HADMÉRNÖK

VI. Évfolyam 2. szám - 2011. június

Harmati István
harmati@iit.bme.hu

Kisfaludi Péter
kisfaludi.peter@gmail.com

MILITARY STRATEGY PLANNING FOR AUTONOMOUS GROUND VEHICLES

Absztrakt

Több ágenst tartalmazó katonai robotrendszer koordinációs problémája játékelméleti keretek között hatékonyan vizsgálható. A koordinációs probléma megoldásához a mesterséges intelligencia módszerek eszköztára, így például a megerősítéses tanulás is alkalmazható. A cikkben bemutatásra kerül egy olyan, megerősítéses tanuláson alapuló stratégiatervezési módszer, amely képes több ágenst tartalmazó ember nélküli földi járművek esetén az ágensek számára optimális stratégiát kialakítani.

The coordination problem within a multiagent robot system can be efficiently examined in a game-theoretic framework. The solution for the coordination problem can be found using artificial intelligence methods, for example with reinforcement learning. In this article, a reinforcement learning based method is described, which is capable of finding an optimal strategy for a group of Unmanned Ground Vehicle (UGV).

Kulcsszavak: *megerősítéses tanulás, gépi tanulás, multiágens stratégiatervezés, multiágens robotrendszer ~ reinforcement learning, machine learning, multiagent strategy planning, multiagent robot system*

INTRODUCTION

Military strategy planning plays important role in battles since ancient ages. The scientific research of this discipline arrived in a new era since computers and autonomous military vehicles (robots) had appeared on the battle field. It is especially true if one considers that strategies can be simulated and analyzed by computing science. Artificial Intelligence (AI) and game theoretic methods in computer games has also impact on the state-of-the-art military strategies. Military strategies consider and coordinate such tactical operation as for example task assignment, pursuit-evasion games, formation control. Successful mission requires cooperation between agents (UGVs) to reach global (shared) goal. At the same time individual agents (or a group of agents) should execute different, coordinated actions in order to achieve the global and shared goal for the team. Since optimal maneuver planning is too complicated, it is a reasonable approach to decompose the mission planning into different levels. On higher level, the strategy defines a global goal to every team member or groups of team members. It also means that individual military goal is defined for each team mate (or group of team mates) by the strategic level. UGVs should solve their task individually on tactical level. This often includes path planning and collision avoidance algorithms [1]. Based on the planned paths, low level control method should provide control signal to the UGVs via actuator. For example, if UGVs are represented by tanks, low level control signals are the velocities of the wheels on the right hand side and the left hand side. The advantage of splitting up the problem is that the group level computations can be done in a parallel manner, and the complexity of these several computations is less than the complexity of solving the coordination problem for the whole team.

There are several potential methods which attempts to reach optimal strategies to the troops [2]. Since the problem is very complicated, they are mostly based on heuristics, soft computing methods [3], [4] e.g. fuzzy systems, neural network, swarm intelligence, reinforcement learning) or hard computing methods which provide at least sub-optimal solutions (e.g. game theory [5], [2], [6]).

In this paper, we propose reinforcement learning method for high level military strategy planning. Unfortunately, classical reinforcement learning techniques [7], [8] do not provide straightforward solution for team games and thus for military operation planning. On the other hand, these techniques are performing well in their domain (for example in single agent frameworks), that is why one can hope that an extension of these techniques to team games will solve the coordination problem emerging in military operations planning with still a good performance. In the reinforcement learning methods, we apply WoLF principle [7]. This is a solution for efficient reinforcement learning in a multiagent framework, and this method is able to find a locally optimal solution in the multiagent domain, and it is also proven that by applying the principle to reinforcement learning methods these methods become convergent, which is an important issue during military operations planning. In this paper, we propose a possible extension of the single agent framework to a multiagent domain and match it to military operations planning. As a result, military operation planning using hierarchical reinforcement learning is introduced.

Another problem emerging in the domain of team games is that the state space and the action space can easily grow to an intractable level, and therefore some simplifications should be applied to them to make the learning algorithms tractable (this means that without the simplifications, the algorithms can still find a solution in the original space but the computation time will become intractable). One way of simplification is to discretize the state and the action space. Discretization reduces the size of the space to a manageable size, but at the cost of losing information, because after discretization, two previously distinct states or actions can become indistinguishable. Another way of reducing the size of the state and action

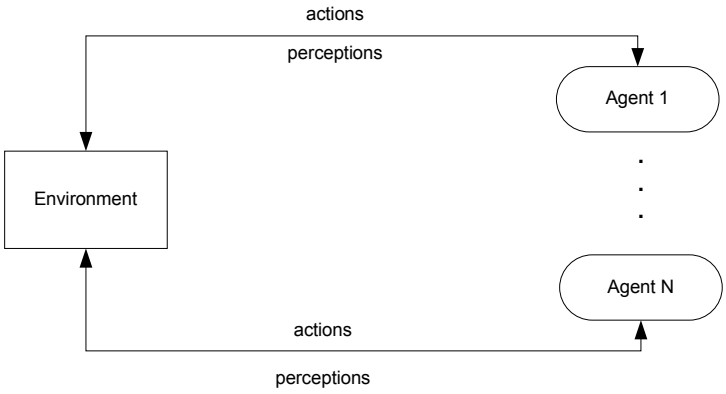
space is to create a simpler model of the environment (for example, by omitting existent but unimportant features of the environment), thus reducing the complexity (and size) of the state and action space.

The methods for strategy planning for military operations are analyzed in a simplified demonstration domain. In this domain, two teams of tanks fight against each other. The tanks are able to move on the map and they can shoot at each other. On the demonstration domain, the map is divided to cells and these cells make up a standard grid. The allowed movement of the tanks is reduced to the four main directions in this grid. The demonstration game simulates the battle in discrete timesteps, the units can move from a cell to an adjacent cell in a timestep or they can shoot. The team that succeeds in destroying all the units in the enemy's team is declared the winner. The multiagent coordination methods are tested and evaluated in this simplified demonstration domain.

The paper is organized as follows. In Section 2, we summarize the theoretical background of multiagent systems, the main results of reinforcement learning methods. Section 3 is devoted to framework used in our investigation and to a specific solution based on reinforcement learning approach is established. Section 4 demonstrates the proposed method via an illustrative example. Finally, we draw the conclusions in Section 5.

