Decentralized Solutions and Tactics for RTS

Eugenio Bargiacchi   Camiel R. Verschoor   Guangliang Li   Diederik M. Roijers

University of Amsterdam, Informatics Institute

Decision-theoretic control of multiple units in game AI [3, 5] is a notoriously hard problem, because the size of the state and action spaces are exponential in the number of agents, making it an interesting testbed for decision theoretic learning algorithms. There are two main approaches: a centralized and decentralized approach. In the centralized approach, a higher authority governs all agents' actions. However, this approach scales poorly because of the large state and action spaces. In the decentralized approach, each agent selects its own action independently. This has the advantage of reducing the action space, and can even reduce the state space when some features are not relevant for all agents. However, this comes at the expense of losing optimality guarantees.

Figure 1: A battle in StarCraft.

In this research, we use the BroodWar API[1] for the real-time strategy (RTS) game StarCraft (see Figure 1) to micro-manage multiple units in a battle simulation with game AI. More specifically, our problem is a *partially observable stochastic game* (POSG) consisting of two separate, homogeneous groups of units in a zero-sum game. Each group's objective is to defeat the other group in battle. The field of battle is partially obscured to the units, as each agent has a visual range. There is perfect cost-free communication between agents on the same team, but because the field is large the team as a whole cannot observe the whole battlefield. We use the decentralized approach and a novel state representation, and since POSGs are NEXP-complete [1] we need to make simplifying assumptions to keep the problem tractable. We apply model-based reinforcement learning, using Monte-Carlo sampling to learn the transition and reward functions, which leads to good coordination and strong performance.

1 Approach

We take the following steps to simplify the full decision problem. First, we note that the partial observability of the problem is not very significant with respect to the local level at which an agent operates, enabling us to discard the partial observability. Secondly, each agent only models all other visible units, and sees them as part of the environment. Each agent then uses a Markov Decision Process (MDP) [4], to approximate its own decision problem and selects its actions independently.

An MDP is a tuple $< S, A, T, R >$, where S is the set of states representing all possible situations that an agent can face, A the set of actions an agent can take, $T : S \times A \times S \rightarrow [0, 1]$ the transition function that gives the probability of a next state given a state and an action, and $R : S \times A \rightarrow \Re$ the expected reward for taking an action in a given state. At the beginning, the agents know S and A but not T and R. We use a model-based approach [4], i.e., the agents learn T and R explicitly and then plan using the learned model.

Following previous work [2, 5], the actions available to each agent are 'attack' and 'escape'. The 'attack' action either moves towards enemies when the agent is not able to shoot any of them, or shoots the weakest enemy unit currently in range. The 'escape' action moves the agent in such a way to keep it within range of

[1]This API is known from several competitions (SCMAI: `http://scmai.hackcraft.sk` and CIG: `http://eldar.mathstat.uoguelph.ca/dashlock/CIG2013/`) that focus on the improvement of game AI.

its friends, but away from enemies and obstacles. A state transition happens when the unit either moves one tile on the game map, or shoots a single hit towards an enemy.

States are represented as a feature vector containing 'weapon cooldown'[1], which represents if the weapon of the unit is currently in cooldown, 'health'[1], a discretized value representing the remaining health in one of four different intervals of 25% of the total health each, 'target available', a boolean value that represents whether the unit is currently able to target and attack an enemy, 'friend range', a boolean value that represents whether the unit is currently in range of a friendly unit, and 'enemy range', a triplet value that represents whether the unit is in range of an enemy, whether it is targeted by an enemy or neither.

As mentioned earlier, the transition and reward functions are unknown. The transition function is unknown because it depends on the other agents. Rewards are connected to game events, like dealing or receiving damage, but those are not trivially associated with states and actions, making the reward function unknown as well. We apply Monte Carlo sampling to estimate the transition and reward functions using a random policy for our agents, and the standard game AI for the enemy agents. The resulting MDP model is then used by the agents to determine their policies, using the LP planning method from the MDPToolbox[2].

Bots	Wins	Draws	Losses
AA	0	0	1000
JB	50	0	950
MC JB	303	5	692
MC MDP	1000	0	0

Table 1: Outcomes of 1000 games of two dragoons (range) versus one ultralisk (melee) for different policies.

We compare our approach with previous ranged vs. melee results[3] in Table 1 (following [2]). A trivial always attacking (AA) strategy always loses. Jackson and Bogert's dec-POMDP approach (JB) [2] wins sometimes. Adding Monte-Carlo sampling to estimate transitions and rewards to Jackson and Bogert's approach (MC JB) greatly improves performance. Finally, using our MDP approach with Monte-Carlo sampling wins all the time. We therefore conclude that a model-based approach using Monte-Carlo sampling can greatly improve game AI. Furthermore, approximating the problem with an MDP for each agent with the appropriate state space can lead to strong play.

2 Demonstration

In this demonstration we show what decision theoretic control can achieve in RTS games. We use simple units from the game, which requires minimal input to demonstrate the project's features. First, our AI plays against the default StarCraft AI, a video of which can be found here: `http://youtu.be/TrKMBIR82Qw`. Secondly, the visitors get challenged to an interactive game, where they can either play against our AI or against the default AI. The demonstration requires two computers containing Windows (XP/7), with StarCraft Broodwar 1.16 and Broodwar API 4.0.0 installed. The agents were developed by Eugenio Bargiacchi and Camiel Verschoor as a "profile project" for the track Intelligent Systems of the Master AI at the University of Amsterdam, using the BWAPI 4.0.0 Beta library, C++11 and MS Visual Studio 2010.

References

[1] Martin Allen and Shlomo Zilberstein. Complexity of Decentralized Control: Special Cases. In *NIPS*, pages 19–27, 2009.

[2] Christopher Jackson and Kenneth Bogert. An Application of a Dec-POMDP in a Real-time Strategy Game. Online Video, The Universtiy of Georgia, June 2010.

[3] Ricardo Parra and Leonardo Garrido. Bayesian Networks for Micromanagement Decision Imitation in the RTS Game Starcraft. In *Advances in Computational Intelligence*, pages 433–443. 2013.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT press, 1998.

[5] Stefan Wender and Ian Watson. Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft: Broodwar. In *CIG*, pages 402–408. IEEE, 2012.

[1]These are also used by Wender and Watson [5].

[2]`http://www7.inra.fr/mia/T/MDPtoolbox/`

[3]The approach of [5], which we used as inspiration for our state representation, is not comparable because they solve a different problem: a single agent kiting numerous weaker enemies.