UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

RELATÓRIO PARCIAL DE BOLSA DE INICIAÇÃO CIENTÍFICA
EDITAIS PIC&DTI, PIPES E PIBIC-EM Nº 01/2022 (CICLO 2022-2023)

Título do Projeto de Pesquisa do Orientador: Aprimorando o Desenvolvimento de Simulações Baseadas em Agentes por meio de Blocos de Construção Abstratos
Orientador: Fernando dos Santos
Bolsista/Estudante IC: Eloísa Bazzanella
Modalidade de Bolsa: PIPES
Vigência das atividades de IC como bolsista neste edital: **Data de Início:** 09/2022 **Data Fim:** 12/2022

Resumo dos principais tópicos desenvolvidos:

Refatoração da extensão QLearning do NetLogo para uso da biblioteca BURLAP.

Disponibilização de outros dois algoritmos de aprendizagem por reforço na extensão NetLogo: SARSA e Actor-Critic.

Validação da refatoração da extensão NetLogo que agora utiliza a BURLAP por meio de um estudo de caso com o cenário *Cliff Walking*.

Revisão bibliográfica efetuada:

ALGORITMOS DE APRENDIZAGEM POR REFORÇO

O método de aprendizagem por reforço mais popular é o Q-Learning (DAYAN; WATKINS, 1992). É um algoritmo para estabelecer autonomamente uma política de ações de maneira iterativa. É possível demonstrar que o algoritmo Q-Learning converge para uma política ótima, a partir do momento em que a hipótese de aprendizagem de pares estado-ação Q for demonstrada por uma tabela com as informações do valor de cada par, também conhecida por tabela Q, ou Q-Table (MONTEIRO; RIBEIRO, 2004). O Q-Learning é um método que, em vez de aprender a utilidade de um estado, aprende uma representação de ação-valor. Utiliza-se $Q(s,a)$ para representar o valor da execução da ação a no estado s. Os valores Q estão relacionados aos valores de utilidade, onde a utilidade de um estado s é o valor Q máximo entre todas as possíveis ações naquele estado.

O algoritmo SARSA(λ) é uma modificação do algoritmo Q-Learning que utiliza um mecanismo de iteração de política, isto é, ele é um algoritmo *policy-based*, enquanto o Q-Learning é um algoritmo *value-based* (MONTEIRO; RIBEIRO, 2004). A principal diferença entre os algoritmos *policy-based* e *value-based* é como a regra de atualização se relaciona com a política sendo aprendida pelo agente (NISSEN, 2007). Eles convergem para diferentes políticas ótimas. O SARSA(λ) convergirá para uma solução ótima sob a suposição de que continuará seguindo a mesma política usada para gerar a experiência, ou seja, a mesma política do seu processo de aprendizagem. Enquanto o Q-Learning convergirá para uma solução ótima sob o pressuposto de que, depois de gerar experiência e treinamento, passará para a política gananciosa (JIANG et al., 2019). Além disso, o SARSA(λ) é uma adaptação do algoritmo SARSA combinado à traços de elegibilidade. Introduzido por Klopf (1972), o conceito de traço de elegibilidade é memorizar, em um curto tempo, os estados visitados anteriormente. Geralmente, o traço de elegibilidade decai exponencialmente levando em consideração o valor lambda λ e o fator de desconto γ (LAMPERTI; NEPOMUCENO; OTTONI, 2013).

Com o objetivo de superar os problemas que surgiram ao utilizar o Q-Learning e o SARSA (λ) esses dois métodos individualmente, surgiu o algoritmo Actor-Critic (WILLIAMS; BAIRD, 1990), que é basicamente a junção dos dois métodos supracitados. O algoritmo consiste em duas etapas, uma etapa de avaliação de política e uma etapa de melhoria de política. A etapa de avaliação de política diz respeito ao uso eficiente de dados experientes, ou seja, avaliar os dados que foram aprendidos. Enquanto a etapa de melhoria da política serve para melhorar a política em todas as etapas até a convergência (PETERS; SCHAAL, 2008). Em outras palavras, no algoritmo Actor-Critic, a etapa de avaliação pode ser interpretada como um elemento Ator que realiza a aproximação da função política. E a etapa de melhoria pode ser interpretada como um elemento Crítico que realiza a aproximação da função valor (WANG; CHENG; YI, 2007).