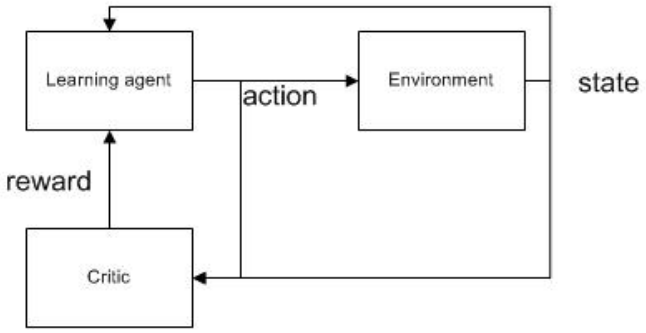
THEORETICAL BACKGROUND

In the field of artificial intelligence and machine learning, agents play a central role. Agent is an entity which can have perceptions from its environment and can change the state of its environment by means of its actuators [3]. The agent generally consists of three main parts: perception, reasoning and actuator, where the reasoning part is responsible for deciding which action the agent should execute based on the actual and previous perceptions. The schematic of an agent is seen on Figure 1. In our description, UGVs can be considered as agents. In multiagent environment one should coordinate their action for successful mission (winning the battle).



1. figure. Multiagent concept

In this paper, we develop a cooperative multiagent control on the base of reinforcement learning. Reinforcement learning is a machine learning technique capable of learning an optimal strategy based on reward signals (see Figure 2). A strategy in this context means a probabilistic distribution over the agent's available actions in a given state. A reinforcement learning problem can be described with the agent's state space and action space. Generally, the actions available to the agent can be different for different states, but usually it is assumed that the available actions are the same for all states. The reward is a signal that is distinct from the state of the environment and the agent is capable of treating the reward signal and the state of the environment distinctly.



2. figure. Reinforcement learning

Thus the learning agent has two inputs: one input is the state of the environment, the other is the actual value of the reward signal. A reward is optionally provided to the agent after executing an action; it is possible that the agent does not receive a feedback (reward) about its executed action. The goal of the learning agent is to maximize the value of the reward signal on the long run. As there are no correct state-action pairs provided to the agent (unlike in supervised learning), the agent can only learn by executing actions in the available states and observing the reward for each state action pair. This also means that the agent should somehow explore its environment, because only after suitably exploring the environment can the agent be sure to have found a strategy which maximizes the long term reward.

Several techniques exists which ensures suitable exploration of the state space, one of them (which is used in the proposed solution) is called the ε-greedy exploration. The ε-greedy exploration makes the agent execute a random action with low probability that is independent of the current strategy. This way no action will be excluded in any state from execution, and the agent has the chance to explore the whole state space.

MILITARY STRATEGY WITH REINFORCEMENT LEARNING

The proposed solution for military operations uses an extension of a single agent reinforcement learning algorithm suitable for stochastic games. The chosen reinforcement learning technique is the GraWoLF technique, it is extended to the domain of team games by using the aggregated agent concept, and the complexity of the solution is reduced by defining a hierarchy between the agents.

Let us start the discussion with some definition. The domain of team games is a subset of stochastic games (stochastic games are multiagent, multistate games [9]). In team games, the agents are partitioned into an arbitrary number of teams and every team has its own reward scheme and goal. The goal and reward are common amongst the agents of the same team, meaning that during learning using a reinforcement learning technique only a single reward signal is provided to the whole team, which all agents can perceive. The agents in the same team are allowed to execute different actions and communication between them is also allowed. Communication makes coordination (executing such individual actions that result in better expected reward than executing actions without coordination) between team members possible. The reward signal is dependant on the performance of the team as a whole.

Military operations belong to the domain of team games, because in a military operation, there are usually more units controlled by the same team trying to achieve a common goal, and the adversary team (or teams) is trying to prevent the team from doing so.

Formally, a team game can be described as a tuple (N, S, $A_{1...N}$, T, $R_{1...N}$), where

- N is the number of teams
- S is the set of states in the game
- A_i is the set of actions available for team i
- T is the transition function
- $T : S \times A_1 \times A_2 \times \ldots \times A_N \times S \rightarrow [0,1]$
- $T(s, a_1, a_2, \ldots a_N, s') = P(s_{t+1} = s' | s_t = s, a_{1_t} = a_1, a_{2_t} = a_2, \ldots a_{N_t} = a_N)$
  $\forall s \in S, \forall a_i \in A_i \quad \sum_{s' \in S} T(s, a_1, a_2, \ldots a_N, s') = 1$
- Ri is the reward function for team i
  $R_i : S \times A_1 \times A_2 \times \ldots \times A_N \rightarrow \Re$

The policy of the agent determines with what probability the agent chooses a particular action from its available action set in a given state. Formally, a policy is a mapping from state and action pairs to a probability. The policy returns the probability of taking a particular action at a given state.

Formally, the policy is denoted by π:
$\pi : S \times A \rightarrow [0,1]$
$\forall s \in S \quad \sum_{a \in A} \pi(s,a) = 1$, where
S is the set of states
A is the set of actions

When one develops a military strategy, then appropriate policies should be found that lead the team to a winning state.

Reinforcement learning in stochastic games: GraWoLF

Reinforcement learning techniques exist that are capable of finding a strategy for a single agent even in the domain of stochastic games. In the domain of team games however, the strategy should handle multiple agents (the team controlled by the strategy), and instead of returning a single action it should return a list of actions where every action in the list corresponds to one agent in the team.

One reinforcement learning technique that is suitable for stochastic games is the GraWoLF (Gradient based Win or Learn Fast) technique [7]. The algorithm is summarized as follows:

1. Let $\alpha \in [0,1], \beta \in [0,1], \delta^l > \delta^w \in [0,1]$ be learning rates
Initialize $\lambda, \gamma, \Delta t, \varepsilon$
Initialize $w \leftarrow \overline{0}, \theta \leftarrow \overline{0}, \overline{\theta} \leftarrow \overline{0}, e \leftarrow \overline{0}$

2. Repeat

(a)
Select action a from state s according to policy π with suitable exploration using ε.

(b)
Observe reward r and next state s'

$$Q_w(s,a) = w^T \phi_{sa}$$
$$Q_w(s',a') = w^T \phi_{s'a'}$$
$$\delta = \begin{cases} \delta_w \text{ if } \sum_a \pi_\phi(s,a) Q_w(s,a) > \sum_a \pi_{\bar{\phi}}(s,a) Q_w(s,a) \\ \delta_l \text{ otherwise} \end{cases}$$

$$e \leftarrow \lambda \gamma^{\Delta t} e + \phi_{sa}$$
$$w \leftarrow w + e\alpha(r + \gamma^{\Delta t} Q_w(s',a') - Q_w(s,a))$$
$$\theta \leftarrow \theta + \gamma^{\Delta t} \delta \sum_a \pi_\phi(s,a) Q_w(s,a)$$

(c)
Update average parameter vector $\bar{\theta} \leftarrow \beta\theta + (1-\beta)\bar{\theta}$

(d)
If s' is the initial state or trial is over then $t \leftarrow t_0, e \leftarrow \bar{0}, s \leftarrow s_0$

In this pseudo code, α is the learning rate for the approximation of the Q-values. The weighting parameter for the maintenance of the average parameter vector is denoted by β. The two other learning rates, δ^l and δ^w means the step size during gradient ascent when the agent is losing or winning, respectively. Parameter λ is influencing the speed of Q-values estimation, γ is the discount parameter for the reward formulation, Δt is the length of one timestep. Parameter ε is the probability of choosing a random action instead of executing an action according to the policy (this is the exploration parameter used in ε-greedy exploration).

