Evolving Teams of Cooperating Agents for Real-Time Strategy Game

Paweł Lichocki[1], Krzysztof Krawiec[2], and Wojciech Jaśkowski[2]

[1] Poznan Supercomputing and Networking Center, Poznań, Poland
[2] Institute of Computing Science, Poznan University of Technology, Poznań, Poland
lichocki@man.poznan.pl, {kkrawiec,wjaskowski}@cs.put.poznan.pl

Abstract. We apply gene expression programing to evolve a player for a real-time strategy (RTS) video game. The paper describes the game, evolutionary encoding of strategies and the technical implementation of experimental framework. In the experimental part, we compare two setups that differ with respect to the used approach of task decomposition. One of the setups turns out to be able to evolve an effective strategy, while the other leads to more sophisticated yet inferior solutions. We discuss both the quantitative results and the behavioral patterns observed in the evolved strategies.

1 Introduction and Related Work

Among the genres of computer games, the popularity of real-time strategy games (RTS) increased significantly in recent years. As the acronym suggests, an important feature of an RTS game is the lack of turns: the players issue moves at an arbitrary pace. This makes timing essential, and requires the players to feature a mixture of intelligence and reflection. However, what makes RTS games distinctive is not their real-time mode of operation (which they share with, e.g., the first-person shooter games), but the strategic and tactical character. Rather than impersonating a specific character in the game, each player operates a set of *units* in a virtual world. The units are semi-autonomous: they obey commands issued by a player, but otherwise operate autonomously, carrying out some low-level tasks (like moving to a specific location, attacking an enemy unit, etc.). The game's world typically hosts also various types of resources that have to be managed. As these features are characteristic for military operations, RTS is typically considered as a subcategory of *wargames*. Contemporary iconic representatives of this genre include *Starcraft* and *Warcraft*.

Some RTS games offer software interfaces that enable substituting the human player with an encoded strategy. There are gamers who specialize in handcoding such strategies, known as 'AIs' or 'bots'. Despite this fact and despite the growing interest of computational intelligence researchers in the domain of games as a whole [8,10,4,1,11], attempts to make an RTS game a playground for computational intelligence surfaced only recently [2,12]. The primary reason for the low interest of computational intelligence community in the RTS realm is probably

M. Giacobini et al. (Eds.): EvoWorkshops 2009, LNCS 5484, pp. 333–342, 2009.

the inherent complexity of such games, resulting from the abundance of actors, heterogeneity of units, complexity of the environment and its dynamic character. These factors make it difficult to draw sound conclusions concerning the utility of a particular approach, parameter setting, or strategy encoding.

Trying to avoid the overwhelming complexity of off-shelf RTS games, we approach a task that is scaled-down yet preserves the essential RTS features. In particular, we verify the utility of gene expression programming as a tool for evolving a strategy for a relatively simple single-player game of gathering resources described in Section 2.

2 The Gathering Resources Game

The game of gathering resources has been originally proposed by Buro and defined within the Open Real-Time Strategy (ORTS) [3], an open-source scaled-down environment for studying algorithms for RTS games. The gathering resources game is one-person, perfect information, non-deterministic RTS game where the player controls a homogeneous set of 20 workers that operate in a discrete 2D world of 32×32 tiles (see Fig. 1), with each tile composed of 16×16 tilepoints. This results in a discrete grid of 512×512 points that may be occupied by workers, discrete resources called *minerals*, and other objects. The goal of the game is to maximize the amount of gathered minerals within the available simulation time.

