Evolving swarm intelligence for task allocation in a real time strategy game

Anderson R. Tavares, Hector Azpúrua, Luiz Chaimowicz
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais

Abstract

Real time strategy games are complex scenarios where multiple agents must be coordinated in a dynamic, partially observable environment. In this work, we model the coordination of these agents as a task allocation problem, in which specific tasks are given to the agents that are more suited to execute them. We employ a task allocation algorithm based on swarm intelligence and adjust its parameters using a genetic algorithm. To evaluate this approach, we implement this coordination mechanism in the AI of a popular video game: *StarCraft: BroodWar*. Experiment results show that the genetic algorithm enhances performance of the task allocation algorithm. Besides, performance of the proposed approach in matches against *StarCraft's* native AI is comparable to that of a tournament-level software-controlled player for *StarCraft*.

Keywords: Task allocation, Evolutionary algorithms, Real-time strategy

Author's Contact:

{anderson,hector.azpurua,chaimo}@dcc.ufmg.br

1 Introduction

Real Time Strategy (RTS) games have become one of the most successful genres in the game industry. In these games, players have to manage a limited set of resources to achieve a particular goal. Normally, RTS games are played at a extremely fast pace and players have to deal simultaneously with several different objectives such as collect resources, construct bases, improve technology and battle against enemy armies [de Freitas Cunha and Chaimowicz 2010]. Due to their characteristics, RTS games have also become an excellent testbed for novel research in AI [Buro 2003]. Particularly, the coordination of multiple agents in RTS games is an interesting research topic and has several commonalities with other complex scenarios such as rescue operations in disaster situations [Kitano 2000] or cooperative robotics [Fierro et al. 2002].

Agent coordination in complex scenarios is one of the greatest challenges in multiagent intelligent systems and normally involves the optimized use of resources by different agents to accomplish a global goal. In general, complex scenarios can be considered as those in which there is a set of agents that must perform multiple tasks in a dynamic and partially observable environment, where existing tasks can disappear and new tasks can arrive. Thus, RTS games have arisen as environments to evaluate multiagent coordination in complex scenarios [Weber et al. 2011].

In complex scenarios, a natural way to organize the work between agents is to divide the goal into tasks. Therefore, task allocation becomes an important part of the coordination problem. In this work we propose an approach for task allocation in RTS games that uses a genetic algorithm to adjust the parameters of Swarm-GAP, a probabilistic, scalable algorithm for task allocation based on swarm intelligence [Ferreira et al. 2008]. The proposed approach is implemented on the *StarCraft: BroodWar* game (*StarCraft* for short).

Experiments in matches against *StarCraft's* native AI show that the proposed approach outperforms random task allocation and a configuration of Swarm-GAP parameters adjusted by hand. The performance of our approach is similar to that of a tournament-level *StarCraft* bot. The genetic algorithm was useful to refine the parameters that configure the behavior of the algorithm for task allocation.

The remainder of this paper is organized as follows: Section 2 introduces some basic concepts regarding task allocation, specifically the Extended Generalized Assignment Problem (E-GAP) and the algorithm Swarm-GAP. Section 3 discusses some related work both in the fields of task allocation in complex scenarios and AI for RTS games. Section 4 discusses the adoption of Swarm-GAP for task allocation in *StarCraft*, while Section 5 presents the genetic algorithm used to evolve Swarm-GAP parameters. Experiments with the proposed approach are presented in Section 6 while Section 7 brings the conclusion and directions for future work.

2 Task allocation

2.1 E-GAP

Task allocation is concerned with the assignment of tasks to agents in order to maximize a global metric of performance, usually related to the skills of the agents to perform each task. In dynamic environments, the task allocation problem can be modeled as an instance of E-GAP (Extended Generalized Assignment Problem) [Scerri et al. 2005]. E-GAP is a generalization of the GAP (Generalized Assignment Problem), which is NP-complete [Shmoys and Tardos 1993].

The E-GAP can be formalized as follows: let $\mathcal{J}$ be the set of tasks and $\mathcal{I}$ the set of agents. Each agent $i \in \mathcal{I}$ has r_i resources to perform tasks. Each task $j \in \mathcal{J}$ consumes c_{ij} resources of agent i, when it performs the task. Each agent i has a capability $k_{ij} \in [0, 1]$ to perform task j. Capability can be regarded as the skill of the agent to perform the task.

An allocation matrix $A_{|\mathcal{I}|\times|\mathcal{J}|}$ has its elements a_{ij} set to 1 if agent i performs task j, and 0 otherwise. In this model, only one agent can perform a given task instance.

In E-GAP, the total reward W is calculated as the sum of the rewards of the agents along t timesteps. In one timestep, the reward is calculated considering the capability of the agents to perform the tasks they were assigned. A delay cost d_j^t is applied as a penalty for not allocating task j in timestep t, as Eq. 1a illustrates. The calculation of rewards along the timesteps captures the dynamics of the environment. That is, the reward in a given timestep depends on the tasks and agents that exist in that timestep. Equation 1b determines that agents must allocate tasks within their resource limits and Eq. 1c determines that a task can be performed by only one agent. Thus, large tasks must be broken down into smaller tasks that can be performed by a single agent. E-GAP also considers task interdependence, but this aspect will not be investigated in this work.

$$W = \sum_t \sum_{i^t \in \mathcal{I}^t} \sum_{j^t \in \mathcal{J}^t} k_{ij} \times a_{ij}^t - \sum_t \sum_{j^t \in \mathcal{J}^t} (1 - a_{ij}^t) \times d_j^t \quad (1a)$$

$$\text{subject to:} \qquad \forall t \forall i^t \in \mathcal{I}^t, \sum_{j^t \in \mathcal{J}^t} c_{ij}^t \times a_{ij}^t \le r_i^t \quad (1b)$$

$$\text{and:} \qquad \forall t \forall j^t \in \mathcal{J}^t, \sum_{i^t \in \mathcal{I}^t} a_{ij}^t \le 1 \quad (1c)$$

2.2 Swarm-GAP

Swarm-GAP is an approximate algorithm for E-GAP, inspired by the labor division of social insects. In swarms, or colonies of social insects, in general there are hundreds or thousands of members that work without explicit coordination. From the aggregation of individual actions of colony members, complex behaviors emerge. One characteristic of swarms is the ability to respond to changes in the environment by adjusting the numbers of members performing each task.

