Tree depth influence in Genetic Programming for generation of competitive agents for RTS games

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

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

Abstract. This work presents the results obtained from comparing different tree depths in a Genetic Programming Algorithm to create agents that play the Planet Wars game. Three different maximum levels of the tree have been used (3, 7 and Unlimited) and two bots available in the literature, based on human expertise, and optimized by a Genetic Algorithm have been used for training and comparison. Results show that in average, the bots obtained using our method equal or outperform the previous ones, being the maximum depth of the tree a relevant parameter for the algorithm.

1 Introduction

Real Time Strategy (RTS) games are a type of videogame where the play takes action in real time (that is, there are not turns, as in chess). Well-known games of this genre are Age of Empires$^{\mathrm{TM}}$ or Warcraft$^{\mathrm{TM}}$. In this kind of game the players have units, structures and resources and they have to confront with other players to win battles. Artificial Intelligence (AI) in these games is usually very complex, because they are dealing with many actions and strategies at the same time.

The *Planet Wars* game, presented under the Google AI Challenge 2010[1] has been used by several authors for the study of computational intelligence in RTS games [1–3]. This is because it is a simplification of the elements that are present in the complex games previously mentioned (only one type of resource and one type of unit).

The objective of the player is to conquer enemy and neutral planets in a space-like simulator. Each player has planets (resources) that produce ships (units) depending on a growth-rate. The player must send these ships to other planets (literally, crashing towards the planet) to conquer them. A player win if he is the owner of all the planets. As requirements, the limit to calculate next actions (this time window is called *turn*[2]) is only a second, and no memory about the

[1] http://planetwars.aichallenge.org/

[2] Although in this work we are using this term, note that the game is always performed in real time.

previous turns must be used. Figure 1 shows a screen capture of the game. The reader is referred to [1–3] for more details of the game.

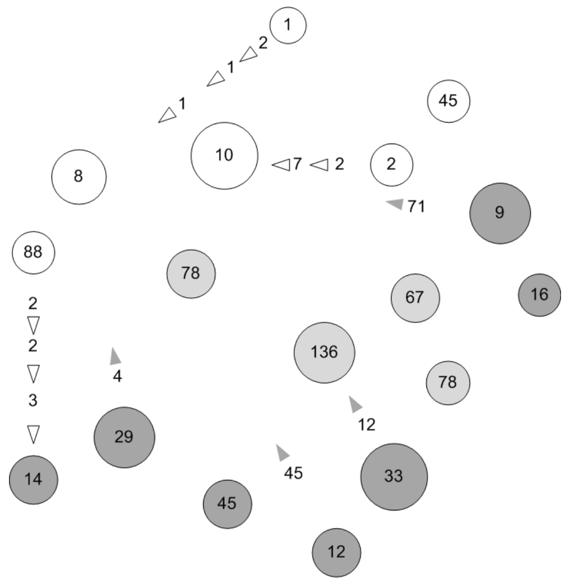


Fig. 1. Example of execution of the Player Wars game. White planets and ships are owned by the player and dark gray ones are controlled by the enemy. Clear gray are neutral planets (not invaded).

In this work Genetic Programming (GP) is used to obtain agents that play the 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 an agent hand-coded by a player with experience and whose parameters have been also optimized?
- Can this agent beat a more complicated opponent that is adapted to the environment?
- How does the maximum depth affect 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 GP is shown (Section 4). Finally, results, conclusions and future works are discussed.

2 State of the art

RTS games have been used extensively in the computational intelligence area (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]. They also have obtained 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 Shothers (FPS) (for example, Unreal$^{\mathrm{TM}}$ [15]).

Planet Wars, the game used in this work, has also been used as experimental environment for testing agents in other works. For example, in [2] the authors programmed the behaviour of a *bot* (a computer-controlled player) with a decision 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 [3] the authors improved this agent optimizing it in different types of maps and selecting the set of optimized parameters depending on 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 GP to create the decision tree, instead of expert human gaming experience to model it, and the resulting agent is compared 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:

- *Decision*: a logical expression formed by a variable, a less than operator ($<$), and a number between 0 and 1. It is the equivalent to a "primitive" in the field of GP.
- *Action*: a leave of the the tree (therefore, a "terminal"). Each decision is the name of the method to call from the planet that executes the tree. This method indicates to which planet send a percentage of available ships (from 0 to 1).

