Aprendizado por Reforço sobre Algoritmos em Jogos de Estratégia em Tempo Real

Marcelo Luiz H. D. Lemos[1], **Luiz Chaimowicz**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG

{marcelolemos,chaimo}@dcc.ufmg.br

Resumo. *Jogos de estratégia em tempo real são bastante desafiadores para o aprendizado por reforço devido aos seus grandes espaços de estados e ações. No entanto, existem várias estratégias que foram desenvolvidas com o objetivo de selecionar a melhor ação dado um estado. Neste trabalho, vamos avaliar o desempenho de um agente que realiza seu aprendizado sobre estas estratégias colocando ele para jogar contra adversários bem consolidados de microRTS.*

1. Introdução

Aprendizado por Reforço (RL) é uma abordagem para se entender e automatizar aprendizado direcionado a objetivos e tomada de decisões [Sutton and Barto 2018]. Nesta abordagem, o agente deve aprender como se comportar, por meio de tentativa e erro, através de interações com o ambiente.

Uma das limitações atuais do aprendizado por reforço é que à medida que o conjunto de estados e ações cresce, o agente tende a ter dificuldades em manter uma boa performance. Por outro lado, existem diversos domínios que possuem algoritmos que foram desenvolvidos especificamente para solucionar o problema em mãos, e um agente poderia delegar esses algoritmos para agir em seu lugar [Rice 1976].

Uma forma de tentar minimizar o impacto desta limitação é realizar o treinamento do aprendizado por reforço sobre algoritmos, a fim de determinar qual algoritmo é capaz de obter o melhor resultado em cada estado do ambiente. Esta estratégia reduz o tamanho do espaço de ações que o agente precisa explorar, facilitando o aprendizado em problemas complexos. Mesmo assim, RL ainda pode ter dificuldades quando o conjunto de estados possíveis é muito grande, e um dos exemplos disto são os jogos de *Estratégia em Tempo Real* (RTS), que são especialmente desafiadores na área de *Inteligência Artifical* [Ontanón et al. 2013]. Para esta segunda parte do problema, podemos utilizar funções aproximativas, que identificam característcas comuns de diferentes estados e os agregam em um mesmo grupo, fazendo com que o aprendizado realizado em algum estado possa ser generalizado para outros semelhantes com facilidade.

A proposta deste trabalho é verificar, por meio de experimentos práticos, qual o desempenho de um agente que se utiliza destas estratégias quando colocado para jogar contra inimigos do estado da arte de um jogo RTS.

2. Referêncial Teórico

Aprendizado por reforço é um dos paradigmas básicos de *aprendizado de máquina*. O framework de *Processos de Decisão de Markov* (MDP) é utilizado para modelar, de

forma simplificada, o RL em termos de estados, ações e recompensas. Referências sobre seus modelos, algoritmos, aplicações, e limitações podem ser encontrados em [Sutton and Barto 2018].

A questão de se realizar aprendizado por reforço sobre ações ou sobre algoritmos é abordada em [Tavares et al. 2018], que é a principal referência para este trabalho. Neste artigo, o aprendizado sobre algoritmos é explorado a fim de se estabelecer quando ele pode ser útil, quais as condições necessárias para seu correto funcionamento, e suas limitações em relação ao aprendizado sobre ações.

O uso de jogos para a realização de pesquisas em IA e seus desafios já vêm sendo explorados a bastante tempo, e podemos ver em [Bellemare et al. 2013] como o aprendizado por reforço profundo pode ser utilizado para solucionar os jogos do clássico Atari utilizando o *Atari Learning Environment*. Além disto, vários trabalhos e desafios quanto uso de IA em RTS são abordados em [Ontanón et al. 2013].

Algoritmos (ou scripts) são amplamente utilizados em pesquisas envolvendo jogos RTS. Alguns exemplos são os seguintes trabalhos: AHTN [Ontanón and Buro 2015] que combina *hierarchical-task network* (HTN) com um algoritmo semelhante a uma arvore de busca minimax. PuppetSearch combina comportamentos de scripts com *game-tree search*. Existem duas variações do PuppetSearch: PuppetAB [Barriga et al. 2015] e PuppetMCTS [Barriga et al. 2017b]. StrategyTactics [Barriga et al. 2017a] utiliza uma rede neural convolucional para prever a saída do PuppetSearch, aumentando o tempo que um algoritmo de busca tática tem para executar. NaiveMCTS [Ontanón 2017] aplica *Monte Carlo Tree Search* mas utiliza uma estratégia de amostragem baseada em *multi-armed bandits*.