Observations about swarm behaviors are the base of the model presented in [Theraulaz et al. 1998], where tasks have associated stimulus[1] and individuals have response thresholds for each task. Let $s_j \in [0,1]$ be the stimulus associated with task $j \in \mathcal{J}$ and $\theta_{ij} \in [0,1]$ be the response threshold of individual (agent) i to task j. The tendency, or probability, of individual i to engage in task j is given by $T_{ij} \in [0,1]$, calculated using Eq. 2.

$$T_{ij} = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} \tag{2}$$

In swarms, due to polymorphism, individuals may be more able to perform certain kinds of tasks. This characteristic is captured in Eq. 3, which determines the response threshold of individual i to task j according to its capability ($k_{ij} \in [0,1]$) to perform task j.

$$\theta_{ij} = 1 - k_{ij} \tag{3}$$

The goal of Swarm-GAP, according to [Ferreira et al. 2008], is to allow agents to individually decide which task they will engage in a simple and efficient way, minimizing computational effort and communication between agents. With Swarm-GAP, agents communicate via a token based protocol. When a given agent perceives new tasks, it creates a token with these tasks. The agent can receive tokens from other agents too. Either way, the holder of a token has the right to decide in which tasks of the token it will engage. The token with the remaining tasks is passed to a random agent that has not held the token before. This is formalized in Alg. 1, which is executed by each agent independently.

Algorithm 1 Swarm-GAP

```
When tasks are perceived: $token \leftarrow \{\text{perceived tasks}\}$
When message is received: $token \leftarrow \{\text{received token}\}$
for all task $j \in token$ do
    if $random() < T_{ij}$ and $r_i > c_{ij}$ then
        Engage in task $j$
        $token \leftarrow token \setminus \{j\}$
        $r_i \leftarrow r_i - c_{ij}$
    end if
end for
Send $token$ to random agent that didn't see the token before
```

Although the execution of Swarm-GAP is simple, a good performance requires a good adjustment of all parameters, i.e., we need to model the stimuli s_j for each task, the resources r_i that each agent has, as well as the capability k_{ij} and task cost c_{ij} for each agent and task. Finding a combination of these parameters that yields a good performance of Swarm-GAP can be very time-consuming specially in scenarios with several different types of agents and tasks.

3 Related work

3.1 Task allocation in complex scenarios

An approximate and decentralized algorithm for the task allocation problem in complex scenarios is LA-DCOP [Scerri et al. 2005], where the modeling of the task allocation problem as E-GAP is introduced. In LA-DCOP, agents communicate and achieve coordination through a token-based protocol. Agents perceive tasks on the environment and create a token containing the tasks. The token holder, based in a global threshold, decide which tasks it should engage. The token with the remaining tasks is communicated to other agents. In LA-DCOP, agents must allocate tasks in order to maximize the sum of their capabilities, respecting their resource limitations. This can be reduced to the binary knapsack problem (BKP), which is NP-complete. Therefore, the efficiency and efficacy of LA-DCOP depend on the method that solves the BKP. Each agent solves several instances of the BKP during the complete process of allocation.

Note that LA-DCOP and Swarm-GAP are very similar regarding the token-based protocol for coordination. The difference in both algorithms lies in the way the agents allocate tasks. In LA-DCOP, the process corresponds to solving an instance of the BKP whereas in Swarm-GAP, agents allocate tasks in a probabilistic fashion inspired by the division of labor in social insects.

The task allocation problem can also be modeled through the formalism of coalition formation. In this formalism, a coalition structure[2] is determined and coalitions of agents are assigned to tasks. An approach that uses this formalism is presented in [Khan et al. 2010]. The work presents a methodology to model the coalition formation problem as a Markov Decision Process (MDP). Initially, the generated MDP has an intractable state-action space. The authors present a method for parallel partition of the generated MDP. With this approach, efficient MDP algorithms can be applied. In fact, authors apply this methodology to problems with hundreds of agents and tasks. However, in [Khan et al. 2010], experiments are performed with homogeneous agents and they must deal with a single task type, i.e., simulated firefighting.

Branch-and-bound fast-max-sum (BnB FMS) is an anytime algorithm presented in [Macarthur et al. 2011] that advances the state-of-the-art regarding task allocation in large-scale scenarios. BnB FMS searches the coalition structure that maximizes a global utility, that considers the contribution of each agent in its coalition when it performs a set of tasks. The search space explored by the algorithm is pruned through the reduction of the number of tasks and coalitions that need to be evaluated. The pruning techniques keep the correctness and robustness of the algorithm when the environment is dynamic. Performed experiments showed that BnB FMS reaches a global utility 23% higher than previous state-of-the-art algorithms, with 31% less computing time and 25% less messages than other algorithms.

As mentioned, in this paper we use Swarm-GAP to perform task allocation, augmenting it with a genetic algorithm to tune its parameters. This is a novel approach, specially in a RTS game scenario, as discussed in the next section.

3.2 Artificial intelligence in RTS games

Developing intelligent systems for RTS games is a complex problem because in addition to the intrinsic constraints of the game, such as partial observation, there are two types of decision making: micro-management, responsible on single unit behavior, and macro-management, responsible for sequencing the construction of structures, resource management and strategy, which involves task allocation.

Probably these aspects make these type of games less researched as their turn-based games counterparts [Fernandez-Ares et al. 2011], despite the great number of know open challenges such as: adversarial real-time planning, decision making under uncertainty of partially observable domains, learning from experience or observation, opponent modeling, spatial and temporal reasoning, navigation, resource management and collaboration [Lara-Cabrera et al. 2013; Ontanon et al. 2013].