The different variables for the decisions 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*: Ratio 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 decision 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 Wealthiest (Neutral—Enemy—NotMy) Planet*: The objective is the planet with higher lower rate.
- *Attack Beneficial (Neutral—Enemy—NotMy) Planet*: The objective is the more beneficial planet, that is, the one with growth rate divided by the number of ships.
- *Attack Quickest (Neutral—Enemy—NotMy) Planet*: The objective is the planet easier to be conquered: the lowest product between the distance from the planet that executes the tree and the number of 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 Wealthiest 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 decisions and 3 actions, and a depth of 3 levels.

The bot behaviour is explained in Algorithm 1.

4 Experimental Setup

Sub-tree crossover and 1-node mutation evolutionary operators have been used, following other researchers' proposals that have used these operators obtaining

```
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.

```
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

good results [15]. In this case, the mutation randomly changes the decision of a node or mutate the value with a step-size of 0.25 (an adequate value empirically tested). 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 is played. Hierarchical fitness is used, as proposed in [2]. Thus, an individual is better than another if it wins in a higher number of maps. In case of equality of victories, then the individual with more turns to be defeated (i.e. the stronger one) 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, all individuals are re-evaluated in every generation.

Two publicly available bots have been chosen for our experiments[3]. The first bot to confront is *GeneBot*, proposed in [2]. This bot was trained using a GA to optimize the 8 parameters that conforms a set of hand-made rules, obtained from an expert human player experience. The second one is an advanced version of the previous, called *Exp-Genebot* (Expert Genebot) [3]. This bot outperformed

[3] Both can be downloaded from `https://github.com/deantares/genebot`

Genebot widely. Exp-Genebot bot analyses the distribution of the planets in the map to chose a previously optimized set of parameters by a GA. Both bots are the best individual obtained of all runs of their algorithm (not an average one).

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

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 Exp-Genebot [3]
Maps used in each evaluation	map76.txt map69.txt map7.txt map11.txt map26.txt

Table 1. Parameters used in the experiments.

After all the executions we have evaluated 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 [16]. The generated tree is compiled in real-time and injected in the agent's code using Javassist [4] library. All the source code used in this work is available under a LGPL V3 License in `http://www.osgiliath.org`.

5 Results

Tables 2 and 3 summarize all the obtained results of the execution of our EA. These tables also show 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 rows are calculated only taking into account the individuals with a number of victories lower than 5, because this number is 0 if they have won the five battles.

[4] `www.javassist.org`

		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

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

		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 3. Average results obtained from each configuration versus Exp-Genebot. Each one has been tested 30 times.

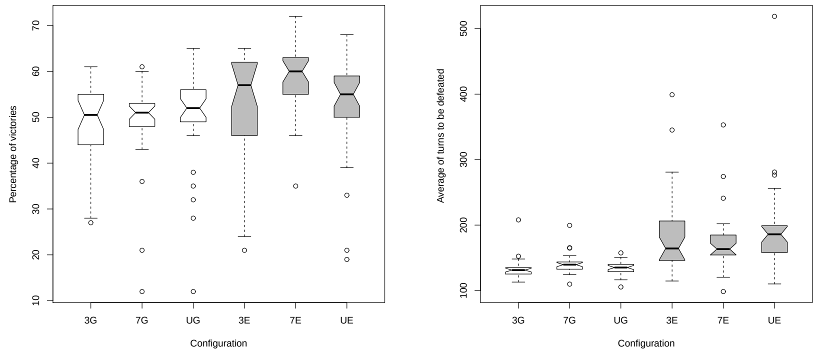
As can be seen, the average population fitness versus Genebot is nearest to the optimum than versus Exp-Genebot, even with the lowest depth. Highest performance in the population is also with the depth of 3 levels. On the contrary, confronting with Exp-Genebot the configuration with unlimited depth achieves better results. This make sense as more decisions should be taken because the enemy can be different in each map.

In the second experiment, we have confronted the 30 bots obtained in each configuration again with Genebot and Exp-Genebot, but in the 100 maps provided by Google. This experiment has been used to validate if the obtained individuals of the proposed method can be competitive in terms of quality in maps not used for evaluation. Results are shown in Table 4 and boxplots in Figure 3. It can be seen that in average, the bots produced by the proposed algorithm perform equal or better than the best obtained by the previous authors. Note that, even obtaining individuals with maximum fitness (5 victories) that have been kept in the population several generations (as presented before in Tables 2 and 3) cannot be representative of a extremely good bot in a wider set of maps that have not been used for training. As the distributions are not normalized, a Kruskal-Wallis test has been used, obtaining significant differences in turns for the experiment versus Genebot (p-value = 0.0028) and victories in Exp-genebot (p-value = 0.02681). Therefore, there are differences using a maximum depth in the generation of bots. In both configurations, the trees created with 7 levels of depth as maximum have obtained the better results.

