Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Evolutionary Machine Learning for RTS Game StarCraft

Lianlong Wu, Andrew Markham
University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD
lianlong.wu@acm.org, andrew.markham@cs.ox.ac.uk

Abstract

Real-Time Strategy (RTS) games involve multiple agents acting simultaneously, and result in enormous state dimensionality. In this paper, we propose an abstracted and simplified model for the famous game StarCraft, and design a dynamic programming algorithm to solve the building order problem, which takes minimal time to achieve a specific target. In addition, Genetic Algorithms (GA) are used to find an optimal target for the opening stage.

RTS Game StarCraft Introduction

StarCraft is one of the most classical RTS games which has been immensely popular since its release in 1998. It is based on a military science fiction. Players start with gathering resources: minerals and vespene gas. Resources are used for building, army and technologies. Some buildings enhance the economy, and others are used to train combat units. Advanced unit types are unlocked by specific building or technologies. Units could be sent for attack, defense, patrol or scout all around the map. All moves are "real-time" with 42ms intervals, while the actions are durative and non-deterministic. Furthermore, there is only partial information on the map available to players.

The complexity of RTS games is huge in terms of state-space size and number of actions available at each decision cycle. For a typical board game Go, the total number of states are 10^{170}, while in the StarCraft game, the conservatively estimated number is 10^{1685} (Ontanon et al. 2013).

RTS Game Artificial Intelligence

In the past few years Artificial Intelligence (AI) has been used to design an autonomous system acting as a player of RTS games. StarCraft presents a much greater challenge than traditional board games, due to its complexity and the requirement for simultaneous actions.

The AI application area can be divided into low level micro operations and high level macro strategies. In micro control, it requires quick reaction for attack, retreat, positioning and use of special abilities of advance units. In the macro level, most research has focused on modeling the enemy and predictions.

Building Order Research for RTS Game

For the building order of RTS games, different approaches have been investigated, e.g. Means-End Analysis scheduling (MEA), Best-First Search (BFS) or A* search (Chan et al. 2007). The BTHAI bot uses a fixed building order (Hagelback 2012). The UAlbertaBot uses StarCraft Build Order Search System (BOSS) with Depth First Search (DFS)(Churchill and Buro 2011). These approaches take minutes during the game play, and they do not ensure the best solution.

Dynamic Programming for Building Order

For a certain number of units and buildings, the best building order is the action sequence that uses the least time to complete in game. The time for the first squad is critical to the game results, as the earlier the first attack, the weaker the enemy defense strength, and consequently the more advantages to gain in the opening stage.

We propose a Dynamic Programming method for constructing the optimal building order. Compared with previous heuristics search methods, the DP method has much lower time complexity for computation, so the execution time is reduced significantly.

State Representation and Abstractions

The game state is represented as an integer vector, while the value of each element represents the number of each unit type.

$$S = (Worker, Marine, Medic, Firebat, Centre, \\ Barrack, Bunker, Depot, Academic) \quad (1)$$

The state space of StarCraft is enormous, and modeling is impossible without certain abstraction and simplification. The total number of frames is 14400 for the opening stage that takes about 10 minutes in StarCraft.

We define constant vector $r_{mine}(x)$ representing the rate of *mine* gathered when there are x *Worker* units. The figures are predefined based on measurements in the real game plays. There is no resource dimension required in this representation, so that total states number is reduced.

Resource Constraints: Define static function $T_{build}(t)$ representing the building time required for unit type t.

Define function $C_{mine}(S)$ as the total number of *mine* required for building up all the units in state S, similarly $C_{gas}(S)$ is the total number of *Gas* required.

Time Constraints: Define $T_{mine}(S)$ as time required for gathering all the Mine required to build units in state S, similarly $T_{gas}(S)$ is the time required to gather all the gas.

Supply Constraints: In the game, one *Depot* building provides 8 supply spaces. For each unit type t, one unit requires $Supply(t)$ spaces. Define $SupplyConstraint(S)$ as

$$S(Depot) \times 8 \geq \sum_{\forall s \in S, s \neq Depot} Supply(s) \times S(s) \quad (2)$$

Precondition Constraints: Most unit type t has a preceding unit type p, and the player must have had p before starting building t.