The most common techniques used to tackle these challenges are:

[1]Stimulus intensity may be associated with pheromone concentration, the number of encounters with other individuals performing the task or another characteristic that individuals can measure.

[2]A coalition structure is a partition of the set of agents. Each subset of the partition is a coalition.

- Change the current strategy in real time depending on the game environment [Spronck et al. 2004; Ludwig and Farley 2009]. This technique has the disadvantage that multiple strategies are commonly hard-coded (scripted) and it's known that hard-coded preconditions hardly will cover all possible events in the game. Another disadvantage of scripted AI systems are that there is no challenge for human players after the user has successfully learned how the script works [Falke II and Ross 2003];

- Use of Bayesian models. There are works implementing Bayesian models for individual unit control [Synnaeve and Bessière 2011b]. Their results outperform *StarCraft's* native AI and several other bots on unit micro-management. Authors also develop other works to predict how to act in the first minutes of the game [Synnaeve and Bessière 2011c] and to recognize plans [Synnaeve and Bessière 2011a] using Bayesian models;

- Potential fields, developed first as a navigation method to avoid obstacles in robotics [Khatib 1986], can be seen used in conjunction with a modified $A*$ algorithm for navigation in *StarCraft* game [Sandberg and Togelius 2011]. Their results have shown improvements over the navigation using only the $A*$ algorithm. Other works implement potential fields in conjunction with fuzzy measurements for micro-management in the game *Warcraft III* [Ng et al. 2011]. Their results have shown success on minimizing the score of the opponent on most cases;

- Evolutionary algorithms. This method is generally used to fine tune the parameters of other techniques. Works such as [Lin and Ting 2011] and [Rathe and Svendsen 2012] implement evolutionary algorithms to improve their performance but, in these works, the focus is on micro-management of units. Micro-management in this case consists of unit positioning, target selection and retreat during battles. In the present work the focus is on macro-management, which is related to task allocation (i.e., which tasks should be done), whereas micro-management is concerned with how the tasks should be done.

An example of evolutionary algorithm applied to evolve parameters of a RTS game player can be found in [Fernandez-Ares et al. 2011]. The authors define a rule-based player with adjustable parameters that are evolved with genetic algorithm. The approach is tested in a game called Planet Wars. In this game, a match takes place in a map containing several planets, each one with a number of space ships in it. The action of a player consists in sending space ships to other planets in order to conquer them. The player who conquers all enemy's planets is the winner. This scenario is simpler than the one studied here (see Section 4.1). In the scenario studied in the present work, the number of distincts actions (or tasks) that need to be performed is substantially higher.

4 Swarm-GAP in *StarCraft*

4.1 *StarCraft* game

StarCraft is a RTS game, whose domain presents several challenges that share similarities with the real world, because of the characteristics of the environment, enumerated below.

- The environment is continuous in space in time (or at least it is discretized in a reasonably thin granularity);

- Partial observability: a player can only access information within the visual range of his units and buildings;

- Dynamicity: due to the actions of several agents, the environment is always changing, thus demanding quick decisions;

- The decision process is sequential, meaning that actions taken now affect which actions can be done in the future.

In RTS games, there is a need to perform hundreds of actions per minute. The actions are divided in several tasks involving resource gathering, creation of new units, construction of buildings and attacks to the enemy [Weber et al. 2011].

StarCraft has an application programming interface, called BWAPI [BWAPI 2011], intended for algorithm development, especially algorithms for agent coordination. BWAPI is capable of retrieving the same information and sending the same commands a human player is allowed in *StarCraft*.

In *StarCraft*, there are three races with different characteristics, according to a community of gamers[3]: Protoss, characterized by powerful units that demand a higher amount of resources to produce, Zerg, characterized by attacking with large amounts of cheap units and Terran, which has units of intermediate power and cost. There are two types of resources to produce units and buildings: mineral and gas. To win a game, a player must destroy all the buildings of his opponent. Figure 1 presents a game screenshot.

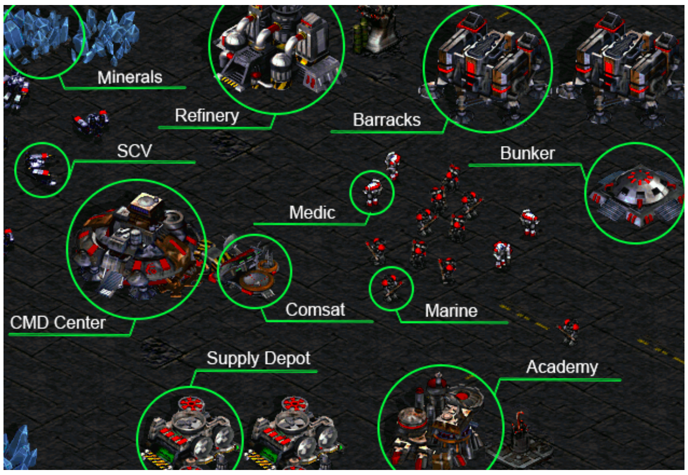

Figure 1: *StarCraft screenshot of a Terran base. Each unit and structure has a specific function in the game.*

In *StarCraft*, agents can perform only one task at a time. This eliminates the need to model agent resources and task costs (r and c in Alg. 1, respectively). These are needed to address the situation where agents can perform multiple tasks simultaneously.

4.2 Running Swarm-GAP in *StarCraft*

In order to execute Swarm-GAP algorithm in *StarCraft*, we implemented a software-controlled player (bot), called GASW (abbreviation of genetic algorithm Swarm-GAP) through BWAPI. In this bot, task allocation among agents is performed according to Alg. 1.

GASW bot plays with the Terran race, using 7 out of 17 available Terran buildings and 3 out of 13 Terran units. Although via Swarm-GAP we obtain which tasks will be performed, we need to hand-code how these tasks will be performed in the game. This hand-coded behavior can be complex depending on the unit type. Thus, due to the complexity of modeling and implementing the task execution behavior of every entity inside the game, we model only a subset containing the basic buildings and units available to GASW bot.