Parameter w is used to approximate the Q-values corresponding to states and actions, this is the weighting vector for the approximation function, f_w. The actual parameter vector is denoted as θ, the average parameter vector with $\bar{\theta}$. The e vector is called the eligibility trace, basically it describes the contribution of the state and action pair to the actual error in the estimation of the Q-values.

In step (a), the agent chooses an action according to the actual policy, but with a small probability (ε) it chooses a random action from its action list. In step (b), the agent observes the new state of the environment (s') after executing the chosen action and it optionally receives a reward signal. After that, the approximations for the Q-values are updated (this means updating the weighting vector w and the eligibility vector e). The new parameter vector is calculated using the new approximation of the Q-values and the learning parameter is based on whether the agent is winning or losing. In step (c), the average parameter vector is calculated using β as the learning parameter and using the previous value of the actual and the average parameter vector. In step (d), the agent checks whether it is in its starting state or the time limit for the learning trial is reached, when the trial is restarted and the state of the environment is reset to the initial state and the eligibility vector is nullified.

This technique is scalable and it is able to find a locally optimal strategy for a single agent acting in a stochastic game domain. It requires a Boolean vector describing the state of the game (this vector is called the feature vector) as an input and provides a parameter vector as a result of learning. The action of an agent can be calculated by using the parameter vector and the feature vector. The strategy using the parameter vector calculated with GraWoLF is

locally optimal with regards to the reward function. A strategy (π) in this context means a mapping from state and action pairs to a number between 0 and 1, which is the probability of the agent choosing the given action in a particular state. Formally, $\pi : S \times A \rightarrow [0,1]$ and as the strategy defines a probabilistic distribution over the state-space:

$\forall s \in S \quad \sum_{a \in A} \pi(s,a) = 1$ where S is the set of states and A is the set of actions.

GraWoLF is a gradient ascent technique, meaning that in every learning step it modifies the parameter vector in the direction of the positive gradient of the expected reward function (note that the goal of the agent is to maximize the expected value of the reward, thus it modifies the parameter in the direction of the positive gradient). The step size for the gradient ascent technique is chosen according to the Win or Learn Fast principle. Win or Learn Fast means that the step size is relatively small if the agent is "winning" and the step size is relatively large if the agent is "losing".

Winning and losing are determined by comparing the performance to a so called average performance (it is impossible though to determine exactly if the agent is winning or losing).

Extending GraWoLF to the domain of team games

Handling multiple agents

GraWoLF in its original form is capable of finding a strategy for a single agent, but in team games, an action list for the team is required, where the actions in the list correspond to individual agents in the team.

Handling of multiple agents can be done by using the so called "aggregated agent" concept. The aggregated agent concept is a technique that can be used for an arbitrary number of agents. It defines an aggregated agent, whose state space is the joint state space of the state spaces of the individual agents and the action space of the aggregated agent is the joint action space of the action spaces of the individual agents. Every state and action in the state and action space of the aggregated agent has a unique identifier assigned, therefore the aggregated agent can be treated as a single agent, because it is in one state at a time and executes one action at a time (although the state denotes a list of states and the action denotes a list of actions).

The problem with the aggregated agent concept is that the state and the action space increases exponentially with additional agents, therefore in order to keep the solution tractable, the number of agents in the aggregated agent must be kept small.

Example of an aggregated agent: Two agents with two actions

Assume that there are two agents in the environment; both agents can execute an action from an action list which length is 2. The two agents belong to the same team. The action list for all agents is {a1, a2}.

If S1 denotes the state of agent 1 and S2 denotes the state of agent 2, the state of the game is described by the joint state space of the agents, S1xS2.

The action list for agent 1 is denoted by A1, the action list for agent 2 is denoted by A2 (note that A1=A2={a1, a2}). The action space available for the team, which consists of these two agents, is A1xA2 according to the aggregated agent concept. The resulting action list available for the team as follows {a1xa1, a1xa2, a2xa1, a2xa2}.

Reducing complexity

By using the aggregated agent concept, the size of the state and the action space can easily grow to an intractable size. A hierarchy between the agents is defined in order to reduce the size of the state and the action space. The agents of the same team are partitioned into subgroups, and the subgroups make up the whole team.

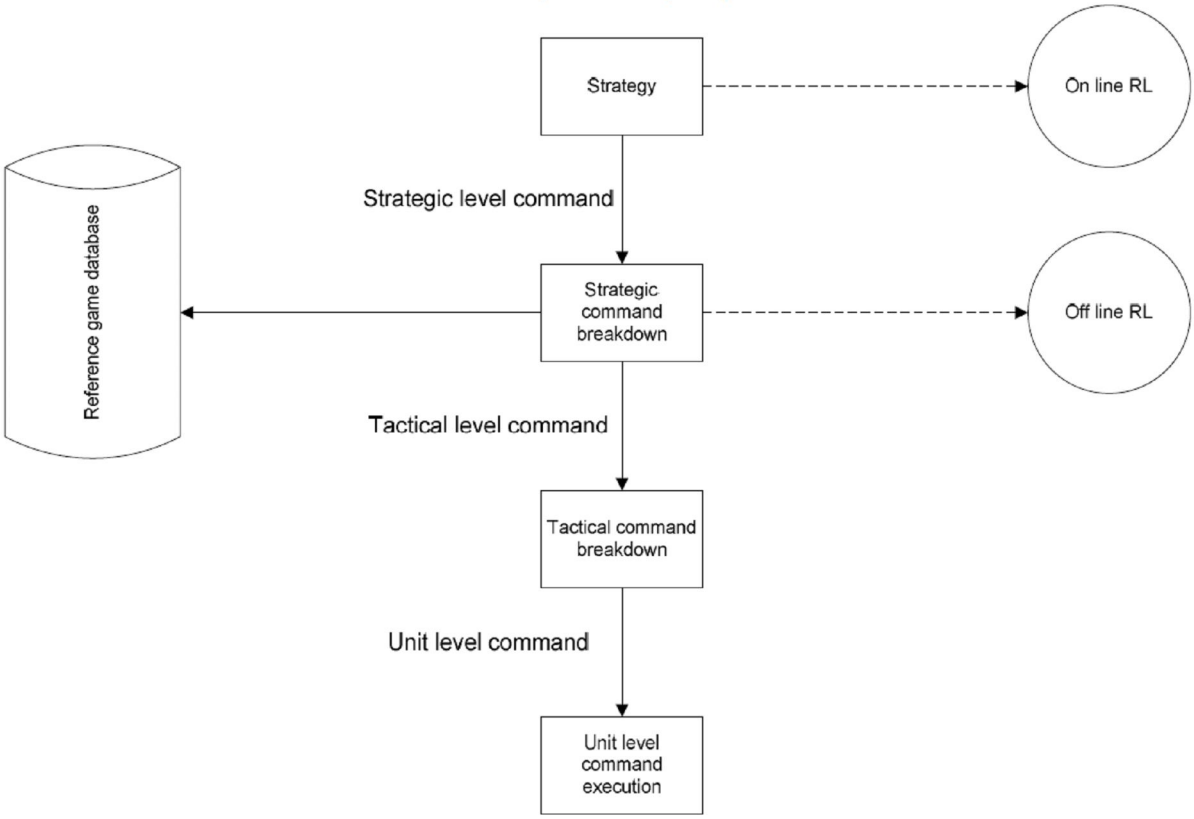
Defining a hierarchy between the agents efficiently reduces the complexity of the aggregated agent at every level, because generally there are less groups than units, and from the point of view of the team, only the actions for the groups has to be calculated.

