Folha de Rosto para Projeto de Iniciação Científica — UFABC

Edital 01/2025

Título:	Aprendizado por Reforço para Problemas de Otimização Combinatória
Supervis@r:	Prof@. Dr@.
Candidat@:	
Período:	01/09/2025 a 31/08/2026
Instituição Sede:	Centro de Matemática, Computação e Cognição Universidade Federal do ABC
Palavras Chave:	Otimização Combinatória, Programação Dinâmica, Aprendizado por Reforço, Redes Neurais Gráficas
Área de Conhecimento:	Ciência da Computação, Matemática

Declaração de Interesse por Bolsa

Declaro que a/o candidata/o, nos termos do edital 01/2025, deseja participar do programa de iniciação científica como **bolsista**.

APRENDIZADO POR REFORÇO PARA OTIMIZAÇÃO COMBINATÓRIA

Resumo. Muitos problemas NP-difíceis em otimização combinatória são, contra-intuitivamente, rotineiramente resolvidos na prática via métodos exatos, aproximados, ou heurísticos. Em comum a essas abordagens, o fato de de terem sido desenvolvidas pata produzirem soluções para instâncias gerais e individuais dos problemas. Em muitas situações práticas de interesse, no entanto, é preciso resolver instâncias de problemas com características ou padrões específicos e o desenvolvimento de algoritmos mais eficientes para tais casos que explorem padrões comuns nas instâncias fornecidas é desejável. Este projeto visa estudar e desenvolver algoritmos desta natureza via aplicação de técnicas de Aprendizado por Reforço ao problema do Caixeiro Viajante e algumas variações. Este tema – e as técnicas e algoritmos envolvidos – são raramente vistas durante a graduação e, quando o são, costumam representar um desafio para a maioria dos alunos. A produção de material abordando esses assuntos pode ser de grande utilidade para estudantes de Ciência da Computação e cursos relacionados.

1. Introdução e Justificativa

A **Otimização Combinatória** (CO = *Combinatorial Optimization*) se desenvolveu como um campo interdisciplinar que abrange ciência da computação, matemática discreta, otimização e pesquisa operacional. Apresenta resultados profundos e engenhosos do ponto de vista teórico que encontram vasta usabilidade em aplicações críticas do *"mundo real"* como emparelhamentos, escalonamentos e roteamento de veículos, entre tantas outras (veja [6, 10] para uma visão geral).

Em poucas palavras – e de maneira informal – a CO lida com problemas que envolvem a otimização (minimização ou maximização) de uma função objetivo expressando, por exemplo, peso ou custo ao selecionar um subconjunto de um conjunto finito, com este último codificando restrições no espaço da solução. A explosão combinatória do número de subconjuntos (= soluções candidatas) é inerente a tais problemas e, a menos que alguma variação

discreta do conceito de convexidade esteja presente, costumam ser notoriamente difíceis do ponto de vista de complexidade computacional (geralmente NP-difíceis, mas podendo atingir níveis mais altos na hierarquia polinomial de tempo; veja [1, 8, 10]). Apesar disso e contra-intuitivamente, muitos desses problemas difíceis são rotineiramente resolvidos na prática via métodos exatos [5, 10, 13], aproximados [19, 20], ou heurísticos [3, 15]. Em comum a essas três categorias bastante distintas, está o fato de terem sido desenvolvidas com o objetivo de produzirem soluções para instâncias gerais e individuais dos problemas.

Em muitas situações práticas de interesse, no entanto, é preciso resolver instâncias de problemas com características ou padrões específicos. Por exemplo, uma transportadora pode resolver instâncias de roteamento de veículos para a mesma cidade diariamente, com apenas pequenas diferenças entre as instâncias nos tempos de viagem devido às variadas condições de tráfego. Assim, algoritmos orientados/dependentes a/de dados ou abordagens de aprendizado de máquina, que podem explorar esses padrões, surgiram recentemente como alternativas ([2, 9]) para o desenvolvimento de algoritmos mais rápidos para casos práticos ao explorar padrões comuns nas (coleções de) instâncias fornecidas.