Terran race was chosen because it is the only among the three races whose basic combat unit can target ground and air enemies, which would increase the chances of victory.

All units and buildings used by Swarm-GAP are shown in Fig. 1. Their function is described in the list that follows.

- Buildings:
  - Command center: receives gathered resources and produces workers (SCV).

[3]`http://wiki.teamliquid.net/starcraft/Portal:Beginners`

  - Comsat: allows periodic scans in the map, useful for scouting.
  - Supply depot: are required to increase the number of units that can be created.
  - Barracks: produces marines and medics.
  - Academy: is required for the production of medics and allows the research of combat upgrades for marines.
  - Refinery: is required to collect gas, which is needed to train medics and to perform upgrades at the Academy.
  - Bunker: defensive building that provides shelter for up to 4 ground units. Allows marines to attack enemies with increased range.
- Units:
  - SCV: worker unit that gathers resources, constructs and repairs buildings.
  - Marine: ranged combat unit. Can target air and ground enemies.
  - Medic: auxiliary combat unit that heals other units.

In GASW bot, three types of agents allocate tasks according to Alg. 1: SCV, marine and a commander agent. SCV and marine have counterparts in the game. Commander is an abstract agent, responsible for deciding when SCV, marine and medic units should be produced. The behavior of the medic unit is hand-coded as it is an auxiliary unit created to keep other units alive for longer periods.

Table 1 presents the tasks that agents must allocate via Swarm-GAP. Cells are marked where an agent can perform the given task.

Task	SCV	Marine	Commander
Gather minerals	√		
Build barracks	√		
Build supply depot	√		
Build academy	√		
Build refinery	√		
Build command center	√		
Repair building	√		
Explore map	√	√	
Attack	√	√	
Train SCV			√
Train medic			√
Train marine			√

Table 1: *Agent-task compatibility for tasks allocated via Swarm-GAP*

For the execution of Swarm-GAP algorithm, stimulus must be modeled for all tasks and agent capabilities must be modeled for every compatible agent-task combination. Some game-related tasks are not allocated via Swarm-GAP thus they do not appear in Tab. 1. These tasks include: build Comsat, Bunker and gather gas. Allocation and performance of these tasks are hand-coded in the following way: the Comsat is built after 15 minutes of game, one Bunker is built near each Command Center and three SCVs are allocated for gas collection in the Refinery. This configuration was set after preliminary experiments to allow basic scouting (with Comsat), basic defenses (with Bunker) and a suitable gas collection rate (with three SCVs at refinery).

The behavior of GASW bot is controlled by 27 parameters: one stimulus parameter for each task in Tab. 1, in a total of 12, one capability parameter for each compatible agent-task combination, in a total of 14 (9 for the SCV, 2 for the marine and 3 for the commander), and one parameter that controls the size of an attacking group of marines. The last parameter is not related to execution of Swarm-GAP algorithm, but controls an important aspect of GASW bot.

5 Optimizing Swarm-GAP parameters

5.1 Genetic algorithm

Finding a good combination of Swarm-GAP parameters may be impractical for large scenarios if done manually. To address this issue, we employ a genetic algorithm to automatically find a combination of parameters that maximizes a global metric of agent performance.

Briefly, a genetic algorithm (GA) is a metaheuristic that mimics the process of natural selection. An initial population is generated, the fitness of its individuals is evaluated, individuals are selected to produce the population of the next generation, genetic operators are then applied, and the process is repeated until a stop criteria is satisfied. The stop criteria is usually related to the solution convergence or number of generations. Genetic algorithms are usually applied to complex optimization problems [Haupt and Haupt 2004].

In our approach, an individual is given by a chromosome represented by an array of 27 parameters that control the behavior of GASW bot (see Section 4.2). The domain of the 26 parameters related to Swarm-GAP algorithm (stimulus and capabilities for agent-task combinations) is the set of real values from 0 to 1 spaced by 0.05: $\{0, 0.05, 0.10, ..., 0.90, 0.95, 1.0\}$. The set $[0, 1]$ was discretized in this way in order to reduce the search space of the genetic algorithm without significant loss in precision for the control of Swarm-GAP algorithm. The domain of the last parameter of GASW bot (attacking marine group size) is the set $\{6, 8, 10, ..., 20, 22, 24\}$, i.e., the set of even integers between 6 and 24, inclusive. Odd integers are not considered for search-space reduction as well.

5.2 Fitness function

Evaluation of an individual in our approach is based on the performance of GASW bot in a *StarCraft* match. Our bot implements Swarm-GAP, loading the parameters contained in the chromosome of the individual to be evaluated in order to perform allocation of the game-related tasks presented in Tab. 1.

At the end of a *StarCraft* match, each player has an assigned score according to its performance in resource gathering, structure construction, army deployment and attacks to enemy forces. The fitness function in our approach is the ratio of the score obtained by GASW bot and the score obtained by its adversary. GASW bot wins the game when its fitness is greater than one and it loses when it is lower.

Fitness evaluation depends on the entire execution of a *StarCraft* match. In order to accelerate match execution, *StarCraft* application is executed with minimal graphics and user interactivity. Also, a match is interrupted when it reaches one hour of in-game time. With minimal graphics and user interactivity, in-game time is much faster than real-world time. It takes about 2 real-world minutes to simulate one-hour of in-game time[4]. Considering that many individuals must be evaluated across several generations in a genetic algorithm, fitness evaluation is a time-consuming task.

5.3 Accelerating the genetic algorithm

As fitness evaluation is a time-consuming task, in this paper we test the method described in [Salami and Hendtlass 2003] to accelerate the genetic algorithm.

