Designing Competitive Bots for a Real Time Strategy Game using Genetic Programming

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

Universidad de Granada

June 24, 2014

Index

Introduction

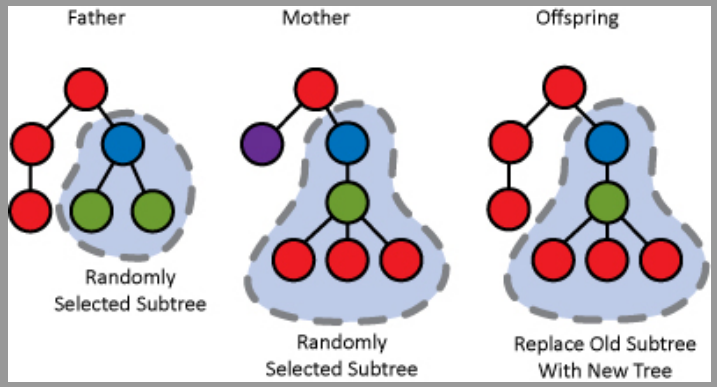
RTS

Real-Time Strategy games (RTS-games)

are a sub-genre of strategy-based video- games in which the contenders struggle to control a set of **resources**, **units** and structures that are distributed in a playing arena. A proper control and a sound strategy and tactics for handling these units is essential for winning the game, which happens after the game objective has been fulfilled, normally eliminating all enemy units, but sometimes also when certain points or game objectives have been reached.

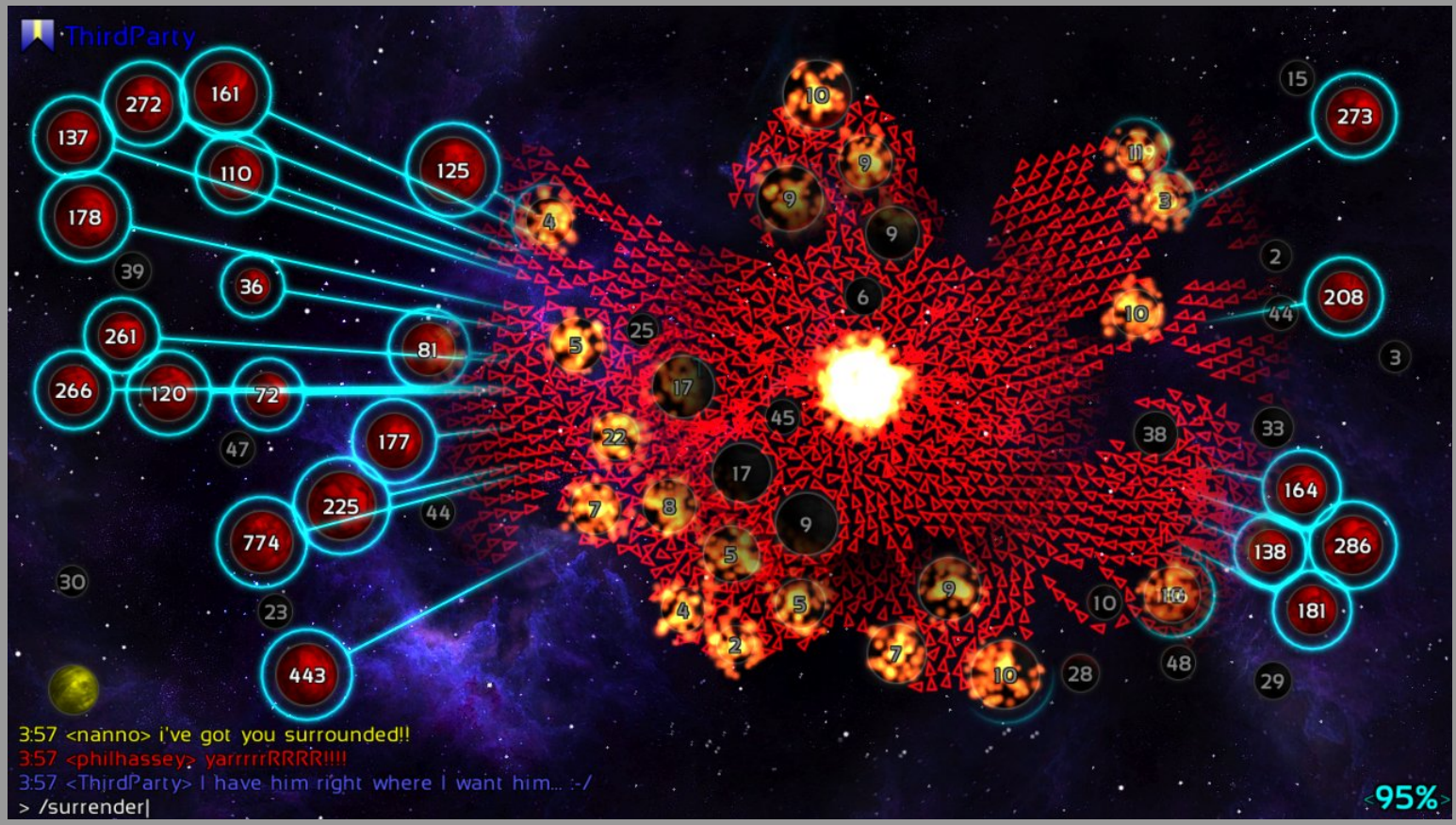
Genetic Programming (GP)

