr.es

Abstract. This work presents the results obtained from comparing ... Results show that

1 Introduction

Real Time Simulators (RTS) are a type of videogame that...

One of the most used games for study computational intelligence in RTS is *Planet Wars* [1–3] because it is a simple RTS that defines a ... Although it has been described in previous works, we summarize saying that the objective of the player is to conquer enemy and neutral planets in a space-like simulator. Each player has planets that produce ships depending of a growth-rate and the player must send this ships to other planets (literally, crashing towards the planet) to conquer them. A player win if is the owner of all the planets. As requirements, each second a decission has to be made (turn), and no memory about the previous turns must be used. Figure 1 show a screen capture of the game.

In this work we use Genetic Programming (GP) to obtain agents that play Planet Wars game. The objective of GP is to create functions or programs to solve determined problems. Individual representation is usually in form of a tree, formed by operators (or *primitives*) and variables (*terminals*). These sets are usually fixed and known. The genome size is, therefore, variable, but the maximum size (depth) of the individuals is usually fixed, to avoid high evaluation costs. GP has been used to evolve LISP (LISt Processing) programs [4], or XSLT (eXtensible Stylesheet Language Transformations) scripts [5], among others.

We try to solve the next questions:

- Can a tree-generated behaviour of an agent defeat a hand-coded one whose parameters have been also optimized?
- Can this agent beats a more complicated oponent that is adapted to the environment?
- How does the maximum depth affects the results?

The rest of the work is structured as follows: after the state of the art, the description of our agent is presented in Section 3. Then, the experimental setup conduced with the EA are showed (Section 4). Finally, results, conclusions and future works are discussed.

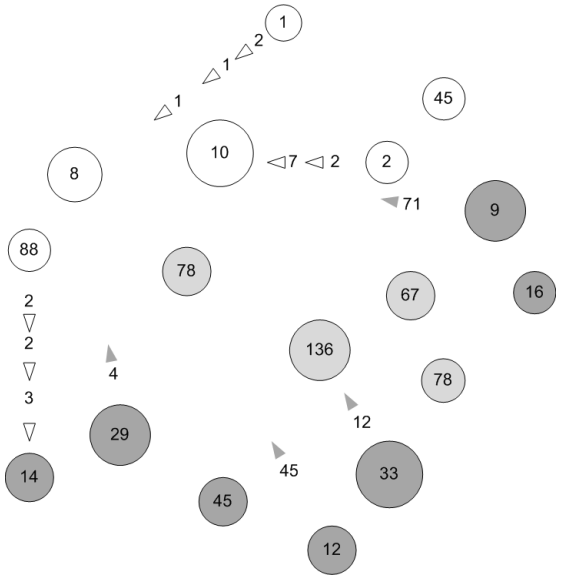

Fig. 1: Example of execution of the Player Wars game. White planets and ships are owned by the player and dark gray ones are controlled by the enemy. Clear gray are neutral planets (not invaded).

2 State of the art

RTS games (see [6] for a survey).

Among other techniques, Evolutionary Algorithms (EAs) have been widely used in computational intelligence in RTS games [6]. For example, for parameter optimization [7], learning [8] or content generation [9].

One of these types, Genetic Programming, has been proved as a good tool for developing strategies in games achieving results comparable to human, or human-based, competitors [10], and achieving higher ranking than solvers produced by other techniques or even beating high-ranking humans [11]. GP has also been used in different kind of games, such as board-games [12], or (in principle) simpler games such as Ms. Pac-Man [13] and Spoof [14] and even in modern video-games such as First Person Shoothers (FPS) (for example, Unreal [15]).

Planet Wars, the game we are going to use in this work, has been used as experimental environment in other works. For example, in [2] the authors programmed the behaviour of a bot with a decission tree of 3 levels. Then, the values of these rules were optimized using a genetic algorithm to tune the strategy rates and percentages. Results showed a good performance confronting with other bots provided by the Google AI Challenge. In their next work [3] they improved this agent optimizing in different types of maps and selecting the set of optimized parameters depending of the map where the game was taking

place, using a tree of 5 levels. These results outperformed the previous version of the bot with 87% of victories.

In this paper we use Genetic Programming to create the decission tree, instead of using our own gaming experience to model it, and compare this agent with the two presented before.