Briefly, acceleration is obtained by estimating fitness of some individuals instead of actually evaluating them. In this approach, each individual i has associated values of fitness (f_i) and reliability ($w_i \in [0, 1]$). A value of 1 for w_i means that the actual fitness of individual i was evaluated. Other values mean that the fitness was estimated. Let individuals a and b be the parents of c and d. Considering individual c, its fitness is estimated via Eq. 4 and its reliability is calculated according to Eq. 5 (calculations for individual d are analogous).

[4]Approximate value, estimated using modern PCs.

$$f_c = \frac{f_a w_a \rho_{ac} + f_b w_b \rho_{bc}}{w_a \rho_{ac} + w_b \rho_{bc}} \tag{4}$$

$$w_c = \frac{(w_a \rho_{ac})^2 + (w_b \rho_{bc})^2}{w_a \rho_{ac} + w_b \rho_{bc}} \tag{5}$$

In Eqs. 4 and 5, $\rho_{ac} \in [0, 1]$ is the similarity between a and c. Fitness of a child estimated via Eq. 4 is the weighted average of the parents' fitness. The weight is the product of parent reliability and parent-child similarity (w times ρ). Reliability calculated via Eq. 5 is the weighted average of the product between parent reliability and parent-child similarity (w times ρ). The weight is also w times ρ. This way, child reliability is closer to that of the most similar and reliable parent.

To calculate similarity ρ_{ac} between individuals a and c, let A be the chromosome of a and C be the chromosome of c. Also, let max_i and min_i be the maximum and minimum value of the domain of the variable in locus i of the chromosome. The similarity between a and c is then calculated via Eq. 6.

$$\rho_{ac} = 1 - \frac{1}{|A|} \sum_{i=1}^{|A|} \frac{abs(A[i] - C[i])}{max_i - min_i} \tag{6}$$

In Eq. 6, each term of the sum represents the normalized difference (i.e. between 0 and 1) of values in locus i of the chromosomes. The average of these normalized differences gives us an index of divergence between the two chromosomes. The similarity between the individuals is the complement of the divergence index.

If the estimated fitness of an individual i falls below a given threshold ($\tau \in [0, 1]$), the actual fitness is evaluated and assigned to f_i. Also, the reliability w_i is set to 1. Note that, when the fitness of an individual is estimated, its fitness lies between the fitness of the parents and its reliability is lower than the highest reliability of the parents. This is desirable, because with successive estimations across generations, reliability should drop below the threshold and an actual evaluation must take place.

A value of reliability threshold (τ) close to 1 results in many actual fitness evaluations, slowing down the genetic algorithm. On the other hand, a value of τ close to 0 results in many successive fitness estimations which may yield fitness values that differ too much from the actual fitness of the individuals. Thus, τ is an important parameter of this fitness estimation method and should be carefully adjusted.

In order to ensure that some individuals are evaluated instead of estimated in every generation, a probability of evaluation ($p_e \in [0, 1]$) is employed. Individuals with reliability above the threshold are evaluated with probability p_e. This prevents some generations from having all individuals with estimated fitness.

For a complete description of the fitness estimation method adopted in this paper, the reader may refer to [Salami and Hendtlass 2003].

Algorithm 2 formalizes our approach. In this algorithm, $P^{(n)}$ is the population in generation n, the maximum number of generations is η and κ is the population size. Method $select\_parents$ selects two individuals from the population. Method $crossover\_and\_mutation$ receives two individuals, performs crossover according to a crossover probability, mutates the individuals according to the mutation probability and returns the two individuals. Note that selection and crossover methods are not specified and can be chosen according to the situation. For example, in this paper we use tournament selection and one-point crossover (see Section 6).

5.4 Evolving Swarm-GAP in *StarCraft*

This section describes the architecture used in our approach to implement the genetic algorithm that searches for an array of parameters that results in good performance of Swarm-GAP algorithm, i.e., that leads GASW bot to victories.

Algorithm 2 Genetic algorithm for Swarm-GAP

```
1:  $P^{(0)} \leftarrow$ {initial random population}
2:  for all $p \in P^{(0)}$ do
3:      $f_p \leftarrow Evaluate(p)$
4:      $w_p \leftarrow 1$
5:  end for
6:  for $n \in [0..\eta]$ do
7:      $P^{(n+1)} \leftarrow \emptyset$
8:      while $|P^{(n+1)}| < \kappa$ do
9:          $(a, b) \leftarrow select\_parents(P^{(n)})$
10:         $(c, d) = crossover\_and\_mutation(a, b)$
11:         $f_c \leftarrow$ fitness estimated via Eq. 4
12:         $w_c \leftarrow$ reliability calculated via Eq. 5
13:         if $w_c < \tau$ or ($w_c \geq \tau$ and $random() < p_e$) then
14:             $f_c \leftarrow Evaluate(c)$
15:             $w_c \leftarrow 1$
16:         end if
17:         /* repeat lines 11-15 for individual $d$ */
18:         $P^{(n+1)} \leftarrow P^{(n+1)} \cup \{c, d\}$
19:     end while
20: end for
```

GASW bot communicates with *StarCraft* via BWAPI calls. However, during the execution of the genetic algorithm, GASW bot is responsible only for the evaluation of an individual, i.e., subroutine *Evaluate* in Alg. 2. The remainder of Alg. 2 is implemented in the genetic algorithm controller module (GA controller). This module places files with the data of the individual to be evaluated in a specific directory that GASW bot reads. After the evaluation of an individual, which corresponds to a *StarCraft* match, GASW bot writes the match outcomes in the same directory, where the external module extracts score information and calculates the fitness of the evaluated individual.

Figure 2 illustrates the implemented architecture. In the beginning of an experiment, a configuration file with the parameters of the genetic algorithm (crossover and mutation probabilities, reliability threshold, etc.) is loaded by the external module.