Only the agents in the same group are allowed to communicate, this restriction in the communication channels means that the strategy will tend to be sub optimal, but this is a trade off between optimality and complexity. Every level in the hierarchy is connected only with at most two other levels, it is the responsibility of every level to transform the action received from a higher level to an action acceptable to a lower level (this process is called action breakdown).

CONCRETE SOLUTION

Hierarchy

In this particular solution, the hierarchy defined within the team has three levels: team (strategic level), group (tactical level) and unit level. Basically, the strategic action calculated at team level is broken down to a list of tactical level actions applicable to groups in the team, and group level actions are broken down into a list of unit level actions. This way a strategic action can be broken down to a list of unit level actions, where the commands can be directly executed by the units. The levels of the strategy and the process of converting a strategic action to a list of unit actions are shown on figure 3.



3. figure. Levels of the strategy [10]

The hierarchy is built up using spatial information between the units. The groups are created from units that are near each other. The number of groups and the number of agents in a group could change dynamically as the agents change their location, but in this particular solution the number of agents in a group and the number of groups was fixed, only the actual groups were created dynamically. The partitioning of the team members into groups is done according to the following algorithm [10]:

```
initialize DistLim, GrNumLim
groups ← ∅
foreach units in team
   group ← ∅
if allocated(unit)
   continue
endif
group ← group ∪ unit
neighbors ← team / unit
repeat
   nearest ← getNearestNeighbor (unit)
   if allocated(nearest)
      continue
   endif
   if distance (nearest) > DistLim
      continue
   endif
   group ← group ∪ nearest
   set Allocated (nearest)
   if size (group) > GrNumLim
      break
   endif
end repeat
```

- DistLim: the maximum distance between the units in a group
- GrNumLim: the maximum number of units in a group.
- Allocated: If the unit is already assigned to a group
- The "group", "groups" and "neighbors" are mathematical sets; the union operation means adding a new item (unit) to the set, the subtract operation means removing an item (unit) from the set. After running this algorithm, all units are partitioned into groups (the variable "groups" will contain the final partitioning), and the distance between the units in a group is less than DistLim, and the number of units is less than GrNumLim.

The strategic actions defined in the demonstration game are the Encircle, Retreat, Advance and Destroy commands; these are the actions available to the team agent at the highest level.

The meaning of these commands:

- Encircle: The chosen groups that are executing the encircle command are trying to encircle the chosen groups of the enemy, meaning that the goal of the action is to block the movement opportunities of the opponent groups in as many directions as possible
- Retreat: The group executing the retreat command tries to increase the distance between itself and all enemy groups.
- Advance: The goal of the advancing group is to decrease the distance between itself and all enemy groups.
- Destroy: The group that executes the destroy command tries to kill as many opponent units from the chosen opponent group as possible.

At tactical and unit level, the Left, Right, Forward, Shoot commands are available.
The meaning of these commands:

- Left: The unit/group turns left.
- Right: The unit/group turns right.
- Forward: The unit/group moves forward.
- Shoot: The unit/group shoots.

Learning at the levels of the hierarchy

Reinforcement learning is used at two parts of the algorithm: at calculating the strategic action for the team, and at strategic command breakdown.

The strategic action calculation part uses GraWoLF in its original form, because the team is treated as a single agent which can execute one action at a time. The action set available to the team is the set of strategic actions (Encircle, Advance, Retreat, and Shoot); the state of the game is represented as the joint state of the teams. This part of learning runs on line, and it is the responsibility of the lower hierarchy levels to convert the chosen strategic action to a list of unit level actions.

The strategic command breakdown module is responsible for creating a list of actions applicable to groups from a strategic action. As the number of groups in a team is usually greater than one, the original form of GraWoLF cannot be used; rather the extension of GraWoLF using the aggregated agent is used.

For every strategic action, at least one reference game is stored in a reference game database. In this database, every entry contains a game state and the locally optimal parameter vector to be used in that particular game state. The optimal parameter vectors are calculated during an off line reinforcement learning session. During this off line learning, the aggregated agent concept is used by the learning agent.

The action set available to the learning agent is generated according to the aggregated agent concept, which means that the actions are lists of actions in their inner representation, where each list item correspond to a group in the team. These action lists are assigned a unique identifier and the learning agent chooses from the set of these identifiers, meaning that from the point of view of the learning agent, the problem is single agent reinforcement learning in a stochastic game domain, where a locally optimal policy can be calculated using GraWoLF. But upon execution of the actions, the chosen action is split (this can be done because the action in reality is an action list), and the resulting group actions are executed by the groups in the team.

Tactical command breakdown is the process of converting a group level action to a list of unit level actions. This process is implemented in a predefined way, meaning that the rules for the breakdown are not updated during the game and no learning is done at this level of the hierarchy. As the action set available to a group is identical to the action set available to a unit, the breakdown is implemented in a simple way: first, the orientation of the agents in the same group is made equal to each other, and then all units in the group execute the action sent to the group.

SIMULATION RESULT

A simulation environment was developed in Matlab to enable development and evaluation of military strategies. The simulator module is the work of Lajos Szarka and Peter Kisfaludi. The simulator is capable of simulating a military operation on an arbitrarily sized two-dimensional map with an arbitrary number of units partitioned into two teams. The module simulates the battle in discrete timesteps. The map on which the battle takes place is also discretized, thus the location of every unit is also discrete (at every timestep, a unit resides on a discrete location called a tile). The orientation of the units is also discretized: it can be any of 0, 90, 180 or 270 degrees.