BIBLIOTECA BURLAP

A biblioteca de código Java Brown-UMBC Reinforcement Learning and Planning, também conhecida como BURLAP, é usada para desenvolver algoritmos de aprendizagem por reforço (RL) para sistemas mono e multiagentes. A BURLAP estabelece uma estrutura geral para que os usuários possam definir um domínio de problema. A biblioteca fornece uma

ampla variedade de algoritmos RL integrados, domínios pré-fabricados e ferramentas de visualização (NGUYEN; NGUYEN; NAHAVANDI, 2017).

Entre as vantagens de utilizar a BURLAP, é possível mencionar que ela possui algoritmos que variam desde o planejamento clássico até o planejamento estocástico e algoritmos de aprendizado. A BURLAP também conta com ferramentas para definir visualizações de estados, episódios, funções de valor e políticas. Possui ferramentas para configurar experimentos com vários algoritmos de aprendizado e verificar o desempenho usando várias métricas. Possui ainda uma estrutura de shell extensível para controlar experimentos em tempo de execução (MACGLASHAN, 2016).

A BURLAP está disponível online (http://burlap.cs.brown.edu/) , e para conseguir utilizá-la, primeiramente é necessário se familiarizar com as interfaces Java e estruturas de dados que a biblioteca disponibiliza, apresentadas na Figura 1.

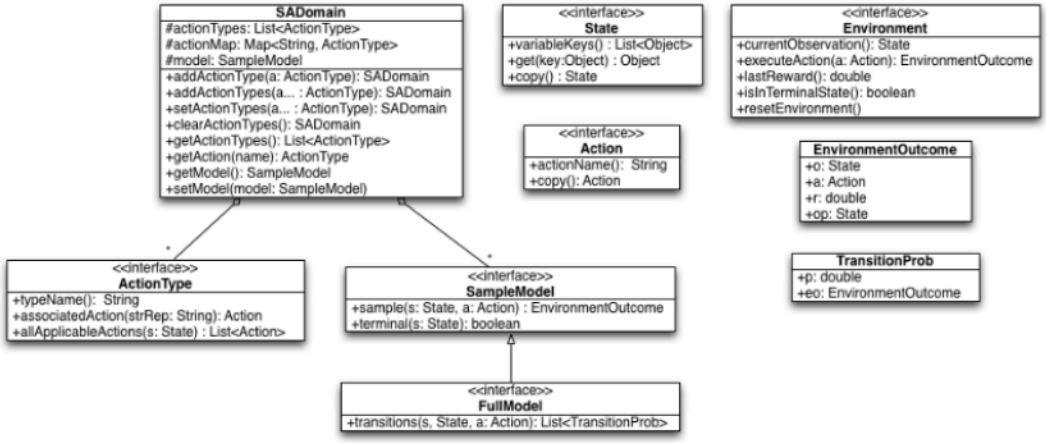

Figura 1: Diagrama UML das interfaces/classes Java da BURLAP

A classe SADomain é uma estrutura de dados responsável por armazenar as informações de como será realizada a recompensa, a definição de estado terminal e a definição de estados. A interface chamada State, por sua vez, serve para definir as variáveis de estado, ou seja, uma instância de um objeto que implementa essa interface especificará um único estado do espaço de estados. Já a interface Action, ao ser implementada, serve para definir uma possível ação que o agente pode selecionar.