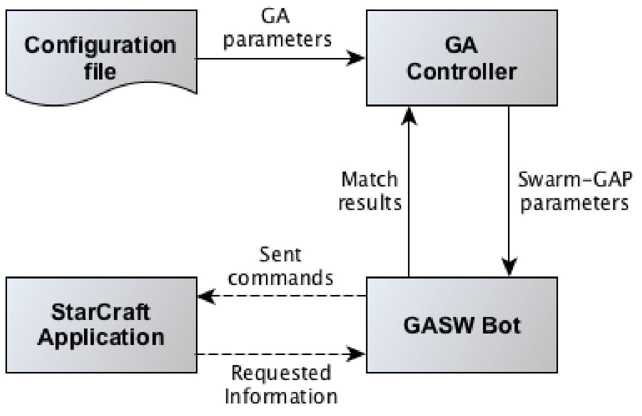

Figure 2: *Architecture of the proposed approach. Solid lines represent data exchanged via files. Dashed lines represent data exchanged via BWAPI calls.*

6 Experiments and discussion

In this section we evaluate our approach in two sets of experiments. First, we analyze genetic algorithm behavior in terms of fitness along generations, comparing performance with fitness estimation and with actual evaluation of all individuals. Second, we analyze the best individual found by the genetic algorithm compared to other *StarCraft* bots in matches against *StarCraft's* native AI, or SC bot for short. SC bot is able to play with the three races, using different strategies and army compositions. SC bot is competitive against beginner human players.

Experiments presented in this paper were executed with selection by tournament with 2 participants and one-point crossover, where a crossover point on both parents' chromosomes is randomly selected. All data beyond that point is swapped between parents to produce the children. Also, we employ elitism, adding the best individual from a generation to the next one. Mutation occurs by replacing the value in a locus by a randomly selected value from that locus' domain. Previous experiments were performed in order to find a suitable combination of parameters for the genetic algorithm, which are:

- Crossover probability: 0.9.
- Mutation probability: 0.01 per locus.
- Number of generations: 100.
- Population size: 30 individuals.

6.1 Genetic algorithm behavior

In this section, we analyze the behavior of the genetic algorithm in terms of the mean fitness of the population. We analyze the performance of the genetic algorithm when fitness of all individuals is evaluated ($\tau = 1$ for Alg. 2) compared to when fitness estimation is employed. Fitness estimation is performed with $\tau = 0.8$ and $p_e = 0.1$. Matches are played against SC bot.

Figure 3 shows fitness along the generations for GASW bot evolved by the genetic algorithm with fitness estimation (dotted lines) and without it (solid lines). Each line in Fig. 3 shows the average of 5 genetic algorithm runs. The distance between solid lines (or dotted lines among themselves) shows that GASW bot performs better against Zerg race controlled by SC bot. Conversely, GASW bot performs worse against Protoss. A Terran adversary controlled by SC bot lies between them. In all cases, fitness stabilizes above 1.0 which means that individuals are winning the matches on average.

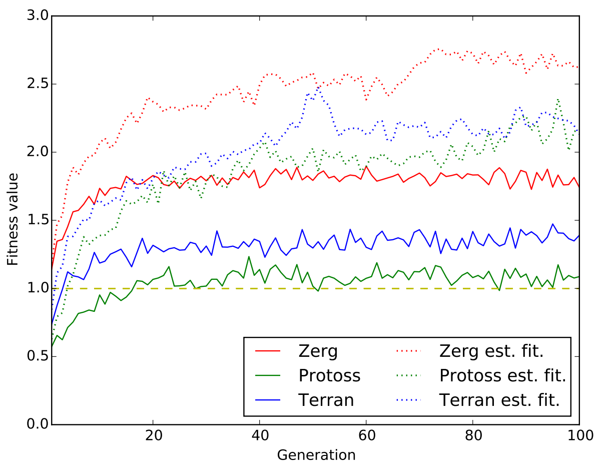

Figure 3: *Genetic algorithm performance. Dotted lines are from experiments with fitness estimation. Solid lines are from experiments where all individuals are evaluated. Fitness above 1.0 (baseline) means victory in a match.*

The genetic algorithm stabilizes with a higher fitness when estimation occurs, as dotted lines are always above solid lines of the same color. This could lead to the misleading conclusion that the genetic algorithm finds better individuals when fitness estimation is employed, which is not the case.

The genetic algorithm with fitness estimation appears to have superior performance because of the nature of our fitness function, which is noisy. That is, the same array of parameters can receive distinct fitness evaluations in different matches. Noise in the fitness function comes in two ways. First, it depends on the probabilistic way that agents choose to perform tasks in Swarm-GAP. Second, *StarCraft* is an adversarial game. Thus, the score ratio also depends on the adversary actions, which may be randomized in order to become more difficult to be predictable.

With the noisy fitness function, fitness estimation became misleading in our experiments. An individual whose observed fitness is high might in fact have a low actual fitness. With fitness estimation, this individual may propagate itself across generations without being actually evaluated again. This may direct the search of the genetic algorithm to the neighborhood of that individual, which may be less promising than other regions where the observed fitness happened to be lower.

6.2 Comparison with other bots

In this section we employ the best individual found by the genetic algorithm and analyze its performance compared to Random, ManSW and AIUR bots in matches against SC bot. Random and ManSW bots have the same limitations of GASW: they play with Terran race using the same units and buildings. Task execution is the same of GASW. These bots differ only in the way tasks are allocated.

Task allocation in Random bot is as follows: for each agent, given a list of tasks, one is chosen with uniform probability. This way, tasks are given equal importance during the allocation process.

ManSW bot allocates tasks via Swarm-GAP, similarly to GASW. The difference is that ManSW bot runs with a hand-configured array of parameters whereas GASW runs with parameters configured via genetic algorithm. We configured the parameters of ManSW bot with the values that achieved the best performance in a set of matches against SC bot. The number of matches performed to adjust the parameters of ManSW is small compared to that of the genetic algorithm for GASW.

AIUR is a competitive bot that placed 3rd among 8 bots in AIIDE 2013 competition[5] and in IEEE CIG 2013 competition[6]. Among all competitors, AIUR was the bot with best performance that we were able to obtain the source code, compile and execute it with success in order to compare the performance of our approach. Briefly, in AIUR, several game-related tasks are divided among many different modules. At the beginning of a game, the bot initializes a "mood" that influences the adopted strategy (focus on resource collection, early attacks or defense). This bot does not perform reactive controls (micro-management) [Ontanon et al. 2013].