Environment

The environment of the concrete game where the experiments are carried out is a two dimensional, discretized map in which multiple units reside. The map is represented by a weighted graph, where the nodes of the graph correspond to locations on the map, while the edges of the graph represent paths between two locations.

In the current implementation, the map is a standard grid, where the length of the path between any two adjacent nodes is equal to 100. The edges of the graph are weighted, the weight of an edge corresponds to the length of the path between the two locations the edge connects.

The state of the environment is made available to the learning agents in discrete timestep, and the state update also happens in such fashion.

Objects on the map

Objects on the map can be moving objects or static objects. The static objects are not controlled by any of the teams; the moving objects (the units) are assigned to one of the teams in the game. Every object on the map occupies exactly one node on the map at a time.

The possible objects on the map are the following:

- control point: the control point is a special location on the map which can be owned by any of the teams participating in the battle, or it can be neutral, which means that none of the teams has possession over the control point.
- obstacle: obstacles are locations on the map to where tanks cannot move (and control points also cannot be located there), although path can lead to nodes which contain an obstacle.
- unit: units are moving objects on the map, they are controlled by one of the teams and they are described in detail at Section 4.3.

Units

The moving units on the map are tanks, which can move on the edges of the graph and can execute actions in their environment. The units can be individually controlled by a team. There are two teams present in this environment which battle against each other. All the units are identical regarding their properties, they can be described by the same parameters. These parameters are the position, orientation, healthpoint. The meaning of these parameters is

- position: the location of the tank identified by the node on the graph. A unit can occupy exactly one node at a time, and at most one unit can reside on a node (this restriction also applies to units in the same team).
- orientation: the angle of the tank, which determines the angle of shooting and on which edge the unit can move forward. The orientation of the tank considering that the map is a standard grid can be 0, 90, 180 or 270 degrees.
- health point: the health point of the unit is the number of shots the unit can take without being destructed. If the health point of a unit is above zero, the unit can execute actions and can move on the map. If the health point of a unit reaches zero, it means that the unit is destructed and it cannot execute actions and cannot move. It is also removed from the node it resided in, and another unit can occupy that node.

The available actions for the individual agents are the Left, Right, Forward, Shoot and NoOperation (NOP). The meaning of these actions is:

- Left: the orientation of the unit changes by +90 degrees
- Right: the orientation of the unit changes by -90 degrees
- Forward: the unit tries to move to the adjacent node in the direction of its orientation. If the adjacent node is an obstacle or there is no adjacent node (because the unit is located at the perimeter of the graph) the unit remains on the same node it tried to move away from.
- Shoot: the unit shoots. The angle of the shoot is not deterministic, every tank has a spread of shoot and the actual angle of shoot is the orientation of the tank modified by a random positive or negative angle between 0 and the half of the maximum angle of shoot. The range of shoot is unlimited, and in case of the shoot hits another unit, the health point of the unit that was hit is decreased by one. Friendly fire is available, i.e. tanks from the same team can also shoot each other. Units cannot shoot through obstacles and control points. The path of the bullet that is fired is calculated using the actual angle of the shoot. Every discrete location (node) on the map this path intersects is tested for hit, by always checking the nearest unchecked tile starting from the firing unit. If the path intersects a node that is not empty, the object located on the node is considered hit and the bullet stops. If the object is a tank, the tank that is hit loses a healthpoint, while in case of obstacle and control points nothing happens. The process of determining which object is shot is shown in the following pseudocode:

```
angle ← calculate shoot angle (orientation, spread)
path ← calculate path (angle, position)
nodes ← get intersecting nodes (path)
while there is unchecked node in nodes
  node ← get nearest node (nodes, position)
  if isChecked (node)
    continue
  endif
  setChecked (node)
  obj ← getObject (node)
  if obj ∈ {empty, obstacle, controlPoint}
    break
  endif
  if obj is tank
    decrease HP (tank)
    break
  endif
end while
```

- NOP: the unit does nothing (does not shoot and stays where it was).

Teams

Two teams are present in the concrete game, each team controls four units. The starting position of the units is predefined; they are located at opposite sides of the map. The teams control their units by means of strategic level actions. The available list of strategic level actions is: Encircle, Retreat, Advance and Destroy. The strategic level actions are sent to the whole team, and are further broken down to group level actions. The meaning of these strategic level actions is as follows:

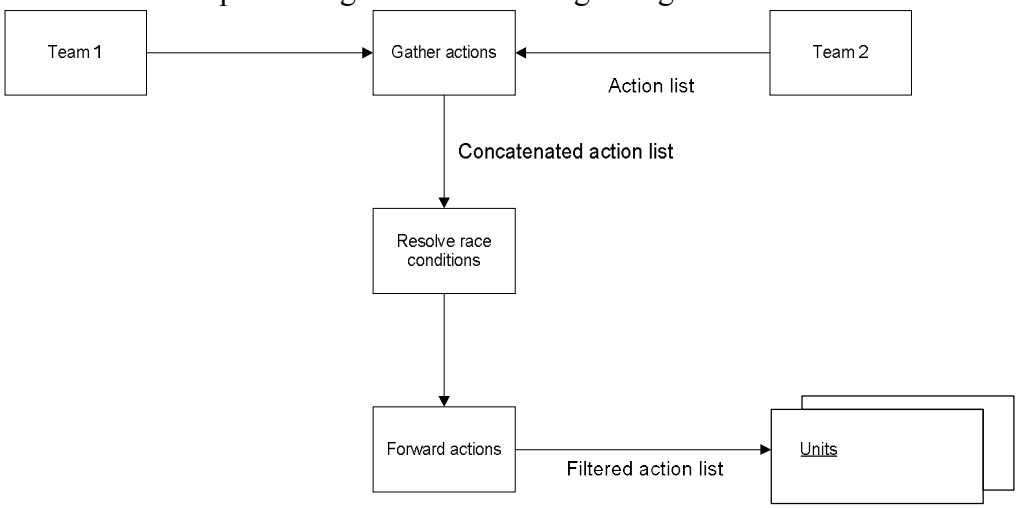
- Encircle: The goal of this command is to stall the opponent group and prevent it from joining the others group for reinforcement. After executing the Encircle strategic action, the expected outcome is that the groups participating in the execution of the Encircle command move closer to the targeted opponent group, and they block the directions in which the opponent group can move as much as possible.
- Retreat: The goal of the Retreat strategic level command is to increase the distance between the groups owned by the executor of the command and the opponent groups. The expected outcome of this strategic level action is that the groups move as far away as they can from the opponent groups, even if the opponent groups are executing some kind of chasing maneuver, during which they try to decrease the distance before the groups.
- Advance: The goal of the Advance strategic level command is to decrease the distance between the controlled groups and the opponent groups. The expected outcome of this command is that the controlled groups move as close to the opponent groups as possible, thus decreasing the distance between them. The groups should be

able to decrease the distance even if the groups belonging to the opponent are executing some kind of retreating maneuvers.

