An Improved Dynamic Algorithm for Google's AI AlphaStar

Authors: Guo Feng, Guo Feng

Date: 2024-08-10T00:00:00+00:00

Abstract

[Objective] Analyze the reasons for AlphaStar's losses against human players in StarCraft II and propose an improved strategic algorithm. [Methods] Propose a dynamic strategic algorithm that adapts its strategy according to the opponent's strategy; design a default strategy method that randomly selects default strategies to avoid opponent anticipation; implement jiahuibot code to verify the feasibility and effectiveness of the strategies, and provide theoretical guidance for multiplayer battle methodologies. [Results] The dynamic strategic algorithm and default strategy algorithm enhance artificial intelligence capabilities in real-time strategy games; experiments with jiahuibot validate the feasibility of the strategies, providing implementation guidance for AlphaStar and other StarCraft AI researchers. [Limitations] The research is limited to strategic algorithms within the StarCraft II game and does not address broader real-world applications. [Conclusion] The research improves AI effectiveness in real-time strategy games and provides directions for future research.

Full Text

Abstract

[Objective] This paper analyzes the reasons for AlphaStar's defeat against human players in StarCraft II and proposes improved strategic algorithms. **[Methods]** We introduce a dynamic strategy algorithm that adapts to the opponent's strategies in real-time, design a default strategy mechanism that randomly selects actions to avoid predictability, implement a demonstration bot named jiahuibot to verify the feasibility and effectiveness of the proposed strategies, and provide theoretical guidance for multiplayer gameplay methodologies. **[Results]** The dynamic strategy algorithm and default strategy mechanism contribute to enhancing AI capabilities in real-time strategy games. The jiahuibot experiments demonstrate the feasibility of these strategies, offering implementation guidance for AlphaStar and other StarCraft AI developers. **[Limitations]**

The research is limited to strategic algorithms within the context of StarCraft II and does not address broader real-world applications. **[Conclusions]** This study contributes to improving AI competitiveness in real-time strategy games and provides directions for further research to refine dynamic strategy algorithms.

Keywords: AI, machine learning, AlphaStar, StarCraft, AlphaGo
Classification Number: TP393

Following DeepMind's victory with AlphaGo over human Go champions, the company created AlphaStar to conquer StarCraft II, a complex real-time strategy game, using artificial intelligence. While AlphaStar achieved notable success, it ultimately lost to human player Mana in their final live-streamed match. The decisive battle between the main forces is illustrated in [Figure 1: see original paper].

1 Introduction

1.1 Analysis of AlphaStar vs. Mana Strategies

This paper examines the strategies employed by both AlphaStar and the human top-player Mana, summarized in . AlphaStar utilized a composition of Oracles and Stalkers, using Oracles to harass enemy workers while waiting to amass a sufficient Stalker force before engaging in a decisive battle. Its strategy remained static throughout the match. In contrast, Mana adapted his strategy mid-game: after scouting that AlphaStar was massing Stalkers, he transitioned to producing Immortals and Zealots that counter Stalkers, and created Twilight Archons to compensate for anti-air capabilities. He also employed a drop of two Immortals to harass AlphaStar while focusing attacks on mineral lines.

Prior to the live match, Mana had privately played five games against AlphaStar, with both sides primarily using Stalker-based armies and Mana winning all encounters. In the sixth game (the live broadcast), AlphaStar lost. This pattern demonstrates that the more human player Mana faced AlphaStar, the higher his win probability became, while AlphaStar's performance deteriorated—highlighting its inferior learning and adaptation capabilities compared to humans.

AlphaStar's primary failure stems from its static strategy algorithm, which fixes a single strategy at the game's onset without adapting unit composition based on enemy forces. Moreover, no fixed strategy can defeat all opponents because unit types form a rock-paper-scissors balance, creating counterplay loops.

1.2 Dynamic Strategy Algorithm

To address these limitations, this paper proposes a dynamic strategy algorithm, illustrated in [Figure 2: see original paper]. This approach effectively adapts our strategy based on enemy tactics. What units or buildings we produce at

any given time depends on our policy, which in turn depends on the opponent's strategy. The opponent's strategy is determined by scouting information about their structures, army composition, and unit counts.

1.3 Multiplayer Battles

AI multiplayer battles currently lack robust execution environments and comprehensive research literature. This paper provides a preliminary discussion of multiplayer strategies to guide code implementation and establish a theoretical foundation for future research.

1.4 Contributions

This paper makes three primary contributions: (1) We propose a dynamic strategy that adjusts our tactics based on enemy strategy, enabling counter-unit production to effectively defeat opponents; (2) We introduce a method of maintaining multiple default strategies and randomly selecting among them to prevent predictability; and (3) We present an efficient multiplayer battle strategy to guide implementation and provide theoretical groundwork for subsequent multiplayer research.

2 Related Work

2.1 StarCraft AI Programming Interfaces

We are grateful to Oriol Vinyals and 18 DeepMind colleagues, along with Kevin Calderone and 7 Blizzard employees, for providing the interface and codebase that enables AI implementation and automated gameplay in StarCraft II [Figure 3: see original paper][1].

Due to StarCraft II's complexity, strategies effective on small maps often fail against human players on larger maps. DeepMind's AlphaStar employed reinforcement learning from extensive human replay data to develop several strategies, then had these strategies play against each other extensively to discover an optimal approach.

2.2 Unit Counter Relationships

In reality, no single unit type can defeat all others. As shown in [Figure 4: see original paper], unit types exhibit mutual counter relationships, making an undefeated strategy impossible. Many StarCraft AIs, like AlphaStar, maintain several fixed strategies and randomly select one when facing a new opponent, reusing successful strategies in subsequent matches. However, their strategies remain static throughout individual games.