Uma abordagem que tem potencial promissor e o tem demonstrado em alguns problemas de CO [11, 21] atende pelo nome de **Aprendizado por Reforço** (RL = *Reinforcement Learning*). Intuitivamente, o RL é uma estrutura para um *agente* aprender a tomar decisões sequenciais ótimas em um *ambiente*, a fim de maximizar um *sinal* de recompensa numérico. Formalmente, o problema central do RL pode ser definido ([14, 18]) via um **Processo de Decisão de Markov** (MDP = *Markov Decision Process*), uma 5-tupla $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, em que:

- $\mathcal{S}$ é um conjunto finito de estados.
- $\mathcal{A}$ é um conjunto finito de ações.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ é a função de probabilidade de transição de estado, tal que $\mathcal{P}(s'|s, a)$ representa a probabilidade de transição para o estado s' a partir do estado s após tomar a ação a. Observe que $\mathcal{P}$ satisfaz a propriedade de Markov, em que o

próximo estado depende apenas do estado e da ação atuais, e não da sequência de estados e ações precedentes.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ é a função de recompensa, tal que $\mathcal{R}(s, a, s')$ representa a recompensa imediata recebida após a transição do estado s para o estado s' ao tomar a ação a.
- $\gamma \in [0, 1]$ é o fator de desconto, que determina o valor presente de recompensas futuras. Um γ maior significa que recompensas futuras são consideradas mais valiosas.

O objetivo do agente é aprender uma **política** $\pi : \mathcal{S} \to \mathcal{A}$ (determinística) ou $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ (estocástica, onde $\pi(a|s)$ é a probabilidade de tomar a ação a no estado s) que maximize o retorno cumulativo descontado esperado (recompensa total) a partir de qualquer estado dado. O retorno G_t no passo de tempo t é definido como:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

em que R_{t+k+1} é a recompensa recebida no passo de tempo $t + k + 1$.

A **política ótima** π^* é aquela que alcança o retorno esperado máximo para todos os estados:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ E_\pi[G_t|S_t = s, A_t = a].$$

Isso envolve encontrar as **funções de valor** ótimas:

- **Função de valor de estado** $V^\pi(s)$: O retorno esperado ao iniciar no estado s e seguir a política π.

$$V^\pi(s) = E_\pi[G_t|S_t = s]$$

- **Função de valor de ação** $Q^\pi(s, a)$: O retorno esperado ao iniciar no estado s, tomar a ação a, e a partir daí seguir a política π.

$$Q^\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

As funções de valor ótimas, $V^*(s)$ e $Q^*(s,a)$, satisfazem as **equações de otimalidade de Bellman**:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^*(s')],$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma \max_{a'} Q^*(s',a')].$$

O problema do aprendizado por reforço, portanto, é encontrar a política ótima π^* ou, equivalentemente, a função de valor de ação ótima $Q^*(s,a)$, tipicamente através da interação com o ambiente, sem conhecimento explícito de $\mathcal{P}$ e $\mathcal{R}$. Especializações/variantes do problema do aprendizado por reforço (Model-Based vs. Model-Free, Value-Based vs. Policy-Based vs. Actor-Critic, On-Policy vs. Off-Policy) são categorizadas de acordo como o agente aprende e representa o conhecimento adquirido (detalhes em [14, 18]).

Este projeto de Iniciação Científica irá explorar a compreensão, modelagem (como MDP), construção e implementação de algoritmos de RL aplicados a problemas de CO. Particularmente, o projeto focará primariamente na aplicação de métodos de RL ao clássico Problema do Caixeiro Viajante (TSP = *Travelling Salesperson Problem*) – que busca a rota mais curta ou de menor custo que visita um conjunto de cidades exatamente uma vez e retorna à cidade de origem – e algumas de suas variações como as com janelas de tempo (= *time-window*) e coleta de prêmios (= *prize-collecting*).

O intuito subjacente é que os princípios e métodos estudados/desenvolvidos neste projeto possam ser estendidos a uma gama mais ampla de problemas de CO, consistindo em uma abordagem híbrida, que combina o poder de aprendizagem do RL com elementos de algoritmos clássicos de otimização e visa a superação das limitações de ambos. Assim, outras variações e outros problemas serão considerados mediante a evolução da iniciação / caso o tempo permita.

Por fim, vale ressaltar que as técnicas, os métodos e os algoritmos brevemente sugeridos acima são pouco mencionados nas disciplinas do curso e, quando o são, costumam representar um desafio para a maioria dos alunos. O motivo costuma ser a necessidade de maiores graus

de raciocínio abstrato e maturidade matemática. Assim, a produção de material abordando esses assuntos pode ser de grande utilidade para os alunos do Bacharelado de Ciência da Computação e cursos relacionados que queiram adentrar à área.