Define static function $Requires(t)$ representing the pre condition unit type before building unit type t. Define $RequireConstraint(S)$ as $\forall t \in S \implies Require(t) \in S$.

Concurrent Production Constraints: At any time, there is only one unit training in one building.

State Transition Equation: Define function $F(S)$ representing the minimal time required to achieve the state S.

$$\begin{aligned} F(S) = \min_{\forall s \subsetneq S} \quad & \max(F(S-s) + T_{build}(s), T_{mine}(S), T_{gas}(S)) \\ \text{s.t.} \quad & F(S) \leq T_{max} \\ & SupplyConstraint(S) \\ & RequireConstraint(S) \\ & ConcurrentConstraint(S, s) \end{aligned} \quad (3)$$

The time complexity of this DP method is $O(nm)$, where n is the number of total states, and m is the average number of state variations for each state. The Python prototype takes 75.67s (wall clock time) to visit all the 13576 states, which can be completed off-line before game starts.

Optimal Building Order Retrieval: The previous state is stored for each state S in $Prev(S)$, so that the exact building order could be retrieved in the end for any specified state, by tracing backwards from end state to the initial state. The differences between two states are the unit to be built at that step. The time complexity is $O(n)$. It takes only 0.09s for one building order retrieval.

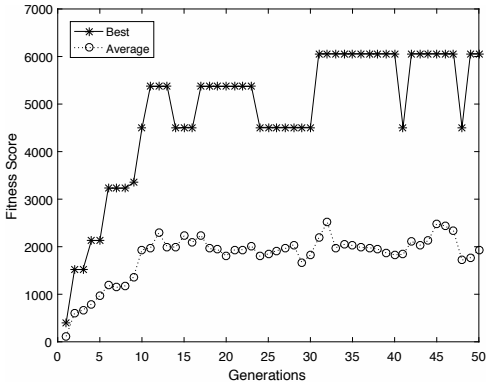
Comparison with Fixed Building Order For building up a target squad (8 *Marines*, 3 *Medic* and 1 *Firebat*) in a fixed map, it takes 497.95s for the BTHAI bot v3.3 with its fixed building order "Marine Rush". By applying the DP building order, it takes only 432.45s, 65.5s faster than fixed building order.

Machine Learning using Genetic Algorithm

In order to find the target state for the DP model, Genetic Algorithms (GA) are used for evolutionary learning. The chromosome is the same state representation as DP. The in-game fight score is used as fitness score. Crossover operator and mutation operator are used for evolving the population.

We present a distributed learning platform utilizing large scale cloud computing infrastructure[1], with fully automated deployment ability, fault tolerance and high availability design, to achieve a machine learning framework with 1 to 1000s Virtual Machines running simultaneously. This system is capable of performing training for 250,000 game plays within 3 hours, instead of 3 weeks in similar works.

As shown in the figure below, the obvious ascending fitness scores from a randomly generated initial population (zero prior knowledge). From the experimental results, the best army composition is 15 *Marines*, 7 *Medics* and 1 *Firebat*. It is inline with human experiences, in terms of the ideal 2:1 ratio of Marines to Medics, and the single *Firebat* effectively eliminates enemy type *Zergling*.



Conclusion and Future Work

In this paper we present an abstracted model and DP method for optimal building orders in StarCraft. We have shown its run time is significantly reduced compared with previous heuristics methods, and the optimal building order performs better than the fixed ones in the real game play.

In the experiments, GA effectively shows increasing fitness scores, and finds an optimized squad composition, which is comparable to professional human players.

This model can be extended to later stage of the game. A similar approach may be applied to other RTS games, or AI system for wider areas.

References

Chan, H.; Fern, A.; Ray, S.; Wilson, N.; and Ventura, C. 2007. Online planning for resource production in real-time strategy games. In *ICAPS*, 65–72.

Churchill, D., and Buro, M. 2011. Build order optimization in starcraft. *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Hagelback, J. 2012. Potential-field based navigation in starcraft. In *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*.

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. *IEEE Transactions on Computational Intelligence and AI in games* 293 – 311.

[1]Sponsored by Microsoft through Azure for Research Award.