This approach has significant flaws. If opponents identify your strategy, they can always find a counter, given the rock-paper-scissors nature of unit relationships shown in [Figure 4: see original paper]. Our proposed dynamic strategy

algorithm architecture is illustrated in [Figure 5: see original paper]. Current production decisions depend on our strategy, which depends on enemy strategy, which must be calculated through scouting enemy structures and armies. When the enemy strategy is unknown, our default strategy randomly selects from two or more options to prevent pre-game prediction.

3 Dynamic Strategy Algorithm

3.1 Core Algorithm

The dynamic strategy algorithm operates according to the following relationships:

$$produce(t) = myPolicy(t) = f(enemyPolicy(b))$$
$$enemyPolicy(b) = f(enemyStructure(b), enemyArmy(b))$$

Our policy $myPolicy(t)$ depends on the opponent's previous strategy $enemyPolicy(b)$, which in turn depends on scouted enemy buildings and army composition.

Based on these formulas, we propose specific battle strategies summarized in . The table predicts enemy strategies based on observed buildings and forces, then suggests counter-unit production. When enemies field multiple unit types, we respond with corresponding mixed compositions. We implemented a demonstration program called jiahuibot that transitions to producing Thors when detecting enemy Spire construction, rather than continuing with the original tank production.

3.2 Initial Strategy Selection

Before identifying the enemy strategy, we require an initial policy with at least two or more options selected randomly. If opponents know our predetermined strategy, they can easily prepare counters. This paper proposes two Terran default strategies, detailed in , with random selection at each game start. We implemented this in our jiahuibot demonstration AI.

3.3 Multiplayer Combat Strategies

Currently, AI multiplayer battles lack robust environments and research literature. This section discusses multiplayer strategies to guide implementation and provide theoretical foundations for future work. Classic formats include 2v2 (two players versus two opponents), 3v3, and 4v4. outlines multiplayer strategies and implementation guidance.

Key implementation suggestions include modifying the command line from two-AI to four-AI battles: changing `python run_{custom}.py -m SubmarineLE -p1 protossbot -p2 tank` to `python run_{custom}.py -m`

`SubmarineLE -p11 4gate -p12 zealot -p21 protossbot -p22 tank`. Communication methods include: (1) Sending text messages in allied chat, such as "11" meaning "I will attack the enemy at 11 o'clock position," or "rush" indicating a rush strategy requiring ally cooperation; (2) Sharing army positioning and coordination through chat or proximity. The sc2 library's `BotAI` includes `await self.chat_{send}(message="11", team_{only}=True)` for messaging, requiring additional message handling interfaces like `on_{{{received}}_{{chat}}}`. For 3v3 and 4v4, the same principles apply, with command lines extended to include additional bots.

4 Experiments

4.1 Experimental Purpose

The primary goal is to demonstrate the feasibility of our proposed dynamic strategy—specifically, that our algorithm can change strategies based on enemy tactics. Our jiahuibot demonstration program battles Mutalisk strategies twice and faces the hardest AI with random races.

4.2 Demo Environment

The experiments run on Windows 7 64-bit with 16GB RAM and 2GB VRAM. Using PyCharm, open the sharpy-starter-bot folder with Python 3.7 and execute the command lines in .

4.3 Experimental Process

Our implementation employs a Cyclone strategy, continuously producing Cyclones until detecting an enemy Spire, then transitioning to Thor production, as shown in [Figure 6: see original paper]. Additional strategy execution results are documented in .

When our scout detects enemy Spire construction, jiahuibot transitions to Thor production rather than continuing tanks. Without this dynamic adaptation, tanks cannot attack air units and would inevitably lose to Mutalisks. Conversely, without a Spire, jiahuibot continues tank production. The experiments also demonstrate successful random selection from default strategies, with logs and early-game army compositions in the recorded videos showing random strategy selection.

4.4 Results

The proposed strategies are implementable and demonstrate strong competitive performance.

5 Conclusion and Outlook

This paper thoroughly analyzes AlphaStar's limitations and proposes a dynamic strategy algorithm for improvement. We implemented a demonstration program, conducted experiments, and published recorded videos. Our jiahuibot randomly selects between tank-heavy and Cyclone-heavy strategies at game start, then transitions to Thor production upon detecting enemy Spire construction to counter Mutalisks. Additionally, we propose implementation methods for multiplayer AI battles to guide Blizzard and other developers in creating multiplayer AI interfaces.

Future work will focus on developing strategies for Zerg and Protoss races, expanding the strategy pool, and refining counter-tactics.

We thank DeepMind and Blizzard for providing StarCraft AI programming interfaces and AlphaStar, the sharpy-sc2 open-source project and its authors for Python interfaces and examples, and the sc2ai battle network for providing the StarCraft AI competition platform.

References

[1] Oriol Vinyals… StarCraft II: A New Challenge for Reinforcement Learning. 2017.8.16. https://arxiv.org/abs/1708.04782
[2] Vinicius Zambaldi… DEEP REINFORCEMENT LEARNING WITH RELATIONAL INDUCTIVE BIASES. 2019. https://openreview.net/pdf?id=HkxaFoC9KQ

Author Contributions

Guo Feng: Conceived research ideas, designed research methodology, conducted experiments, drafted manuscript, revised final version.

Corresponding Author: Guo Feng, E-mail: forky@126.com

Note: Figure translations are in progress. See original paper for figures.