The initial position of a worker is close to the *control center* (*base*). The worker should approach minerals, which are grouped into several patches, mine up to 10 of them (this is worker's maximum 'capacity'), and bring the collected minerals back to the base. This task is non-trivial due to the presence of static obstacles (hills), dynamic obstacles (randomly moving 'sheep' and other workers), and the

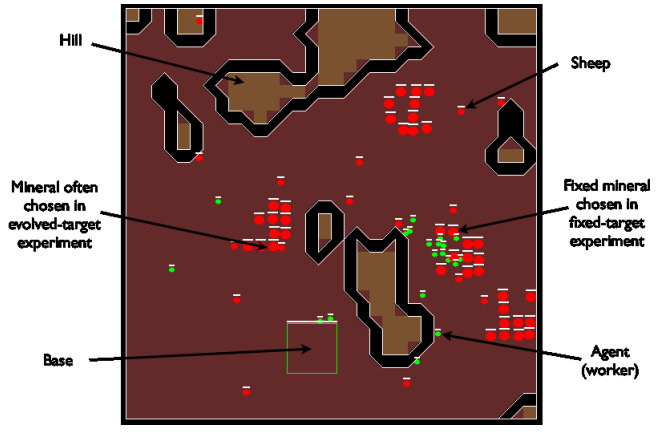


Fig. 1. A board of the gathering resources game used in the experiments

constraint that one mineral cannot be mined by more than four workers at a time. Thus, the two elementary skills required to attain a reasonable score are obstacle avoidance and collaborative pathfinding.

The player exerts control over each worker by means of four possible actions:

- *move*(x,y) – start moving towards location (x,y) (with a predefined speed),
- *stop*() – stop moving or mining (depending on the current activity),
- *mine*() – start mining minerals (effective in the vicinity of a mineral patch),
- *drop*() – drop all minerals (effective in the vicinity of the base).

The pace of the game is determined by the speed of worker movement (4 tile-points per 1 simulation frame) and mining (1 mineral per 4 simulation frames). When played by a human, the complete game lasts for 10 minutes at 8 simulation frames per second (totalling 4800 frames), with the player allowed to issue every unit 1 command per simulation frame.

3 The Approach and Setup

3.1 Worker Control and Strategy Encoding

We implement the model of *homogenous agents*, meaning that each worker implements the same behavior, which at the top level may be viewed as a finite-state machine. To ease the emergence of well-performing solutions, we manually decompose the task into two components and encode them in separate expression trees hosted by one individual that encodes the complete strategy. The first tree, called *commander* in following, is responsible for selecting a mineral and is queried when a particular worker is free. Once a particular mineral is selected by the commander as a *target* for the worker, the control over worker is taken over by the second tree, *navigator,* which is intended to guide the worker towards the target and may refer to it via one of available terminals. The state of approaching the mineral lasts until the worker reaches it, when it switches to the 'mineral mining' state for 40 simulation frames (mining speed $\times$ worker's capacity). After mining is over, the control over the worker is passed back to the navigator, this time substituting the base as the navigation target. When reaching the base, the worker drops the minerals and becomes ready for another round. This working cycle is fixed; only commander and navigator undergo evolution.

When designing the representation (the set of terminals and functions) we had to handle the inevitable trade-off between the compactness of representation (and the resulting cardinality of the search space) and its expression power. In order to keep the representation compact, commanders and navigators use disjoint, task-oriented data types. The former uses only scalars (real numbers) and the latter only vectors (tuples of real numbers).

Table 1 shows the set of functions and terminals used by commanders. Each time an agent has to choose which mineral to gather, the commander tree built from these components is applied to every mineral on the board to compute its 'utility'. The mineral with the highest utility is assigned to the worker; this assignment remains unchanged until the worker reaches the prescribed mineral.

Table 1. Functions and terminals (arity=0) used by the commander tree. The notation 'a ? b : c' means 'if a is true return b ohterwise return c'.