is a kind of **Evolutionary Algorithm** (EA) , that is, a probabilistic search and optimization algorithms gleaned from the model of **darwinistic evolution**, based on the idea that in nature structures un- dergo adaptation. EAs work on a population of possible solutions (individuals) for the target problem and use a selection method that favours better solutions and a set of operators that act upon the selected solutions.



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

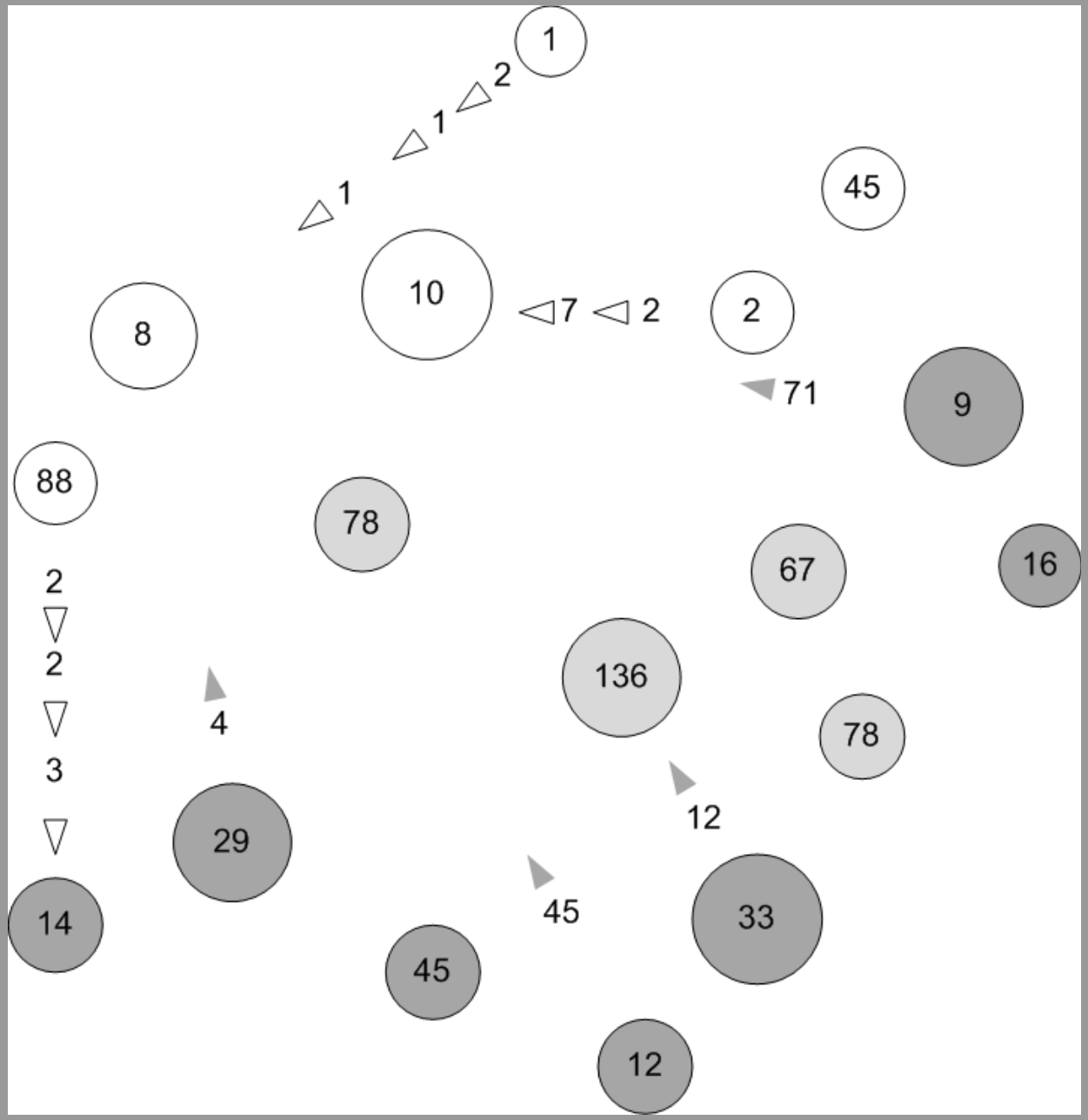
Planet Wars



Designing Competitive Bots for a Real Time Strategy Game using Genetic Programming

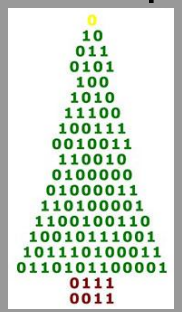
A.Fernández-Ares, P.García-Sánchez, A.M.Mora, P.A.Castillo, and J.J.Merelo

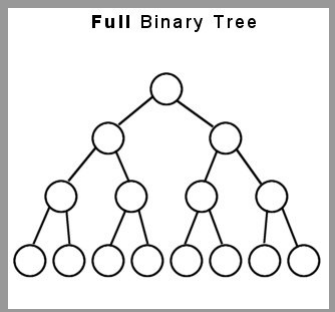
Planet Wars



GPBot

Evolves a set of rules which a **Decision Tree**.
During the evolution, every individual in the population (a tree) must be **evaluated**. To do so, the tree is set as the behavioural engine of an agent, which is then placed in a map against a rival in a Planet Wars match. Depending on the obtained results, the agent (i.e. the individual) gets a **fitness value**, that will be considered in the evolutionary process as a measure of its validity.
Thus, during the match the tree will be used (by the bot) in order to select the best strategy at every moment, i.e. for every planet a target will be selected along with the number of ships to send from one the other.





GPBot

Decisions & Actions

The used Decision Trees are binary trees of expressions composed 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 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).

