15.4 Games

李长河

中国地质大学（武汉）自动化学院

lichanghe@cug.edu.cn

李长河

自动化学院 710

lichanghe@cug.edu.cn

Contents

Games

For AI research, researchers attempt to devise computational systems that would exhibit aspects of human intelligence and achieve human-level problem solving or decision making abilities.

The highly formalized, symbolic representation allowed AI to success in many cases.

Naturally, games, especially board games, have been a popular domain for AI researches as they are formal and highly constrained, yet complex, decision making environments.

There are two main applications of intelligent optimization on games:

- Game playing
- Content generation

Game Playing

For game playing, there are two main ways to use the intelligent optimization:

- Offline optimization: Optimizing the parameters of a pre-defined controller, then using the optimized controller to play the game.
- Online optimization (or evolutionary planning): Optimizing the actions to be deployed while the game is running.

In the offline optimization

The intelligent optimization algorithm can be combined with many other methods. In particular, using EAs to evolve:

- the weights and/or topology of neural networks,
- or programs, typically structured as expression trees (e.g., genetic programming),
- or some other models, such as potential field.

This fitness evaluation consists of using the neural network, program or other models to play the game, and using the result (e.g., score) as a fitness function.

In the online optimization

The basic idea is to optimize the action sequence you want to deploy from now on. Evaluating such an action sequence is done by taking all the actions of the sequence in simulation, and observing the results after taking all those actions.

The online optimization has been applied on several types of games such as:

- arcade game
- turn-based strategy game
- real-time strategy game

Content Generation

Procedural content generation refers to the methods which generate game content either automatically or with only limited human input, such as:

- levels
- maps
- game rules

The search-based methods are widely used in this area because we will eventually get the solution we want if we keep iterating and tweaking solutions by keeping the good changes and discarding the bad changes.

A simple online optimization example

Playing a simple scene in a famous real-time strategy (RTS) game: StarCraft II

The features of a RTS game bring some challenges for AI research, including:

- Incomplete information
- Long term planning
- Large search space
- Real time

Scenario Settings

The figure presents an example map for a game scenario:

Components

- Three Thors and one Marine in our team
- Some Auto-turrets in the enemy team
- Some toxic areas in the middle of the map, which will harm our units

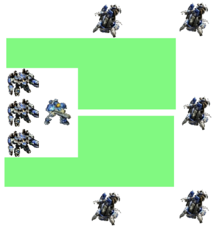

图: An example map: Auto-turrets vs Thors and a Marine, where toxic areas are indicated in green

The focus of this scenario is to test whether the algorithm can find and maintain reasonable tactical positions for different units.

A Multi-objective Approach

- The algorithm runs in the online mode within a given time during the game.
- A solution is selected and deployed when the running time is over.

Components

- Solution representation
- Problem representation
- Evaluation
- Selection

Solution representation

A solution contains the sequence of actions to be performed by all units in the next period of time.

Suppose we have M units, and each unit has C actions, then the j-th action of the i-th unit can be represented as a_{ij}, $i \in \{1, 2, \ldots, M\}$, $j \in \{1, 2, \ldots, C\}$, then a solution s can be represented as:

$$\boldsymbol{x} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1C} \\ a_{21} & a_{22} & \cdots & a_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MC} \end{pmatrix}$$

where a is a 2-tuple, i.e., $a = (actionId, targetPoint)$, $actionId$ represents the action to be deployed and $targetPoint$ determines the target location of the action.

To simplify the problem, we only take the attack action into consideration.

Problem representation

The problem is transformed to a dynamic multi-objective optimization problem, which is shown below:

where t_0 and t denote the starting and ending time of a simulation; M_t and N_t denote the number of units of our team and the enemy team at simulation time t ; h_i and s_i are the health points and shield of unit i; t^{end} is the loop at which the simulation ends when one team has been eliminated.

$$\max f_1(x(t)) = -(\sum_{i=1}^{M_t}(h_i(x(t_0)) + s_i(x(t_0)) - h_i(x(t)) - s_i(x(t))) \tag{1}$$

$$\max f_2(x(t)) = \sum_{i=M_t+1}^{M_t+N_t}(h_i(x(t_0)) + s_i(x(t_0)) - h_i(x(t)) - s_i(x(t))) \tag{2}$$

f_1 and f_2 are the damage of both teams during the simulation, respectively.

Games-A Simple Example of Real-time Strategy Game

where t_0 and t denote the starting and ending time of a simulation; M_t and N_t denote the number of units of our team and the enemy team at simulation time t ; h_i and s_i are the health points and shield of unit i; t^{end} is the loop at which the simulation ends when one team has been eliminated.

$$\max f_3(x(t)) = \begin{cases} -(t^{end}(x(t)) - t_0) & \text{if win} \\ -(t - t_0) & \text{else} \end{cases} \tag{3}$$

$$\max f_4(x(t)) = \sum_{i=M_t+1}^{M_t+N_t} \left(\min_{i\leq j\leq M_t} \mathrm{Dis}\left(p_i(x(t_0)), p_j(x(t_0))\right) - \min_{1\leq j\leq M_t} \mathrm{Dis}\left(p_i(x(t), p_j(x(t)))\right)\right) \tag{4}$$

f_3 is the duration of the game when any team is completely destroyed in the simulation;
f_4 is used to guide our team to move to the target position, especially when our team can not get near enough to fight against the enemy during the simulation as f_1, f_2, f_3 can not obtain valid values (the values of f_1, f_2, f_3 for all solutions are the same of zero if no fight happens).

Considering the importance of the four objectives, the more important objectives f1, f2, f3 could be used first to sort the solutions, then the f4.

So this model can be regarded as a two-level four-objective dynamic optimization problem.

Evaluation

A solution is evaluated through the simulation.

The debug interface of SC2API can be used to call other game processes running simultaneously to copy the current state and the orders in the solution then forward the game to a certain period.

After the evolutionary search, a set of non-dominated solutions may be obtained. However, only one of the solutions $\tilde{x}(t)$ at time will be selected and deployed. A rule-based method is used for the decision-making.

Selection

he solution is selected based on the current total points of both team:

$$\tilde{x}(t) = \arg\max_{x(t)}\left(\frac{f_1(x(t))}{\sum_{i=1}^{M_{t_0}}(h_i(x(t_0)) + s_i(x(t_0)))} + \frac{f_2(x(t))}{\sum_{i=M_{t_0}+1}^{M_{t_0}+N_{t_0}}(h_i(x(t_0)) + s_i(x(t_0))}\right) \tag{5}$$

Games-A Simple Example of Real-time Strategy Game

Results and Discussions

The algorithm can led our Thors approach the enemies along a narrow security area then stand outside the enemies' fire range to attack them. The only Marine roams outside the enemies' fire range.

Results and Discussions

The algorithm can led our Thors approach the enemies along a narrow security area then stand outside the enemies' fire range to attack them.

This is just a simple example with some static enemies.

Results and Discussions

- When there are movable enemies, it needs to handle the uncertainty of the enemy strategy.
- In terms of understanding the enemy, opponent modeling can help.
- If the algorithm run without any priori knowledge, adversarial co-evolution could be a way to generate reasonable solutions.
- To make the solutions more robust, more adaptable for various scenario, robust optimization can be useful.

A special algorithm with some problem-specific search operators, which takes correlation between units and actions into consideration rather than basic EAs, would be more effective in solving this problem.