3 Proposed Agent

The proposed agent receives a tree to be executed. The generated tree is a binary tree of expressions formed by two different types of nodes:

- *Decission*: a logical expression formed by a variable, a less than operator (<), and a number between 0 and 1. They are the equivalent to the "primitives" in the field of GP.
- *Action*: the leaves of the the tree (therefore, the "terminals"). Each decission is the name of the method to call that indicates to which planet send a percentage of available ships (from 0 to 1) from the planet that executes the tree.

The different variables for the decissions are:

- *myShipsEnemyRatio*: Ratio between the player's ships and enemy's ships.
- *myShipsLandedFlyingRatio*: Ratio between the player's landed and flying ships.
- *myPlanetsEnemyRatio*: Ratio between the number of player's planets and the enemy's ones.
- *myPlanetsTotalRatio*: Ration between the number of player's planet and total planets (neutrals and enemy included)-
- *actualMyShipsRatio*: Ratio between the number of ships in the specific planet that evaluates the tree and player's total ships.
- *actualLandedFlyingRatio*: Ratio between the number of ships landed and flying from the specific planet that evaluates the tree and player's total ships.

The decission list is:

- *Attack Nearest (Neutral—Enemy—NotMy) Planet*: The objective is the nearest planet.
- *Attack Weakest (Neutral—Enemy—NotMy) Planet*: The objective is the planet with less ships.
- *Attack Wealthest (Neutral—Enemy—NotMy) Planet*: The objective is the planet with higher lower rate.
- *Attack Beneficious (Neutral—Enemy—NotMy) Planet*: The objective is the planet more beneficious, that is the one with growth rate divided by the number of ships.

```
if(myShipsLandedFlyingRatio<0.796)
        if(actualMyShipsRatio<0.201)
                attackWeakestNeutralPlanet(0.481);
        else
                attackNearestEnemyPlanet(0.913);
else
        attackNearestEnemyPlanet(0.819);
```

Fig. 2: Example of a generated Java tree.

- *Attack Quickest (Neutral—Enemy—NotMy) Planet*: The objective is the planet with higher facility to conquest: the lowest product between the distance from the planet that executes the tree and the number of the ships in the objective planet.
- *Attack (Neutral—Enemy—NotMy) Base*: The objective is the planet with more ships (that is, the base).
- *Attack Random Planet*.
- *Reinforce Nearest Planet*: Reinforce the nearest player's planet to the planet that executes the tree.
- *Reinforce Base*: Reinforce the player's planet with higher number of ships.
- *Reinforce Wealthest Planet*: Reinforce the player's planet with higher grown rate.
- *Do nothing*.

An example of a possible tree is shown in Figure 2. This example tree has a total of 5 nodes, with 2 decissions and 3 actions, and a depth of 3 levels.

The bot behaviour is explained in Algorithm 1.

```
At the beginning of the execution the agent receives the tree;
tree← readTree();
while game not finished do
    // starts the turn
    calculateGlobalPlanets();// e.g. Base or Enemy Base
     calculateGlobalRatios();// e.g. myPlanetsEnemyRatio
     foreach p in PlayerPlanets do
        calculateLocalPlanets(p);// e.g. NearestNeutralPlanet to p
         calculateLocalRatios(p);// e.g actualMyShipsRatio
         executeTree(p,tree);// Send a percentage of ships to destination
    end
end
```

Algorithm 1: Pseudocode of the proposed agent. The tree is fixed during all the agent's execution

4 Experimental Setup

To generate the tree we have used a Genetic Programming Algorithm. Evolutionary operators used are the sub-tree crossover and 1-node mutation. The choice is suggested by other researchers that have used these operators obtaining good results [15]. In our case the mutation randomly changes the decission of a node or mutate the value with an step-size of 0.25. Each configuration is executed 30 times, with a population of 32 individuals and a 2-tournament selector for a pool of 16 parents.

To test each individual during the evolution a battle with a previously created bot is performed in 5 different (but representative) maps provided by Google. The fitness used is a hierarchichal one previously used by other researchers in [2]. In our case, an individual is better than another if it wins in more maps. In case of equality of victories, then the individual with more turns to be defeated (i.e. it is stronger) is considered better. The maximum fitness is, therefore 5 victories and 0 turns. Also, as proposed by [2], and due to the noisy fitness effect, in every generation all individuals are re-evaluated.