Notation	Arity	Result
SOPP(S)	1	opposite value
SINV(S)	1	inverted value
SABS(S)	1	absolute value
SSIG(S)	1	sigmoid function
SADD(S1,S2)	2	addition
SSUB(S1,S2)	2	substraction
SMUL(S1,S2)	2	multiplication
SMAX(S1,S2)	2	maximum
SMIM(S1,S2)	2	minimum
SLEN(SR,S1,S2)	3	SABS(SR-S1) < SABS(SR-S2) ? S1 : S2
SIF1(SR1,SR2,S1,S2)	4	SR1 < SR2 ? S1 : S2
SIF2(SR1,SR2,S1,S2)	4	SR1 >= SR2 ? S1 : S2
S_ONE	0	1.0
S_DISTANCE	0	distance to the mineral
S_MINERAL_INDEX	0	current mineral index
S_WORKER_INDEX	0	current agent index
S_LAST_VALUE	0	previously computed value
S_AVG_VALUE	0	the average value of all minerals
S_MAX_VALUE	0	the maximum value of all minerals
S_REFERENCES	0	number of workers going to the mineral
S_VISITS	0	number of workers' visits
S_TASKS	0	S_REFERENCES + S_VISITS

Table 2 presents the set of functions and the set of terminals used for encoding of navigators. In each simulation frame the ORTS simulator queries the navigator to obtain a temporary movement target for each worker. The navigator tree evaluates to a vector represented as a tuple of two real numbers: the length and the angle. The vector is then transformed into board coordinates and an appropriate $move(x,y)$ command is performed by the worker.

Both commander and navigator are encoded in a common linear chromosome using Gene Expression Programming (GEP) [5]. In canonical GEP, an expression tree is encoded as a sequence of numbers subdivided into *head* and *tail*, with the head consisting of both function and terminal symbols, and the tail containing terminal symbols only (see [6] for more details). The length of the head determines the maximal size of the expression tree. In addition to GEP's head and tail, we introduce an extra part called *forehead* that contains only function symbols. This allows us to control the minimal number of nodes in an expression tree, which might be helpful in forcing the evolution to focus on complex solutions. Table 3 presents the complete chromosome setup.

Defining individual's fitness as the the number of collected minerals proved ineffective in one of the preliminary runs. We noticed that evolution may stagnate

Table 2. Functions and terminals (arity=0) used by the navigator tree

Notation	Arity	Result
VOPP(V)	1	opposite vector
VINV(V)	1	vector with inverted length
VRIG(V)	1	perpendicular vector (turn counter clockwise)
VLEF(V)	1	perpendicular vector (turn clockwise)
VSHO(V)	1	normalization to a short distance
VMID(V)	1	normalization to a middle distance
VLON(V)	1	normalization to a long distance
VADD(V1,V2)	2	addition
VSUB(V1,V2)	2	subtraction
VMAX(V1,V2)	2	longer vector
VMIN(V1,V2)	2	shorter vector
VROT(VR,V)	2	rotates V towards VR
VMUL(VR,V)	2	scales V accordingly to VR
VCRD(VR,V)	2	represents V treating VR as a vector on OX axis
VIFZ(VR,V1,V2)	3	length(VR)=0 ? V1 : V2
VLEN(VR,V1,V2)	3	chooses vector with length "closer" to the VR
VDIR(VR,V1,V2)	3	chooses vector with direction "closer" to the VR
V_CLOSEST_POSITION	0	vector to the closest unit
V_CLOSEST_MOVE	0	move vector of the closest unit
V_ANTISRC	0	inverted vector to the source
V_DST	0	destination vector
V_LAST	0	vector to the previous position
V_ORDER	0	previous move vector
V_ANTIHILL	0	vector that "points" away from the closest hill patch
V_ANTIUNIT	0	vector that "points" away from the local group of units

due to the discrete nature of such straightforward fitness measure. To provide an extra incentive for individuals, we defined fitness as

$$fitness = 5000 \times \left[minerals + \frac{1}{n} \sum_{i=1}^{n} l(1 - \frac{||worker_i, target_i||}{||source_i, target_i||}) \right], \qquad (1)$$

where n is the number of workers, $source_i$ is the starting point of i^{th} worker, $target_i$ is the current target of i^{th} worker (one of the minerals or the base, depending on the current phase of the worker's cycle), $||\,||$ stands for Euclidean distance, and $l()$ is the logistic function ($l(x) = 1/(1+\exp(-x))$). Effectively, such a definition introduces a second objective that rewards a strategy for workers' proximity to their current targets at the end of simulation. The above formula combines these objectives in a lexicographic manner, i.e., given two strategies that collect the same number of minerals, the one that has its workers closer to their current targets is considered better. Such a definition is helpful in the initial stages of an evolutionary run, when the individuals have not discovered yet that collecting minerals is a beneficial behavior.