3. Metodologia

3.1. Ambiente

Este trabalho foi realizado no ambiente do *microRTS*[1], que é um RTS simplificado, voltado para pesquisas, mas que mantem a complexidade de um RTS tradicional.

3.2. MetaBot

O agente utilizado neste trabalho é chamado de *MetaBot*[2], e foi desenvolvido pelo grupo de pesquisa responsável por [Tavares et al. 2018]. Ele utiliza o algoritmo *SARSA* para realizar seu aprendizado por reforço sobre 6 estratégias (scripts) simples de microRTS:

- **Worker:** cria unidades *Worker* (unidade básica), um deles coleta recursos enquanto o resto é enviado para combate.
- **Ranged:** utiliza um *Worker* para coletar recursos. Com recursos suficientes, constrói *Barracks* (edifício que permite a criação de unidades militares), cria unidades *Ranged* (soldados de longo alcance) e as envia para combate.
- **Light:** semelhante à Ranged, mas cria unidades *Light* (soldados leves).
- **Heavy:** semelhante à Ranged, mas cria unidades *Heavy* (soldados fortes).
- **BuildBarracks:** construir uma nova *Barracks*, permitindo uma produção mais rápida de unidades militares.

[1]https://github.com/santiontanon/microrts

[2]https://github.com/andertavares/micrortsMetaBot

- **Expand:** construir uma nova base, aumentando a produção de unidades *Worker* e acelerando a coleta de recursos.

Além disto, este bot também agrupa estados semelhantes dividindo o mapa do jogo em quadrantes e utilizando características destes quadrantes (como quantidade de unidades aliadas e inimigas no quadrante, vida total das unidades aliadas e inimigas no quadrante, etc) para gerar um vetor de *features* que será utilizado para determinar o estado do jogo.

Algumas pequenas modificações foram realizadas no MetaBot com o único objetivo de facilitar a execução dos experimentos.

3.3. Adversários

Os seguintes bots do estado da arte de microRTS foram utilizados para avaliarmos o Metabot:

- AHTN [Ontanón and Buro 2015]
- NaiveMCTS [Ontanón 2017]
- PuppetAB [Barriga et al. 2015]
- PuppetMCTS [Barriga et al. 2017b]
- StrategyTactics [Barriga et al. 2017a]

Vários destes oponentes alcançaram posições de destaque em competições, como o StrategyTactics que ganhou o campeonato de microRTS em 2017 e o NaiveMCTS que ficou entre os 5 melhores desta mesma disputa.

3.4. Experimentos

Dois conjuntos de exprimentos foram executados. No primeiro, chamado de Específico, o agente é treinado por 500 jogos contra AHTN, PuppetAB e PuppetMCTS; e por 100 jogos contra NaiveMCTS e StrategyTactics. A política resultante de cada um destes treinos é então testada, ao longo de 100 jogos, contra o mesmo adversário na qual foi treinada. O segundo, chamado de *Nemesis*, consiste em três etapas:

1. MetaBot é treinado contra PuppetMCTS por 500 jogos e a política resultante é fixada.
2. Uma nova instância do MetaBot é treinada contra a política resultante da etapa anterior durante 500 jogos.
3. A política resultante do treinamento em 2 é testada contra cada um dos adversários em 100 jogos.

Estes experimentos são uma réplica de parte dos experimentos realizados em [Tavares et al. 2018].

4. Resultados

Cada um dos experimentos descritos na seção anterior foram executados 5 vezes e os resultados abaixo são baseados na média obtida.

4.1. Específico

O desempenho do MetaBot contra cada um dos adversários no experimento Específico pode ser visto na Figura 1.