A interface ActionType serve para definir um tipo de fábrica Java para gerar as ações. Esta interface também permite definir pré-condições para as ações, se não houver pré-condições é possível considerar ainda a implementação concreta chamada UniversalActionType. A interface SampleModel serve para que sejam implementados métodos que possam criar uma transição, retornar o próximo estado e recompensar o agente. Já a interface Environment serve para fornecer um ambiente com o qual os agentes BURLAP possam interagir. Um outra alternativa é utilizar a classe SimulatedEnvironment fornecida, que usa um SADomain com um SampleModel e simula um ambiente para ele.

A classe EnvironmentOutcome contém um estado/observação anterior, uma ação realizada nesse estado, uma recompensa recebida e um próximo estado/observação para o qual o ambiente fez a transição. Por fim, a classe TransitionProb contém um objeto do tipo double e um EnvironmentOutcome, que especifica a probabilidade de ocorrência da transição especificada por EnvironmentOutcome

CENÁRIO CLIFF WALKING

Com o intuito de avaliar o funcionamento da extensão, foi realizada a implementação de uma simulação na plataforma NetLogo utilizando a extensão. A simulação faz uso de um cenário clássico de aprendizagem por reforço, o Cliff Walking.

Neste cenário o ambiente é um mundo bidimensional dividido em células, como demonstra a Figura 2. Foi implementado um cenário com 3 células de largura e 6 células de comprimento. O agente sempre ocupa uma única célula e pode se locomover por entre as células com as ações de ir para cima, ir para baixo, ir para a esquerda e para a direita. O estado do agente, portanto, é representado pela sua posição na grade de células (SUTTON; BARTO, 2018). O objetivo do agente é partir do ponto S, e chegar ao ponto G, traçando o menor percurso possível ou o mais seguro, sem passar pelo penhasco, representado pelas células em cinza.

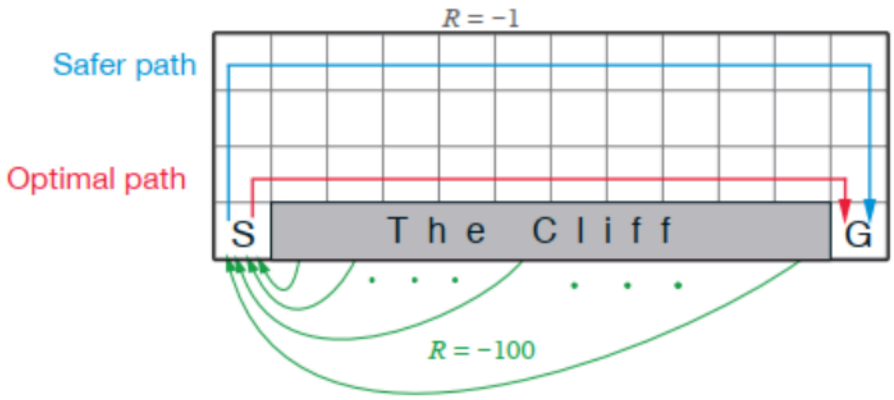

Figure 2: Cenario Cliff Walking

O agente recebe uma recompensa a cada ação tomada. Quando o agente se desloca para qualquer célula que não seja o penhasco, ele recebe uma recompensa no valor -1. Porém, quando ele se desloca para o penhasco, ele recebe uma recompensa no valor -100. O episódio é finalizado toda vez que o agente alcançar o ponto G, ou alcançar o penhasco, e em seguida, é iniciado um novo episódio com o agente no estado inicial S. A linha vermelha mostrada na Figura 2 representa a política ótima que o agente pode aprender e a linha azul representa o caminho seguro que pode ser aprendido pelo agente dependendo do algoritmo a ser utilizado.

Tendo em vista que o método de seleção da próxima ação considera um percentual de aleatoriedade nas escolhas, os resultados de uma única execução não seriam suficientes para alegar uma política ótima. Para tanto, em todas as versões da simulação que foram implementadas, foram feitas 5 execuções e foram coletadas suas respectivas médias. Além disso, vale ressaltar que a quantidade de episódios adotada foi determinada a partir de testes experimentais considerando a convergência para a política ótima. Da mesma forma, os valores dos parâmetros de entrada, que influenciam muito no processo de aprendizado, foram testados até encontrar uma convergência para a política ótima. Vale destacar ainda que a avaliação realizada não possui o intuito de comparar o desempenho dos algoritmos, o objetivo é avaliar se os algoritmos estão funcionando corretamente.