Table 3. Chromosome structure (numbers indicate lengths of chromosome parts)

Navigator			Commander		
Forehead (5)	Head (25)	Tail (61)	Forehead (5)	Head (25)	Tail (121)
Functions	Func+term	Terminals	Functions	Func+term	Terminals

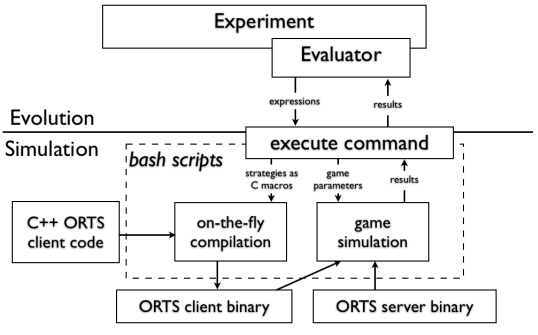


Fig. 2. Experiment framework scheme

It is worth mentioning that the technical realization of the experiment was challenging. We followed and modified the previously verified technical setup for performing evolutionary experiments from [7]. We used ECJ ver. 18 [9] for running the evolutionary experiment, the ORTS framework for real-time game simulations, a homebrew parser for mapping GEP chromosomes to trees, and advanced bash-scripting to "tie the loose ends". Figure 2 shows the major software components and data flow within an experiment. ECJ standard classes are responsible for the main evolutionary loop. In the evaluation phase, individual's genotype is expressed into the C code and passed to the script, which performs on-the-fly compilation of those lines along with ORTS client code (which is constant and responsible for connecting with ORTS server). When the compilation finishes, the script runs the simulation – it starts the ORTS server and than the newly build ORTS client. When the simulation finishes, the result of the game is returned to the ECJ evaluator.

3.2 Evolutionary Parameters

Both experiments discussed in the following used the same parameter settings: population size 200, 150 generations, tournament selection with tournament size 5, and simple elitism (the best-of-generation individual is unconditionally copied to the next population). Though GEP introduces many innovative genetic operators (e.g., gene recombination, RIS-transposition), for simplicity we use the standard one-point crossover applied with probability 0.9 and one-point mutation applied with probability 0.1.

We expect that that due to the large numbers of functions and terminals the search space of this problem is very large. To avoid expanding it even more, we restrain from using random constants.

The rather moderate population size and number of generations is an inevitable consequence of the time-consuming evaluation process that requires simulating the entire game. Even after shortening the game to 1000 simulation frames (compared to 4800 in the original ORTS game specification) and speeding up the simulation rate to 160 frames per second (compared to default 8 in ORTS), a 2.4GHz CPU needed more than 40 minutes to evaluate just one population, so that it took 9 days to finish the computations.

4 The Results

The major objective of the experiment was to test our approach with respect to the ability of evolving a successful cooperation between the commander and the navigator. To this aim, we performed two evolutionary runs. In the first one, the *fixed-target experiment*, we have fixed the commander to always assign the same, arbitrary chosen mineral to all workers. Thus, in this scenario, only the navigator was evolved. In the second one, called the *evolving-target experiment*, evolution of both the commander and the navigator took place.

Figure 3 shows the mean fitness graph for both experiments. From the very beginning of the runs, the evolving-target approach improves the performance of the average individual at notably better rate than the fixed-target run, which seems to stagnate already around the 40^{th} generation. However, without more insight into the evolved solutions, it is difficult to determine the underlying cause for this difference.