Fitness Functions

Based in Victories

In this approach, 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 as better. The maximum fitness in this work is, therefore, 5 victories and 0 turns.

```
A, B ∈ Population
if A.victories = B.victories then
  if A.turns >= B.turns then
    A is better than B
  else
    B is better than A
  end if
else
  if A.victories > B.victories then
    A is better than B
  else
    B is better than A
  end if
end if
```

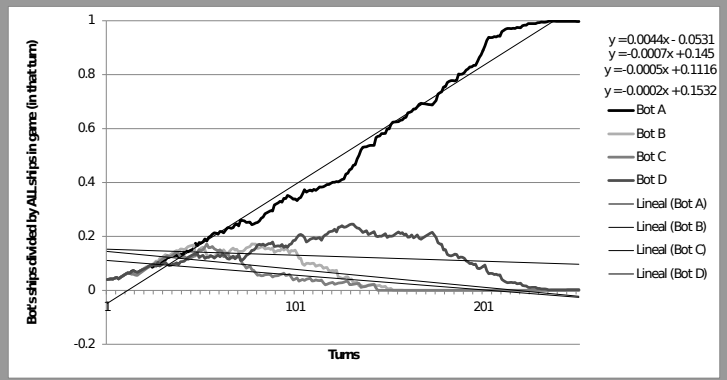
Fitness Functions

Based in Slope

In this case, a square regression analysis is computed in order to transform the cloud of points into a simple line. The line is represented as $y = \alpha \times x + \beta$, where α and β are calculated as shown in Equations 1 and 2, computing a least squares regression. For every bot in the simulation we calculate α and (*slope*). This *slope* is the fitness of every bot for that simulation.

$$\alpha = \frac{\sum_{i=1}^{n}(X_i - \bar{X}_i)(Y_i - \bar{Y}_i)}{\sum_{i=1}^{n}(X_i - \bar{X}_i)^2} \quad (1)$$

$$\beta = \bar{Y} - \alpha\bar{X} \quad (2)$$

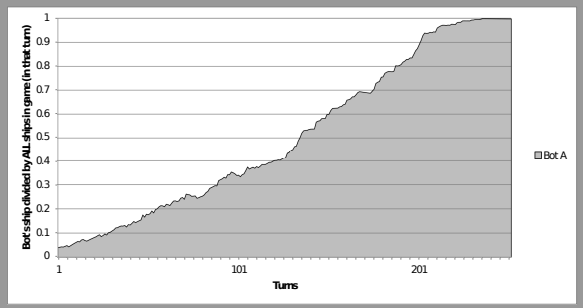


Fitness Functions

Based in Area

In this function, the integral of the curve of the bot's live-line is used for calculating the area that is 'covered' by the fitness cloud of points (see Equation 3). This *area* is normalized considering the number of turns, and thus it represents the average percentage of ships during the battle for each player.

$$area = \frac{\int_0^t \%ships(x)dx}{t} \tag{3}$$



Experimental Setup

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	7
Runs per configuration	30
Evaluation	Playing versus GeneBot
Maps	76,69,7,11,26