As implementações de todas as versões das simulações, bem como os resultados dessas execuções, suas respectivas médias e desvios-padrão, se encontram disponíveis online

(https://github.com/elobazza/rl-extension-validation). Lá também é possível encontrar uma explicação de como rodar essas simulações.

Considerando os resultados obtidos, foi possível constatar que a refatoração não causou nenhuma inconsistência no aprendizado das políticas, isto é, a refatoração não causou nenhum problema na aprendizagem que já estava funcionando. E também foi possível perceber que os valores dos parâmetros de entrada influenciam nos valores das Q-Tables.

As primeiras combinações de valores dos parâmetros de entrada, no caso do Q-Learning e do SARSA(λ) trouxeram resultados com poucas divergências, porém com valores muito próximos. Mas ao reajustar essas combinações, os resultados foram mais satisfatórios e encontraram a política ótima esperada de cada algoritmo.

Por fim, vale mencionar ainda o alto desempenho do Actor-Critic, que atingiu a convergência de aprendizagem mais rapidamente do que os algoritmos Q-Learning e SARSA (λ)

REFERÊNCIAS:

DAYAN, P.; WATKINS, C. Q-learning. Machine learning, v. 8, n. 3, p. 279–292, 1992.

JIANG, H.; GUI, R.; CHEN, Z.; WU, L.; DANG, J.; ZHOU, J. An improved sarsa (λ) reinforcement learning algorithm for wireless communication systems. IEEE Access, IEEE, v. 7, p. 115418–115427, 2019.

KLOPF, A. H. Brain function and adaptive systems: a heterostatic theory. [S.l.]: Air Force Cambridge Research Laboratories, Air Force Systems Command, United …, 1972.

LAMPERTI, R. D.; NEPOMUCENO, E. G.; OTTONI, A. L. C. Aprendizado por reforco no domínio do futebol de rob^os 2d: Uma analise do traco de elegibilidade com uma comparacao entre os algoritmos q-learning (λ) e sarsa (λ). 2013.

MACGLASHAN, J. Brown-UMBC reinforcement learning and planning (BURLAP). 2016.

MONTEIRO, S. T.; RIBEIRO, C. H. Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel. Sba: Controle & Automação Sociedade Brasileira de Automatica, SciELO Brasil, v. 15, p. 320–338, 2004.

NGUYEN, N. D.; NGUYEN, T.; NAHAVANDI, S. System design perspective for human-level agents using deep reinforcement learning: A survey. IEEE Access, IEEE, v. 5, p. 27091–27102, 2017.

NISSEN, S. Large scale reinforcement learning using q-sarsa (λ) and cascading neural networks. Unpublished masters thesis, Department of Computer Science, University of Copenhagen, København, Denmark, 2007.

PETERS, J.; SCHAAL, S. Natural actor-critic. Neurocomputing, Elsevier, v. 71, n. 7-9, p. 1180–1190, 2008.

SUTTON, R. S.; BARTO, A. G. Reinforcement learning: An introduction. 2. ed. [S.l.]: MIT press, 2018.

WANG, X.-S.; CHENG, Y.-H.; YI, J.-Q. A fuzzy actor–critic reinforcement learning network. Information Sciences, Elsevier, v. 177, n. 18, p. 3764–3781, 2007.

WILLIAMS, R. J.; BAIRD, L. C. A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. In: CITESEER. Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems. [S.l.], 1990. p. 96–101.

Cronograma estabelecido para esse período: (x) cumprido ( ) não cumprido

Dificuldade(s) encontrada(s):

Assinatura bolsista:	**Data:** 16/03/2023

Assinatura orientador:	**Data:** 16/03/2023