In following we qualitatively analyze the behavioral changes in strategies encoded by successive best-of-generation individuals from both runs. Some of the emerging behaviours prove that the commander-navigator tandem was adapting to the game characteristic.

In the **fixed-target experiment** all agents aim at gathering mineral from the same static location and only the navigator was evolved. The target mineral was

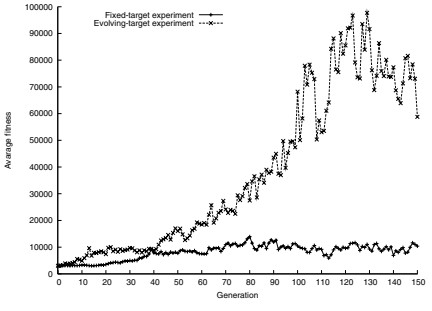
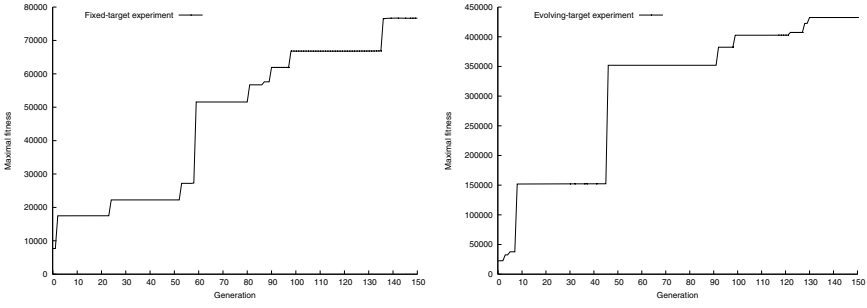


Fig. 3. Average fitness



(a) fixed-target (b) evolved-target

Fig. 4. Best-of-generation fitness graphs

relatively far from the base and there was a wall (a static obstacle) between them. We observed qualitative changes in following generations (the best-of-generation fitness plot (Fig. 4a) reflects well these major 'evolutionary leaps'):

(2) Small-circle. Agents move along small circular paths using the closest unit as the reference (the center of the circle). This is the first attempt of the agents to avoid each other (but not the hills).

(12) Big-circle bouncing. Agents move on large circular paths, but usually do not block each other, as two circles intersect at most only two points. Simultaneously, evolution discovers the first method of walking around the hills – agents "bounce off" the hills by rotating the move vector.

(58) Shaking. The previous behaviours did not prevent agents from blocking when they crowd (which inevitably happens when gathering the same mineral). In this generation agents learned to move towards a common destination by incorporating small random steps into various directions. It looks similar to Brown's movements of chemical particles, however oriented towards a chosen point. Unfortunately, this trick does not help yet walking around the hills.

(59) Upgraded shaking. The agents adapted the shaking move so that it successfully helps avoiding the hills. Evolution discovered also entirely different path to reach the target mineral, which turned out to be far superior to the old one.

(137) Shaking on demand. When there are no other units in proximity agents tend to favor movement on straight or curved lines. "Shaking" is still used to avoid obstacles or other units.

In the **evolving-target experiment** both the navigator and the commander were evolved. The results are far better in terms of fitness, but the observed behaviors are less sophisticated. Already the initial generation turned out to contain a commander that chooses the closest mineral, and this approach dominated the remaining part of evolution (except for the 5th generation when the best individual distributes agents over different minerals).

With the commander always choosing the closest mineral, the task of the navigator becomes less sophisticated than in the fixed-target experiment, because (i) the shortest path to the target does not collide with the hills (see Fig. 1), and (ii) the short distance to be traversed does not require an elaborate mechanism for handling worker collisions. Thus, the major remaining problem to solve was the ability to move 'in herds'. Evolution discovered the *Small-circle* move in the 5th generation, the *shake* move in the 17th generation, and significantly increased the intensity of shaking in the 46th generation (Fig. 4b). The traits acquired later were minor and did not lead to any meaningful behaviour changes. Unfortunatelly, the agents did not learn how to walk around the base and some of them remained blocked for entire simulation.