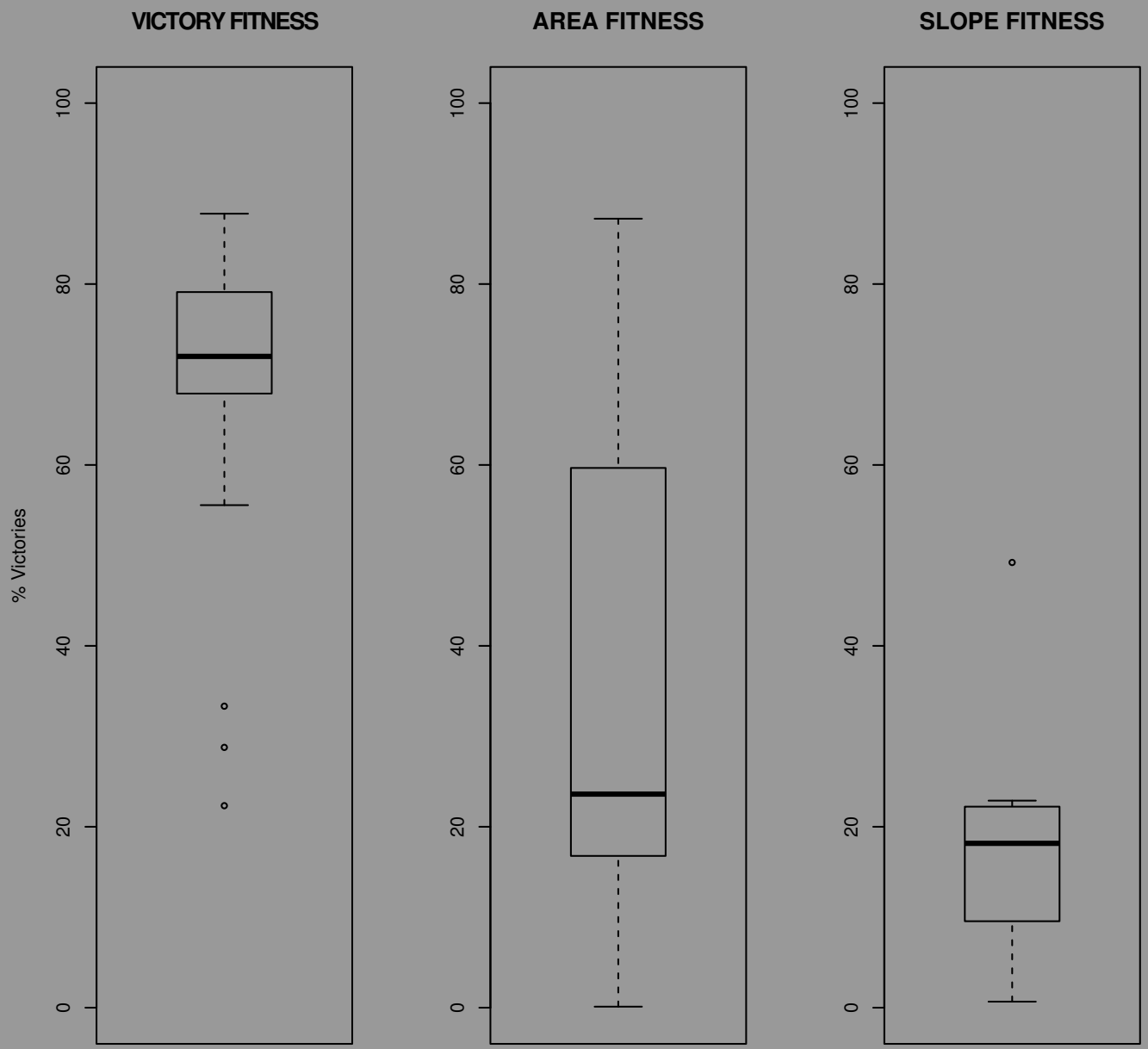
Results

Average results for each approach at end of the runs (20 executions)

	Avg best fitness	Avg population fitness
Victory	4.761 ± 0.624	4.345 ± 0.78
Slope	2.296 ± 0.486	2.103 ± 0.434
Area	2.838 ± 1.198	2.347 ± 0.949