To explain why results versus Genebot (a weaker bot than Exp-Genebot) are slightly worse than versus Exp-Genebot, even when the best individuals produced by the GP have higher fitness, it is necessary to analyse how the best individual and the population are being evolved. Figure 4 shows that best individual using Genebot reaches the optimal before Exp-Genebot, and also the average population converges quicker. This could lead to over-specialization: the generated bots are over-trained to win in the five maps. This is due because these individuals are being re-evaluated, and therefore, they are still changing after they have reached the optimal.

Configuration	*Average maps won*	*Average turns*
Versus Genebot		
Depth 3	47.033 ± 10.001	133.371 ± 16.34
Depth 7	48.9 ± 10.21	**141.386** ± 15.54
Unlimited Depth	50.23 ± 11.40	133.916 ± 10.55
Versus Exp-Genebot		
Depth 3	52.367 ± 13.39	191.051 ± 67.79
Depth 7	**58.867** ± 7.35	174.694± 47.50
Unlimited Depth	52.3 ± 11.57	197.492 ± 72.30

Table 4. Results confronting the 30 best bots attained from each configuration in the 100 maps each.



(a) Victories (b) Turns

Fig. 3. Average of executing the 30 best bots in each configuration (3, 7 and U) versus Genebot (G) and Exp-Genebot (E).

6 Conclusions

This work presents a Genetic Programming algorithm that generates agents for playing the Planet Wars game. A number of possible actions to perform and decision variables have been presented. Two competitive bots available in the literature (Genebot and Exp-Genebot) have been used to calculate the fitness of the generated individuals. These two bots were the best obtained from several runs and the behaviour to be optimized was extracted from human expertise. Three different maximum depth for the trees have been used: 3, 7 and unlimited. Results show that the best individuals outperform these agents during the evolution in all configurations. These individuals have been tested against a larger set of maps not previously used during the evolution, obtaining equivalent or better results than Genebot and Exp-Genebot.

In future work, other rules will be added to the proposed algorithm (for example, the ones that analyse the map, as the Exp-Genebot does) and different enemies will be used. Other games used in the area of computational intelligence in videogames, such as Unreal™ or Super Mario™ will be tested.

Acknowledgements

This work has been supported in part by FPU research grant AP2009-2942 and projects SIPESCA (G-GI3000/IDIF, under Programa Operativo FEDER de Andalucía 2007-2013), EvOrq (TIC-3903), CANUBE (CEI2013-P-14) and ANYSELF (TIN2011-28627-C04-02).

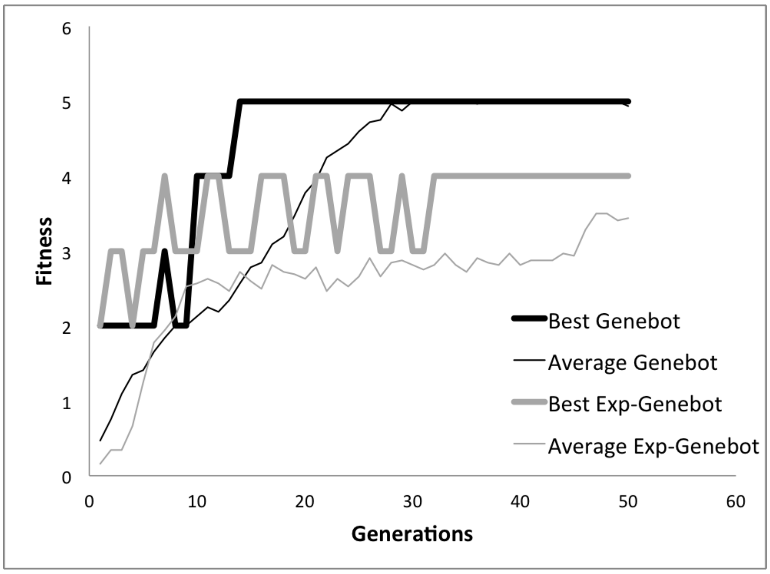


Fig. 4. Evolution of the best individual and the average population during one run for depth 7 versus Genebot and Exp-Genebot.

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
16. García-Sánchez, P., González, J., Castillo, P.A., Arenas, M.G., Guervós, J.J.M.: Service oriented evolutionary algorithms. Soft Comput. **17**(6) (2013) 1059–1075