Figure 4 shows the victory rate of all bots versus Protoss, Terran and Zerg adversaries controlled by SC bot. Results are based in 150 matches. In this figure, GASW uses the best individual found by the genetic algorithm without fitness estimation whereas GASW-e uses the best individual found when fitness estimation is employed (solid versus dotted lines in Fig. 3).

GASW bot outperforms GASW-e, especially against Protoss and Terran. This illustrates the fact that the superior fitness values obtained with fitness estimation is misleading. The best individuals found with fitness estimation perform worse than the ones found with actual fitness evaluation.

All bots had the best performance against a Zerg adversary controlled by SC bot, followed by Terran and Protoss, against which the bots had the worst performance. Assuming that races are balanced in *StarCraft*, this result may suggest that SC bot was developed to perform better with Protoss race and this made the tested bots face more difficult matches.

Random bot had the worst performance among all bots. This is due to the poor task allocation that emerges from the uniformly random selection of tasks performed by agents. GASW outperforms ManSW against Terran. This means that the genetic algorithm was able to find a better combination of parameters than the one manually set into ManSW. This validates the use of genetic algorithm

[5]`http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/report2013.shtml`

[6]`http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2013`

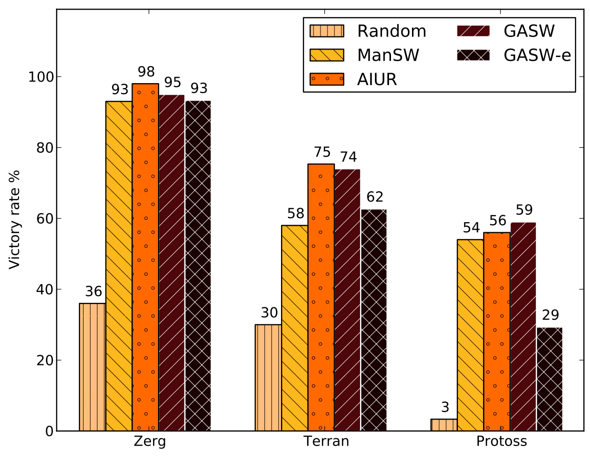

Figure 4: *Victory rates (%) of the bots in 150 matches against Protoss, Terran and Zerg races controlled by SC bot.*

to tune the parameters of Swarm-GAP in the studied scenario. The weakness of Zerg and the strength of Protoss controlled by SC bot made ManSW and GASW perform similarly.

GASW and AIUR achieved similar performance against all three SC-controlled races, which also validates our approach. AIUR is one of the top-performing *StarCraft* bots known to date. Therefore, using scalable task allocation algorithms and tuning them with genetic algorithms is a promising approach for bot development in RTS games.

7 Conclusion

7.1 Overview

In this work we tackled the problem of task allocation in complex scenarios by employing a genetic algorithm to adjust the parameters of Swarm-GAP, a scalable task allocation algorithm. Our approach is tested in the popular RTS game *StarCraft: Brood War*.

Results of our experiments show that the proposed approach is promising. The proposed approach outperforms random task allocation and manually-configured Swarm-GAP. Moreover, the performance of our approach is at par with AIUR, one of the top-performing bots for *StarCraft*. However, performance was measured as the victory ratio in matches against *StarCraft's* native AI and not in direct matches[7].

To deal with slow fitness evaluation, that depends on the execution of an entire match against *StarCraft's* native AI, we tested a fitness estimation method. Performance achieved by the genetic algorithm with estimated fitness was worse than with actual fitness evaluation. Performance degradation with fitness estimation is due to noise in the fitness function, which comes from the probabilistic nature of Swarm-GAP task allocation and from adversary actions in *StarCraft*. Fitness noise may direct the search of the genetic algorithm to regions where observed fitness happened to be higher than more promising regions.

7.2 Future work

A problem that can be addressed in future work is the fitness noise. If we assume that the fitness noise is Gaussian, one possible approach is to evaluate a single individual several times and calculate the mean of the observed fitness as it converges to the actual fitness of the individual.

[7]Unfortunately, direct GASW vs AIUR matches could not be configured due to different, incompatible versions of BWAPI used by the bots.

However, as pointed in [Fitzpatrick and Grefenstette 1988], increasing population size and decreasing fitness evaluations per individual may result in better performance, when the overhead imposed by the genetic operators is negligible compared to fitness evaluation time.

On the other hand, with increased population, the entire execution of the genetic algorithm can be very slow. Thus, accelerating the genetic algorithm becomes crucial. Further studies about the fitness estimation method are needed to address the performance degradation observed in our experiments. The tested fitness estimation method might perform well with noisy fitness, as estimation is itself a form of fitness noise [Salami and Hendtlass 2003]. Future studies can investigate the conditions that lead to reliable fitness estimation. It might be the case that the quality of fitness estimation could be improved with a larger population.

Our approach use only a small subset of buildings and units provided by the race it plays in *StarCraft*. Future work could extend this by adding the remaining buildings and units. This might increase the competitiveness of our approach as more flexible strategies can be adopted. However, to achieve tournament-winning performance, new aspects should be introduced into our architecture. This could include a library of game openings, hierarchical decision making, unit micro-management (reactive control) and terrain analysis [Ontanon et al. 2013].

Other interesting topics for future work include the use of the genetic algorithm to tune the parameters of different task allocation algorithms, such as BnB FMS [Macarthur et al. 2011]. The performance of the different task allocation algorithms evolved by the genetic algorithm can be compared.

At last, this approach could be extended to other domains. Interesting areas outside the game industry are robot soccer and autonomous robot coordination in hostile environments such as rescue operations in disaster situations. On those scenarios, task allocation is a vital part of multiagent cooperation.