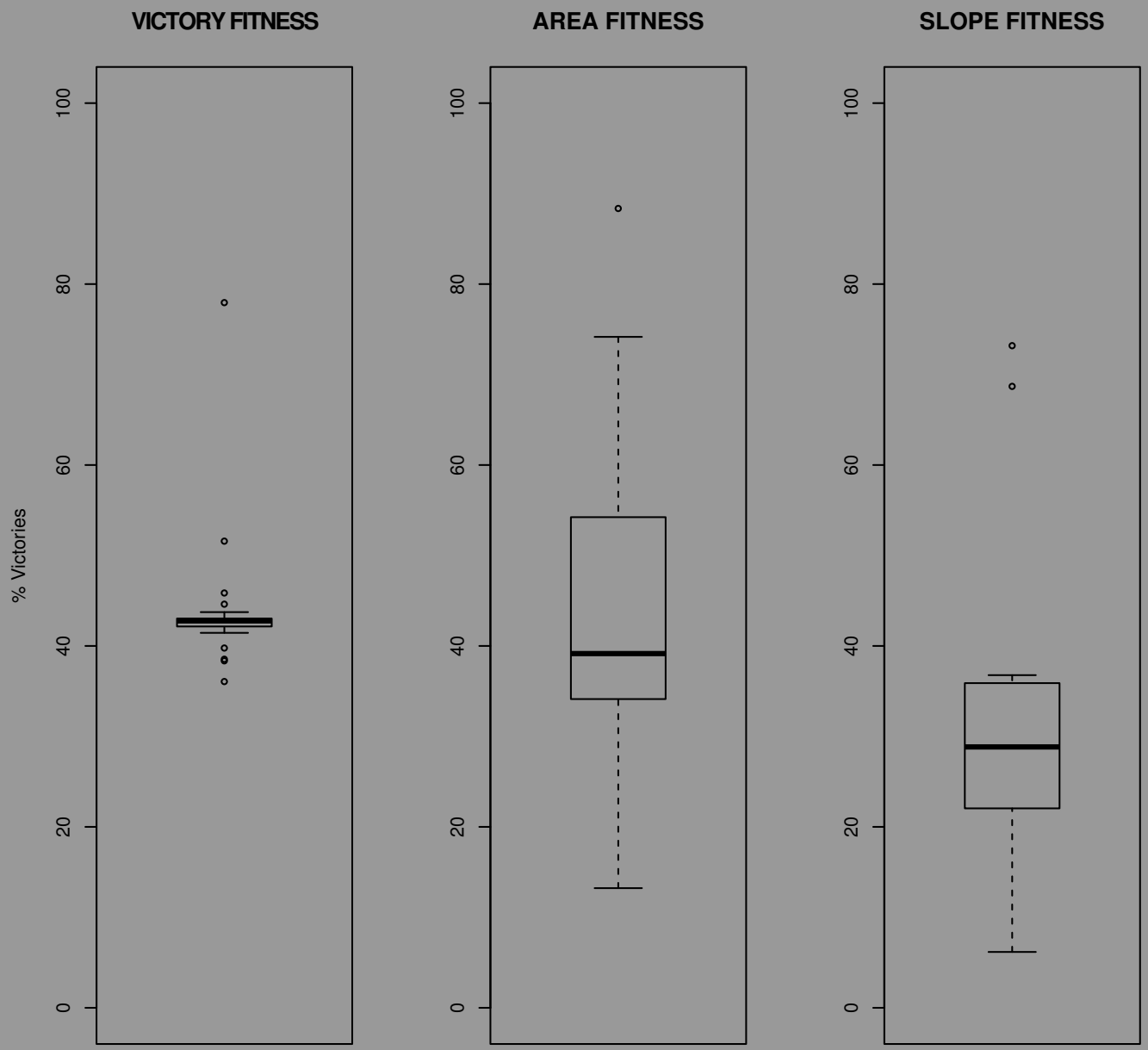
Results

Average percentaje victories by each approach VS Genebot (9 executions)



Results

Average percentaje victories of best bot by method vs the Rest



Conclusions

Genetic Programming

can create competitive bots for RTS games.

Fitness

Study the behaviour of different fitness functions, as they can affect directly to the creation of these bots. Three different fitness functions have been compared to generate bots for the Planet Wars game.

Enemies

A competitive bot available in the literature (GeneBot) has been used to evaluate the generated individuals (fighting against it) and have been tested versus others generated individuals.

Future Word

- Add other rules to the proposed algorithm (for example, rules to analyse the map).
- More competitive enemies.
- Implemente and teste in more complex RTS games, such as Starcraft, or even in different videogames like Unreal TM or Super Mario TM.

- Coevolutionary Genetic Programming

Designing Competitive Bots for a Real Time Strategy Game using Genetic Programming

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