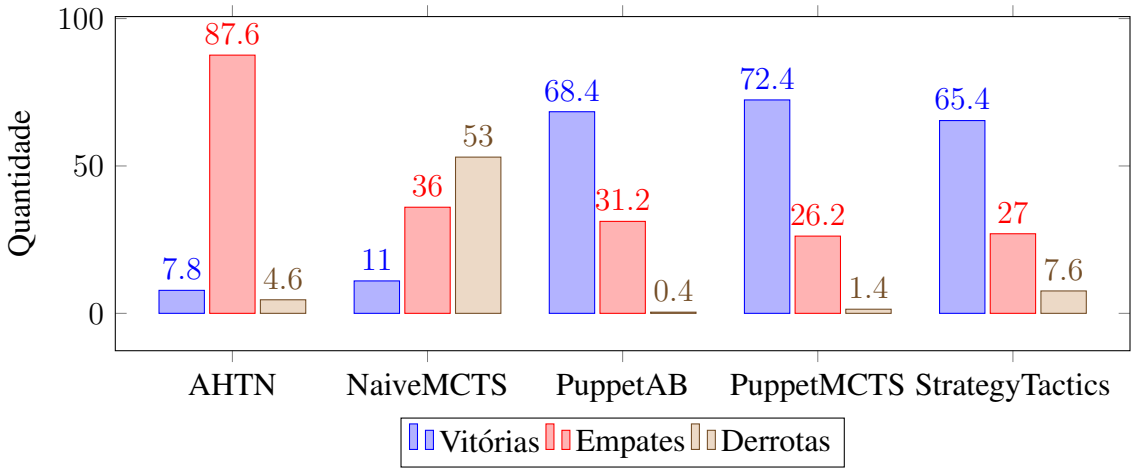

Figura 1. Quantidade de Vitórias, Empates e Derrotas do MetaBot contra cada um dos adversários no experimento Específico

A taxa de vitórias do MetaBot neste experimento foi bastante satisfatória contra PuppetAB, PuppetMCTS e StrategyTactics, enquanto os jogos contra AHTN terminaram, em sua grande maioria em empates. Já contra o NaiveMCTS, vemos um grande número de derrotas com poucas vitórias e vários empates, um resultados aquém do que gostaríamos obter.

4.2. Nemesis

O desempenho do MetaBot contra cada um dos adversários no experimento Nemesis pode ser visto na Figura 2.

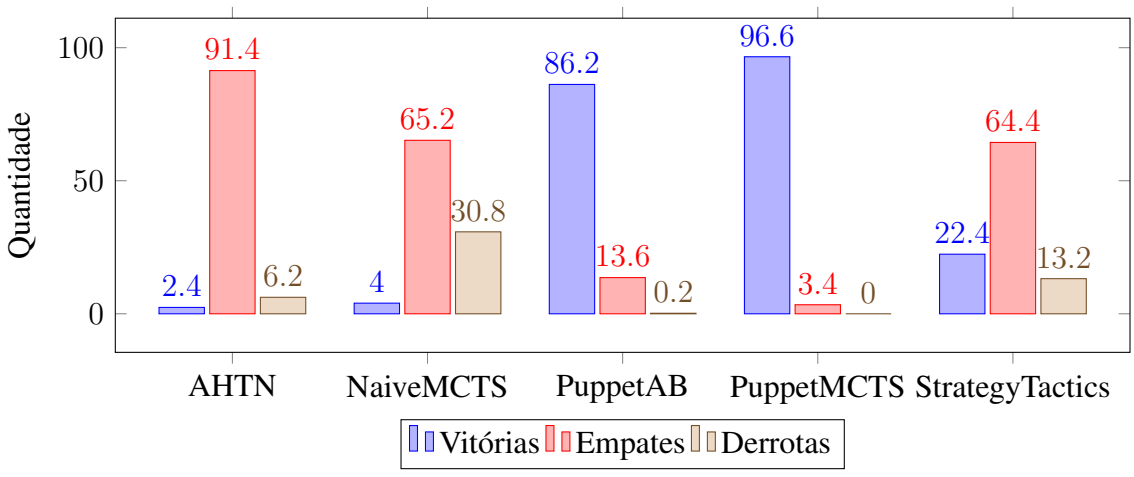

Figura 2. Quantidade de Vitórias, Empates e Derrotas do MetaBot contra cada um dos adversários no experimento Nemesis

Tivemos algumas diferenças interessantes nos resultados deste experimento em relação ao anterior. O MetaBot teve um menor número de derrotas contra o NaiveMCTS, mas isto não representa uma grande melhoria, principalmente porque sua taxa de vitórias também foi reduzida e os resultados agora são predominantemente empates. A taxa de

vitórias contra o StrategyTactics caiu drasticamente em relação ao resultado anterior mas mesmo assim nosso agente conseguiu manter uma taxa de vitórias positiva. Tanto contra o PuppetAB quanto contra o PuppetMCTS, o MetaBot alcançou resultados incríveis e praticamente não perdeu para eles.

4.3. Score Percentage

A métrica *Score Percentage* foi utilizada para realizar uma avaliação geral do desempenho de nosso agente nos experimentos, ela é calculada da seguinte maneira:

$$ScorePercentage = \frac{Vitorias + 0.5 * Empates}{Total\ de\ Jogos}$$

Isto nos ajuda a avaliar o MetaBot levando em consideração tanto a taxa de vitórias quanto a taxa de empates, o que nos permite ver se ele foi melhor, pior, ou do mesmo nível que seu adversário. A Figura 3 mostra o Score do MetaBot contra os diferentes adversários.