- Destroy: The goal of the Destroy command is to decrease the health point of the units in the opponent group. The expected outcome of this strategic level command is that the average health point of the opponent group is decreased as much as possible, the ideal outcome is when all of the units in the opponent group are destroyed, meaning that their health point is decreased to zero.

A score is maintained for each of the teams, which is initially zero, and it is increased in every timestep by the number of control points that are owned by a team at the actual timestep.

The teams are the highest level entities in the game from the point of view of the simulator; the actual action processing is done according to Figure 4.



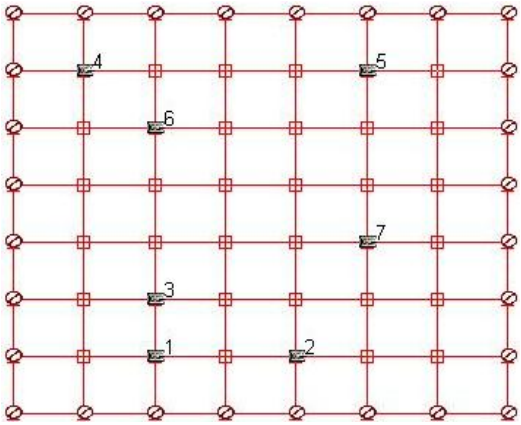
4. figure. Action processing in the simulator

Groups

The maximum number of units in a group was two units. The action list available to a group contained the Left, Right, Forward and Shoot actions. A group level action is sent to a smaller group of units, and it is further broken down to unit level actions. The meaning of these group level actions is:

- Left: The group turns left, meaning that the individual units in the group change their orientation by +90 degrees.
- Right: The group turns right, meaning that the individual units in the group change their orientation by -90 degrees.
- Forward: The group moves forward, meaning that every unit in the team advances one node in the direction of the orientation of the group leader unit. If the orientation of the units is not the same in a group, a group leader is selected and the other units modify their orientation to match the orientation of the leader. If the orientation of every unit in the team is the same, the forward command is executes. The leader of the group is selected by choosing the unit that was first assigned to the group.
- Shoot: Every unit in the group executes the shoot action.

An example partitioning of the units into groups is shown in Figure 4:



5. figure. Example state of the game

The groups created by the group creator module (the group creator module is described in detail in Section 3.3.1) for this state of the game were:

- 1st group: units indexed by 1 and 3, the group leader is unit indexed by 1
- 2nd group: units indexed by 2 and 7, the group leader is unit indexed by 7
- 3rd group: units indexed by 4 and 6, the group leader is unit indexed by 4
- 4th group: the unit indexed by 5, which is also the group leader. Note that there are no more units in this group.

Results

The concrete demonstration game contained two teams, all teams consisting of four units. The teams are placed on opposite sides of the map. The goal of a team is to destroy as many adversary units as possible while keeping as many own units alive as possible. The performance of a strategy is measured by playing against an opponent with a fixed strategy, for example an opponent using random strategy. The performance value is calculated according to the following formula:

$$\text{moreUnit} = \frac{\#\text{own units}}{\#\text{opponent units} + \#\text{own units}}$$

This formula returns relatively high values if the team has more units than its opponent and relatively low values are returned if the opponent has more units. If the number of units in the two teams is the same, the formula returns the neutral 0.5 value (note that values closer to 1 means that the team of the learning agent outnumbers its opponent team, and values closer to 0 means its opposite). The actual value of the performance is dependant of the starting number of units in each of the teams, but changes in the value show the changes in the power relations.

The state of the game is described as the joint state of the individual units (the state of a unit contains its position, orientation and healthpoint) extended with other global features (like the value of the moreUnit feature).

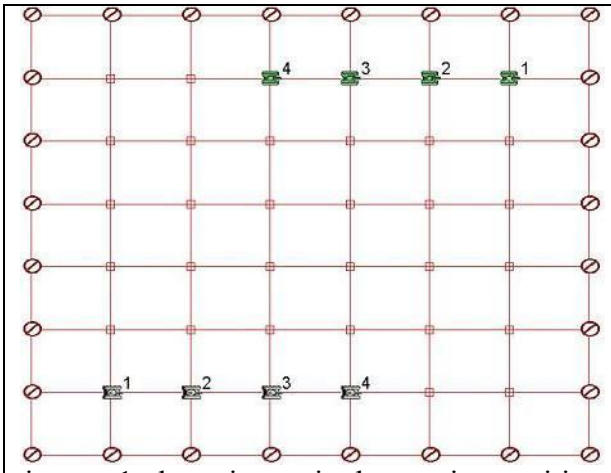
The learning parameters used during learning at the strategic level are shown in Table 1.

name	*value*
α	0.25
δ^l	0.1
δ^w	0.004
β	0.8
λ	0.5
γ	0.99
ε	0.05

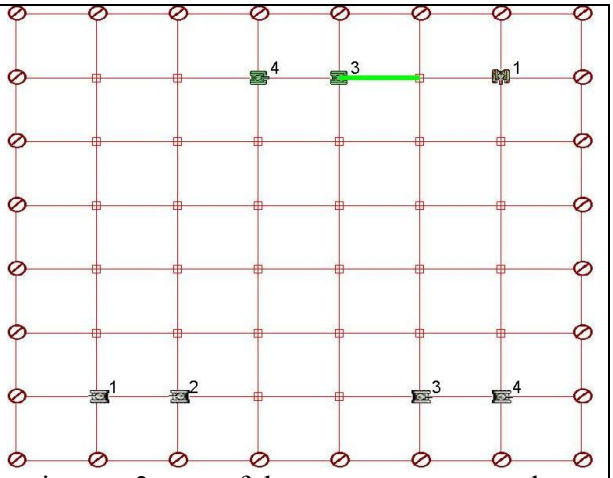
1. table. Parameters of GraWoLF during strategic level learning

The detailed description of the parameters can be found in Section 3.1.