Acknowledgment

Authors would like to acknowledge the support of CAPES, CNPq and FAPEMIG in this research. We also would like to thank the anonymous reviewers for their suggestions of paper improvements.

References

BURO, M. 2003. Real-Time Strategy Games: A New AI Research Challenge. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence, 1534–1535.

BWAPI, 2011. An API for interacting with Starcraft: Broodwar. `https://code.google.com/p/bwapi/`.

DE FREITAS CUNHA, R. L., AND CHAIMOWICZ, L. 2010. An Artificial Intelligence System to Help the Player of Real-Time Strategy Games. In *Proceedings of the 2010 Brazilian Symposium on Games and Digital Entertainment (SBGAMES),*, 71 –81.

FALKE II, W. J., AND ROSS, P. 2003. Dynamic strategies in a real-time strategy game. In *Genetic and Evolutionary Computation—GECCO 2003*, Springer, 1920–1921.

FERNANDEZ-ARES, A., MORA, A. M., MERELO, J., GARCÍA-SÁNCHEZ, P., AND FERNANDES, C. 2011. Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, IEEE, 2017–2024.

FERREIRA, JR., P. R., BOFFO, F., AND BAZZAN, A. L. C. 2008. Using Swarm-GAP for distributed task allocation in complex scenarios. In *Massively Multiagent Systems*, N. Jamali, P. Scerri, and T. Sugawara, Eds., no. 5043 in Lecture Notes in Artificial Intelligence. Springer, Berlin, 107–121.

FIERRO, R., DAS, A., SPLETZER, J., ESPOSITO, J., KUMAR, V., OSTROWSKI, J. P., PAPPAS, G., TAYLOR, C. J., HUR, Y., ALUR, R., LEE, I., GRUDIC, G., AND SOUTHALL, B. 2002. A framework and architecture for multi-robot coordination. *The International Journal of Robotics Research 21*, 10-11, 977–995.

FITZPATRICK, J. M., AND GREFENSTETTE, J. J. 1988. Genetic algorithms in noisy environments. *Machine learning 3*, 2-3, 101–120.

HAUPT, R. L., AND HAUPT, S. E. 2004. *Practical genetic algorithms*. John Wiley & Sons.

KHAN, M. A., TURGUT, D., AND BÖLÖNI, L. 2010. Optimizing coalition formation for tasks with dynamically evolving rewards and nondeterministic action effects. *Autonomous Agents and Multi-Agent Systems 22*, 3 (May), 415–438.

KHATIB, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research 5*, 1, 90–98.

KITANO, H. 2000. Robocup rescue: A grand challenge for multiagent systems. In *Proc. of the 4th International Conference on MultiAgent Systems*, Los Alamitos, IEEE Computer Society, Boston, USA, 5–12.

LARA-CABRERA, R., COTTA, C., AND FERNÁNDEZ-LEIVA, A. J. 2013. A review of computational intelligence in RTS games. In *Foundations of Computational Intelligence (FOCI), 2013 IEEE Symposium on*, IEEE, 114–121.

LIN, C., AND TING, C. 2011. Emergent tactical formation using genetic algorithm in real-time strategy games. In *Technologies and Applications of Artificial Intelligence (TAAI), 2011 International Conference on*, 325–330.

LUDWIG, J., AND FARLEY, A. 2009. Examining Extended Dynamic Scripting in a Tactical Game Framework. In *Artificial Intelligence and Interactive Digital Entertainment*.

MACARTHUR, K. S., STRANDERS, R., RAMCHURN, S. D., AND JENNINGS, N. R. 2011. A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems. In *Proc. of the 25th AAAI Conference on Artificial Intelligence*, 701–706.

NG, P. H., LI, Y., AND SHIU, S. C. 2011. Unit formation planning in RTS game by using potential field and fuzzy integral. In *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, IEEE, 178–184.

ONTANON, S., SYNNAEVE, G., URIARTE, A., RICHOUX, F., CHURCHILL, D., AND PREUSS, M. 2013. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *Computational Intelligence and AI in Games, IEEE Transactions on 5*, 4 (Dec), 293–311.

RATHE, E. A., AND SVENDSEN, J. B. 2012. Micromanagement in Starcraft using potential fields tuned with a multi-objective genetic algorithm.

SALAMI, M., AND HENDTLASS, T. 2003. A fast evaluation strategy for evolutionary algorithms. *Applied Soft Computing 2*, 3, 156–173.

SANDBERG, T. W., AND TOGELIUS, J. 2011. *Evolutionary Multi-Agent potential field based AI approach for SSC scenarios in RTS games*. PhD thesis, Master's thesis, IT University Copenhagen.

SCERRI, P., FARINELLI, A., OKAMOTO, S., AND TAMBE, M. 2005. Allocating tasks in extreme teams. In *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, USA, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds., 727–734.

SHMOYS, D. B., AND TARDOS, V. 1993. An approximation algorithm for the generalized assignment problem. *Mathematical Programming 62*, 3, 461–474.

SPRONCK, P., SPRINKHUIZEN-KUYPER, I., AND POSTMA, E. 2004. On-line adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games & Simulation 3*, 1.

SYNNAEVE, G., AND BESSIÈRE, P. 2011. A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft. In *Artificial Intelligence and Interactive Digital Entertainment*.

SYNNAEVE, G., AND BESSIÈRE, P. 2011. A Bayesian model for RTS units control applied to StarCraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, IEEE, 190–196.

SYNNAEVE, G., AND BESSIÈRE, P. 2011. A Bayesian model for opening prediction in RTS games with application to Starcraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, IEEE, 281–288.

THERAULAZ, G., BONABEAU, E., AND DENEUBOURG, J. 1998. Response Threshold Reinforcement and Division of Labour in Insect Societies. In *Royal Society of London Series B - Biological Sciences*, vol. 265, 327–332.

WEBER, B. G., MATEAS, M., AND JHALA, A. 2011. Building human-level AI for real-time strategy games. In *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*, 329–336.