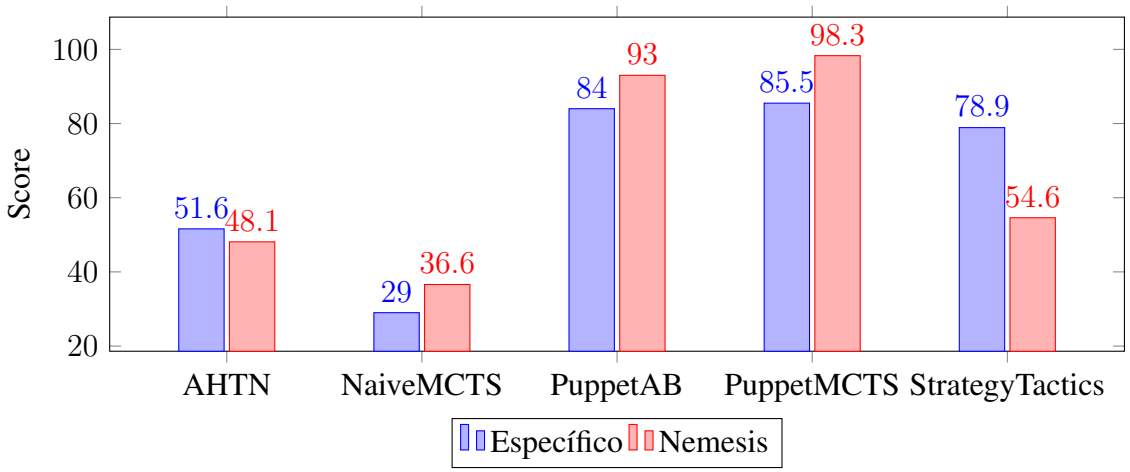

Figura 3. Score do Metabot contra cada um dos adversários nos experimentos realizados

Podemos ver que o desempenho do MetaBot contra PuppetAB, PuppetMCTS e StrategyTactics foram muito bons, mas contra o AHTN ele conseguiu apenas se igualar a seu adversário. O resultado encontrado contra o NaiveMCTS está bem abaixo do que esperávamos e será investigado com mais detalhes em trabalhos futuros.

5. Conslusão

O aprendizado por reforço sobre algoritmos apresentou resultados promissores e conseguimos alcançar um desempenho próximo do que foi utilizado como referência [Tavares et al. 2018] contra alguns dos adversários (AHTN, NaiveMCTS e StrategyTactics). Apesar disto, ele se mostrou não ser imbatível já que o NaiveMCTS acabou conseguindo muitas vitórias contra ele nos experimentos e, portanto, existe espaço para melhorias nesta abordagem.

Um ponto importante que pode ser melhorado é a função utilizada para aproximar estados semelhantes. Como ela leva em consideração apenas características gerais de

cada um dos quadrantes, ela pode estar agrupando estados com importâncias estratégicas bastante distintas, o que pode acarretar em tomadas de decisão ruins.

Outra coisa interessante a ser feita é realizar os mesmos experimentos com o aprendizado sobre ações, o que nos permitirá realizar uma comparação direta do desempenho das duas abordagens. Poderíamos até mesmo realizar experimentos onde o aprendizado sobre algoritmos é colocado para jogar contra o aprendizado sobre ações.

Referências

[Barriga et al. 2015] Barriga, N. A., Stanescu, M., and Buro, M. (2015). Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Barriga et al. 2017a] Barriga, N. A., Stanescu, M., and Buro, M. (2017a). Combining strategic learning with tactical search in real-time strategy games. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Barriga et al. 2017b] Barriga, N. A., Stanescu, M., and Buro, M. (2017b). Game tree search based on nondeterministic action scripts in real-time strategy games. *IEEE Transactions on Games*, 10(1):69–77.

[Bellemare et al. 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Ontanón 2017] Ontanón, S. (2017). Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research*, 58:665–702.

[Ontanón and Buro 2015] Ontanón, S. and Buro, M. (2015). Adversarial hierarchical-task network planning for complex real-time games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Ontanón et al. 2013] Ontanón, 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*, 5(4):293–311.

[Rice 1976] Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier.

[Sutton and Barto 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Tavares et al. 2018] Tavares, A. R., Anbalagan, S., Marcolino, L. S., and Chaimowicz, L. (2018). Algorithms or actions? a study in large-scale reinforcement learning. In *IJCAI*, pages 2717–2723.