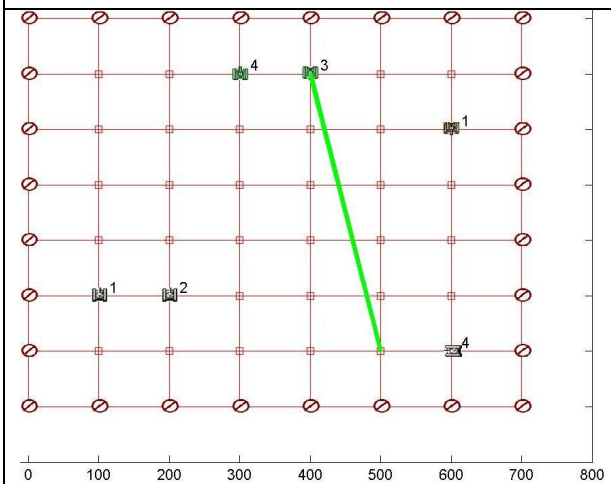
Some snapshots from a battle against a random opponent are shown in Table 2.



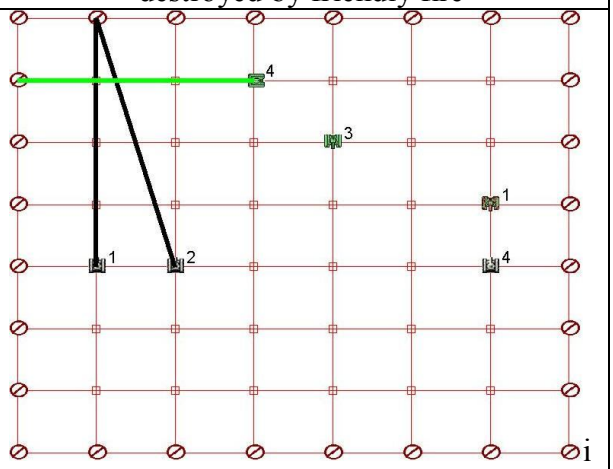
in step 1, the units are in the starting position



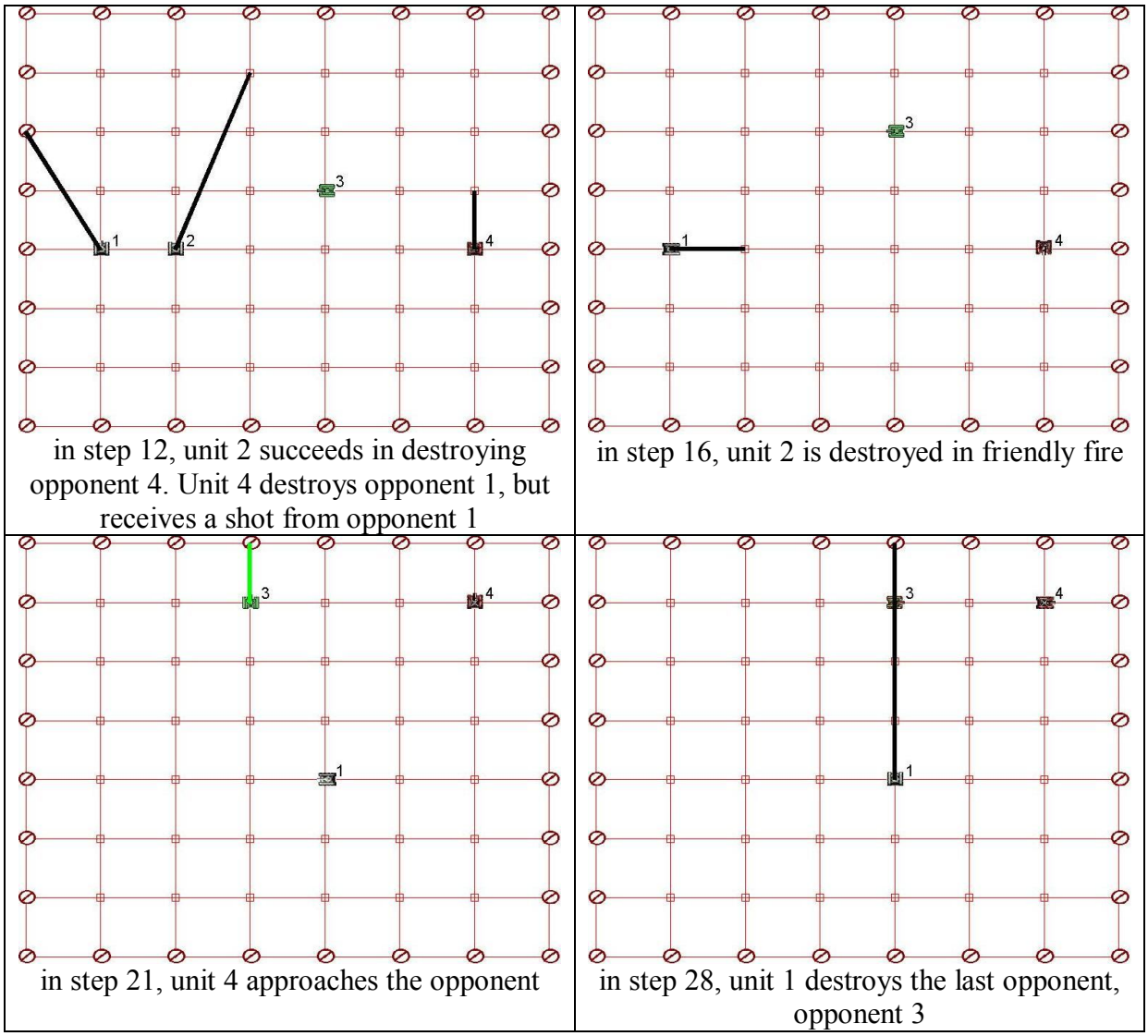
in step 3, one of the groups moves to the right side of the map. An opponent unit is destroyed by friendly fire



in step 7, a unit is destroyed by the enemy. Unit 1 and 2 starts to move upwards



in step 10, unit 1 and 2 are shooting but they hit nothing. Unit 4 approaches opponent 1

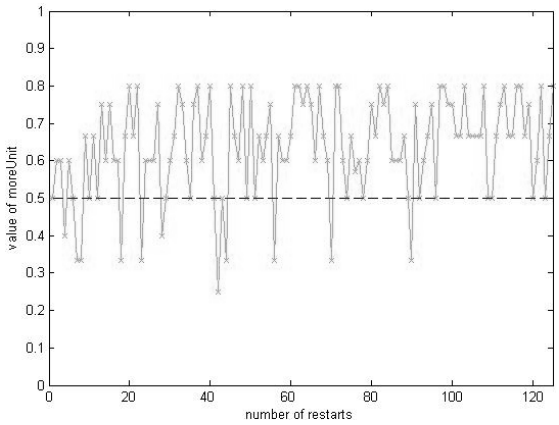


2. table. Battle against a random opponent [10]

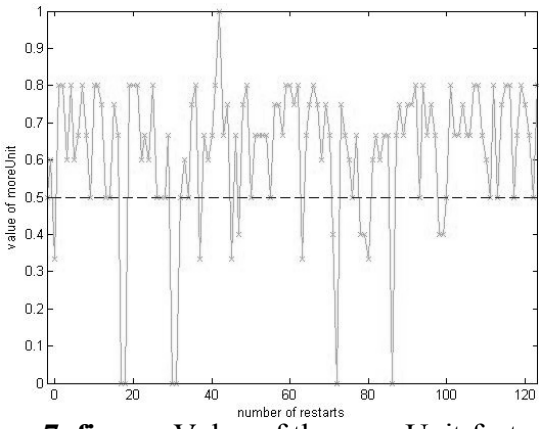
The performance of the learning agent is measured by the moreUnit feature, and as in the initial position the number of units in all teams is the same, the neutral value of the moreUnit feature is at 0.5. Three resulting learning curves are shown in the following figures (figure 6, 7 and 8), two of them correspond to a battle against a random opponent, and one corresponds to a battle against a static opponent.