Two publicly available bots have been chosen for our experiments[1]. The first bot to confront is *GeneBot*, proposed in the work [2]. This bot was trained using a GA to optimize the 8 parameters that conforms a set of hand-made rules. The other chosen bot is an advanced version of the previous one, called *ExGenebot* presented in [3]. This bot outperformed Genebot widely. ExGenebot bot analyses the distribution of the planets of the map to chose a previously optimized set of parameters.

After executing our algorithm without tree limitation in depth we also have executed with the lower and average levels obtained for the best individuals: 3 and 7, respectively, to study if this number has any effect in the results. Table 1 summarizes all the parameters used.

After all the executions we have evaluated all the obtained best individuals in all runs confronting to the bots in a larger set of maps (the 100 maps provided by Google) to study the behaviour of the algorithm and how good are the obtained bots in maps that have not been used for training.

The used framework is OSGiLiath, a service-oriented evolutionary framework. The generated tree is compiled in real-time and injected in the agent's code using Java CAssist library. All the source code used in this work is available under a LGPL V3 License in `http://www.osgiliath.org`.

5 Results

Table 2 summarizes all the obtained results of the execution of our EA. This table also shows the average age, depth and number of nodes of the best individuals obtained and also the average population at the end of the run. The average turns columns are calculated only taking into account the individuals with lower victories than 5, because this number is 0 if they have win the five battles.

[1] Both can be downloaded from `http://github.com/METEAQUILAURL`

Parameter Name	*Value*
Population size	32
Crossover type	Sub-tree crossover
Crossover rate	0.5
Mutation	1-node mutation
Mutation step-size	0.25
Selection	2-tournament
Replacement	Steady-state
Stop criterion	50 generations
Maximum Tree Depth	3, 7 and unlimited
Runs per configuration	30
Evaluation	Playing versus Genebot [2] and ExGenebot [3]
Maps used in each evaluation	

Table 1: Parameters used in the experiments.

Versus Genebot				
		Depth 3	Depth 7	Unlimited Depth
Best Fitness	Victories	**4.933** ± 0.25	4.83 ± 0.53	4,9 ± 0.30
	Turns	244.5 ± 54.44	466 ± 205.44	266,667 ± 40.42
Population Ave. Fitness	Victories	**4.486**± 0.52	4.43 ± 0.07	4,711 ± 0.45
	Turns	130.77± 95.81	139.43 ± 196.60	190,346 ± 102.92
Depth	Best	3 ± 0	5.2 ± 1.78	6,933 ± 4.05
	Population	3 ± 0	5.267 ± 1.8	7,353 ± 3.11
Nodes	Best	7 ± 0	13.667 ± 7.68	22,133 ± 22.21
	Population	7 ± 0	13.818 ± 5.86	21,418 ± 13.81
Age	Best	**8.133** ± 3.95	5.467 ± 2.95	5,066 ± 2.11
	Population	**4.297** ± 3.027	3.247 ± 0.25	3,092 ± 1.27
Versus ExGenebot				
		Depth 3	Depth 7	Unlimited Depth
Best Fitness	Victories	4.133 ± 0.50	4.2 ± 0.48	**4.4** ± 0.56
	Turns	221.625 ± 54.43	163.667 ± 106.38	123.533 ± 112.79
Population Ave. Fitness	Victories	3.541 ± 0.34	3.689 ± 0.37	**4.043** ± 0.38
	Turns	200.086 ± 50.79	184.076 ± 57.02	159.094 ± 61.84
Depth	Best	3 ± 0	5.2 ± 1.84	6.966 ± 4.44
	Population	3 ± 0	5.216 ± 0.92	6.522 ± 1.91
Nodes	Best	7 ± 0	12.6 ± 6.44	18.466 ± 15.46
	Population	7 ± 0	13.05 ± 3.92	16.337 ± 7.67
Age	Best	4.266 ± 5.01	4.133 ± 4.26	**4.7** ± 4.72
	Population	3.706 ± 0.58	3.727 ± 0.62	**3.889** ± 0.71

Table 2: Average results obtained from each configuration. Each one has been tested 30 times.

The second experiment results are shown in Table 3. In this experiment we have confronted the 30 bots obtained in each configuration again with Genebot and ExGenebot but in the 100 maps provided by Google. This table shows how the bots obtained using GP wins more times. It can be seen that it is dificult to beat the adapatative bot, because it is using a map-detection mechanism. BLABLABLA. Note that, confronting with ExGenebot, even obtaining individuals with maximum fitness (5 victories) that have been kept in the population several generations cannot be representative of a extremely good bot in a wider set of maps.