5 Conclusions and Discussion

In general terms, the result obtained in the reported experiments may be considered as positive. Given a vocabulary of quite low-level functions and terminals, and with a little help in terms of experiment design (augmented fitness definition and task decomposition), our approach was able to evolve effective strategies that exhibit most of the functionalities requires to solve the task. Let us emphasize that, apart from the assignment of workers to minerals and the ability of perceiving the board state, the evolved strategies are stateless – in a short run, the agents work exclusively using the stimulus-reaction principle. There is no short-term memory to help agents in carrying out such actions like, e.g., obstacle avoidance.

We have to admit that, considering the behavioral aspect, the evolving-target experiment produced rather disappointing outcome. Our model of artificial evolution proved very effective at cutting off the corners and, at the same time, rather ineffective at going beyond an easy-to-come-up-with yet globally not optimal solution. A lesson learned is that behavioral sophistication does not always go hand in hand with the performance.

There are numerous ways in which the proposed approach could be extended or varied. For instance, the number of functions and terminals used by commanders and navigators is large (see Tables 1 and 2). On one hand, this enables the evolving trees to express complex concepts, on the other, however, lowers the *a priori* probability of picking a particular function or terminal. We look forward for methods that address this difficult trade-off.

The fitness function defined in Formula (1) aggregates two undelying objectives: the number of collected minerals and worker's distance from the assigned target. The particular form of aggregation is quite arbitrary. In theory, a more elegant way would be to approach the problem with multi-objective formulation. However, our past experience with multiobjective evolutionary compuation indicates that it inevitably implies weaker selective pressure and may negatively impact the convergence of the evolutionary run. In the light of very expensive evaluation cost in our study, such approach seemed risky. For the same reason of high computational demand, in seems reasonable to consider distributed

computing. This would also allow us to run the experiments several times to obtain a more reliable estimates of its performance.

Finally, this large-scale study could be possibly used as a yardstick for comparing GEP to GP.

Acknowledgement

This research has been supported by grant # N N519 3505 33 and EC grant QosCosGrid IST FP6 STREP 033883.

References

1. Azaria, Y., Sipper, M.: GP-gammon: Genetically programming backgammon players. Genetic Programming and Evolvable Machines 6(3), 283–300 (2005); Published online: 12 August 2005
2. Buro, M.: Real-time strategy games: A new AI research challenge. In: Proceedings of the International Joint Conference on AI 2003, Acapulco, Mexico (2003)
3. Buro, M., Furtak, T.: ORTS open real time strategy framework (2005), `http://www.cs.ualberta.ca/~mburo/orts/`
4. Corno, F., Sanchez, E., Squillero, G.: On the evolution of corewar warriors. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Portland, Oregon, pp. 133–138. IEEE Press, Los Alamitos (2004)
5. Ferreira, C.: Gene expression programming: a new adaptive algorithm for solving problems. Complex Systems 13, 87 (2001)
6. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. Studies in Computational Intelligence. Springer, New York (2006)
7. Lichocki, P.: Evolving players for a real-time strategy game using gene expression programming. Master's thesis, Poznan Universtity of Technology (2008)
8. Luke, S.: Genetic programming produced competitive soccer softbot teams for Robocup 1997. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, pp. 214–222. Morgan Kaufmann, San Francisco (1998)
9. Luke, S.: ECJ evolutionary computation system (2002), `http://cs.gmu.edu/eclab/projects/ecj/`
10. Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. Machine Learning 32(3), 225–240 (1998)
11. Sipper, M.: Attaining human–competitive game playing with genetic programming. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, p. 13. Springer, Heidelberg (2006)
12. Stanley, K., Bryant, B., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation 9(6), 653–668 (2005)