These learning curves indicate that the learning team agent is able to find a locally optimal strategy against either a randomly acting or a static opponent. A policy is said to be good if it outperforms the initial policy (which is the random policy at the start of the learning sessions).

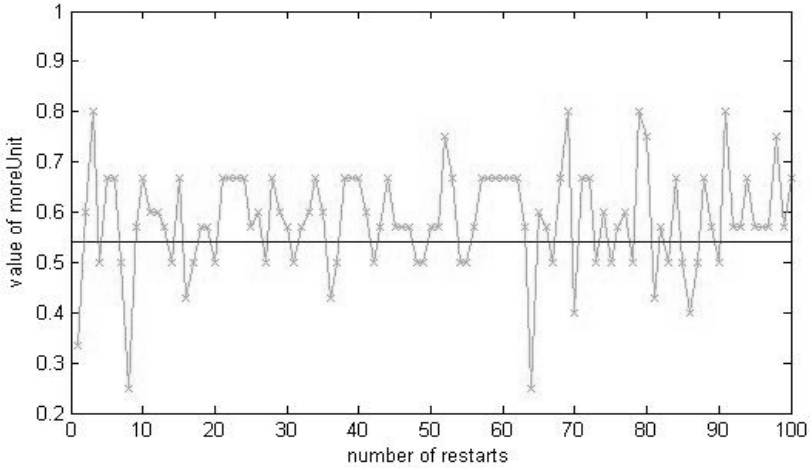
The performance of a final strategy can be measured by the average number of battles that are won by using that strategy. This performance value of a random policy against a random opponent is around 50%. If the learning agent can find a final policy that has a better average winning number, the strategy is better than the initial, which means that the learning session was not useless.



6. figure. Value of the moreUnit feature against a random strategy [10]



7. figure. Value of the moreUnit feature against a random strategy [10]



8. figure. Value of the moreUnit feature against a random strategy [10]

Figure 8:

CONCLUSION

The proposed solution for military operation planning was able to win an average of 60-65% of the games in the demonstrational domain. The solution relies on a spatial relationship between the agents and reduces the complexity of the learning algorithms by defining a hierarchy amongst the agents.

The solution efficiently reduces the complexity of the aggregated agent when there are a limited number of groups defined in a team. The solution assumes that a locally optimal policy can still be found if the communication between the agents is restricted and only the agents in the same group coordinate their actions. The experimental results show that the agent can learn a policy with which it can outperform a rather strong opponent, but in some cases it is possible that the team agent cannot adapt its parameter vector to beat its opponent (this is the case for example when the team agent finds a locally optimal solution that has poor performance).

The original form of GraWoLF is able to find a locally optimal policy, and as this algorithm is used at both levels of learning in the proposed solution, the hierarchical solution guarantees only finding a locally optimal solution. As it can be seen from experiments, if the algorithm converges to a local optimum, it usually get stuck in there and the parameter vector changes only slightly during consecutive learning steps and cannot move away from the local optimum. Therefore the learning has to be restarted several times and strategies with poor performance should be eliminated.

The GraWoLF technique could be improved by using such a technique that is capable of finding a global maximum; and instead of tuning the parameter vector in the direction of the gradient, the global maximum finding method can be used.

The whole learning process could be sped up by reducing the size of the feature vector describing the state of the game, because thus the size of the parameter vector will also be reduced. This can be done by defining fewer features (but this way information will be lost) or by using some compression technique, like hashing [7]. The run time of the learning algorithm can be reduced by doing the hierarchical decomposition less frequently (instead of creating groups in every timestep, the creation of groups can be done in say every ten timesteps, because agents in the same group tend to remain together).

A more intelligent group coordination behavior could be achieved using distributed rewards [8]. This means that a group of agents does not only observe the global reward signal, but they observe also an individual reward. This way the groups will know whether their individual performance is influencing the global reward in a positive or a negative way and they can individually adapt their behavior to increase the local (individual) reward, and thus hopefully increasing the global reward too. If there is only a global reward signal available to the groups in the team, one well performing group can make the others believe that they are performing well too (or one poorly performing group can reduce the global reward in such a way that the other groups believe that they are performing poorly), but by using local rewards this problem is solved.

REFERENCES

[1] I. Harmati, K. Skrzypczyk. Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework. : Robotics and Autonomous Systems 57(1):75-86, 2009.

[2] Y. Liu, M. A. Simaan, J. B. Cruz. An application of dynamic Nash task assignment strategies to multi-team military air operations. : Automatica, 39:1469-1478, 2003.

[3] Russel, S. J., Norvig, P. Artifical Intelligence A Modern Approach. 1995.

[4] R. E. Precup, S. Preitl. Optimisation criteria in development of fuzzy controllers with dynamics. : Engineering Applications of Artificial Intelligence 17(6):661-674, 2004.

[5] von Neumann, J., Morgenstern, O. Theory of Games and Economic Behavior. 1944.

[6] J. B. Cruz, M. A. Simaan, A. Gacic, Y. Liu, Y. Moving horizon game theoretic approaches for control strategies in a military operation. : IEEE Transactions on Aerospace and Electronic Systems 38(3):989-999, 2002.

[7] Bowling, Michael. Multiagent learning in the presence of Agents with Limitations. 2003. Vols. CMU-CS-03-118.

[8] Bagnell, J. A., Ng, A. Y. On Local Rewards and Scaling Distributed Reinforcement Learning. 2005.

[9] Shapley, L.S. Stochastic Games. 1953.

[10] Kisfaludi, Peter. Strategy planning in multiagent robot systems. 2011.

[11] Kok, J. R., Vlassis, N. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. 2006.

[12] T. Basar, G. J. Olsder. Dynamic noncooperative game theory. : SIAM, 2nd edition, 1999.