Configuration	Average maps won	Average turns
Versus Genebot		
Depth 3		
Depth 7		
Unlimited Depth		
Versus ExGenebot		
Depth 3	52.367 ± 13.39	9565.667 ± 5833.60
Depth 7	58.867 ± 7.35	7389.535 ± 3574.26
Unlimited Depth	56 ± 11.52	8247.967 ± 4133.72

Table 3: Results confronting the 30 best bots attained from each configuration in the 100 maps each.

6 Conclusions

This work presents a Genetic Programming algorithm that generates agents for playing Planet Wars game. A number of possible actions to perform and decission variables have been presented. We have used two competitive bots available in the literature (Genebot and ExGenebot) to use to set the fitness of the generated individuals. Three different tree maximum depths have been used: 3, 7 and unlimited. Results show that best individuals outperforms these agents during the evolution in all configurations. We also have tested our agents with a larger set of maps not used for evaluation, obtaining results equivalent or betters that the previous agents.

In future work, other rules will be added to our algorithm (for example, the ones that analyse the map, as the ExGenebot does). Other games used in the area of computational intelligence in videogames, such as Unreal or SuperMario will be tested.

References

1. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A procedural balanced map generator with self-adaptive complexity for the real-time strategy game planet wars. In: EvoApplications. Volume 7835 of Lecture Notes in Computer Science., Springer (2013) 274–283

2. Mora, A.M., Fernández-Ares, A., Guervós, J.J.M., García-Sánchez, P., Fernandes, C.M.: Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms. J. Comput. Sci. Technol. **27**(5) (2012) 1007–1023
3. Fernández-Ares, A., García-Sánchez, P., Mora, A.M., Guervós, J.J.M.: Adaptive bots for real-time strategy games via map characterization. In: 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, September 11-14, 2012, IEEE (2012) 417–721
4. Koza, J.R.: Genetically breeding populations of computer programs to solve problems in artificial intelligence. In: Tools for Artificial Intelligence, 1990.,Proceedings of the 2nd International IEEE Conference on. (1990) 819–827
5. García-Sánchez, P., Guervós, J.J.M., Laredo, J.L.J., García, A.M., Castillo, P.A.: Evolving xslt stylesheets for document transformation. In: Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings. Volume 5199 of Lecture Notes in Computer Science., Springer (2008) 1021–1030
6. Lara-Cabrera, R., Cotta, C., Fernández-Leiva, A.J.: A review of computational intelligence in rts games. In: FOCI, IEEE (2013) 114–121
7. Esparcia-Alcázar, A.I., García, A.I.M., García, A.M., Guervós, J.J.M., García-Sánchez, P.: Controlling bots in a first person shooter game using genetic algorithms. In: IEEE Congress on Evolutionary Computation, IEEE (2010) 1–8
8. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the nero video game. IEEE Transactions on Evolutionary Computation (2005) 653–668
9. Mahlmann, T., Togelius, J., Yannakakis, G.N.: Spicing up map generation. In: Proceedings of the 2012t European conference on Applications of Evolutionary Computation. EvoApplications'12, Berlin, Heidelberg, Springer-Verlag (2012) 224–233
10. Sipper, M., Azaria, Y., Hauptman, A., Shichel, Y.: Designing an evolutionary strategizing machine for game playing and beyond. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews **37**(4) (2007) 583–593
11. Elyasaf, A., Hauptman, A., Sipper, M.: Evolutionary design of freecell solvers. IEEE Transactions on Computational Intelligence and AI in Games **4**(4) (2012) 270–281
12. Benbassat, A., Sipper, M.: Evolving both search and strategy for reversi players using genetic programming. (2012) 47–54
13. Brandstetter, M., Ahmadi, S.: Reactive control of ms. pac man using information retrieval based on genetic programming. (2012) 250–256
14. Wittkamp, M., Barone, L., While, L.: A comparison of genetic programming and look-up table learning for the game of spoof. (2007) 63–71
15. Esparcia-Alcázar, A.I., Moravec, J.: Fitness approximation for bot evolution in genetic programming. Soft Computing **17**(8) (2013) 1479–1487