2. Objetivos

2.1. **Objetivo geral.** O principal objetivo deste projeto é introduzir o aluno a resultados teóricos relevantes em teoria dos grafos, otimização combinatória, complexidade computacional e aprendizado por reforço, e colocá-lo em uma trajetória de proficiência em programação com o intuito de formar uma base sólida para pesquisas futuras. Esperamos que este objetivo seja alcançado através do domínio de alguns dos algoritmos de aproximação para o TSP (e variações) e dos métodos de aprendizados por reforço que serão aplicados a tal problema.

2.2. **Objetivos específicos.** A seguir listamos os objetivos específicos a serem alcançados:

- Desenvolver e aprofundar conhecimentos na linguagem Python e em boas técnicas de programação.
- Obter uma boa compreensão de conceitos elementares em: teoria dos grafos, otimização combinatória, complexidade computacional e aprendizado por reforço.
- Formalizar matematicamente o TSP como um Processo de Decisão de Markov, definindo estados, ações, transições e funções de recompensa adequadas para a construção de soluções.
- Estudar e selecionar algoritmos de Aprendizado por Reforço apropriados para lidar com a natureza sequencial e combinatória do TSP, com ênfase em métodos baseados em valor ou em política que possam ser adaptados para espaços de ação e estado (implícitos) dinâmicos.
- Implementar uma arquitetura de RL (e.g., baseada em redes neurais para aproximação de funções de valor ou políticas) capaz de aprender heurísticas de construção de rotas para instâncias do TSP.

- Avaliar rigorosamente a qualidade das soluções encontradas (custo total da rota) e a eficiência computacional (tempo de execução, tempo de treinamento) da abordagem proposta em diversas instâncias de TSP de diferentes tamanhos e características.
- Realizar uma análise da convergência do processo de aprendizado do RL e da robustez das políticas aprendidas frente a novas instâncias do problema.

3. Metodologia

A metodologia deste projeto consiste de três tipos de atividades que serão realizadas durante todo o projeto:

- Estudo individual dos conceitos e algoritmos envolvidos.

  Este será um estudo individual, em grande parte bibliográfico, de conceitos e algoritmos.
- Atividades de programação através da resolução de problemas.

  Cada algoritmo estudado deverá ser implementado e testado na resolução de problemas.
- Reuniões com o orientador.

  Serão realizadas reuniões frequentes com o orientador para a seleção de problemas, esclarecimentos sobre os conceitos e algoritmos e direcionar as atividades listadas abaixo.

@ alun@ integrará o grupo de **Teoria da computação, Otimização, Combinatória e Algoritmos** (TOCA) da UFABC e utilizará o laboratório do grupo (no bloco L) por ao menos 12 das 20 horas semanais que dedicará ao projeto – as demais 8 poderão ser realizadas em outro lugar de sua escolha.

4. Cronograma de Atividades

A seguir, descrevemos as atividades a serem realizadas durante o projeto, ilustrando em maiores detalhes os objetivos específicos mencionados anteriormente.

- **Atividade 1: Revisão de Python** Um dos focos do projeto será a implementação de algoritmos de aprendizado por reforço para o TSP e variações. O aluno já possui um conhecimento inicial na linguagem Python, que é simples, flexível e apropriada para a tarefa. Para que os objetivos de programação sejam atingidos, no entanto, um aprofundamento nas características da linguagem, bibliotecas existentes, e em boas técnicas de programação faz-se necessário. Será utilizado o livro [17].
- **Atividade 2: Introdução à Teoria e Algoritmos em Grafos** Estudo de conceitos, problemas e algoritmos elementares em teoria dos grafos, incluindo buscas, caminhos, árvores (geradoras mínimas), fluxos e emparelhamentos. Serão utilizados os livros [7, 16].
- **Atividade 3: Problema do Caixeiro Viajante e Variações.** Estudo da definição do problema, formulações (inteira e via fluxos), resultado de NP-dificuldade, relaxamentos (combinatório e Lagrangeano), soluções exatas por programação dinâmica e branch-and-bound, 2 e 3/2 aproximações, algumas heurísticas como 2-opt e Lin-Kerninghan-Helsgaun. Serão utilizados os livros [6,10] e artigos a serem definidos ao longo da IC para as variações.
- **Atividade 4: Teoria e Algoritmos para Aprendizado por Reforço.** Estudo dos fundamentos de aprendizado por reforço com ênfase em processos de decisão de Markov (explícitos, parcialmente observados e contextuais caso o tempo permita), equações de Bellman, funções de valor e política (exatas e aproximadas). Foco em algoritmos que lidam com espaços grandes/volumosos/massivos de estados, como Deep Q-Networks (DQN), Policy Gradients (REINFORCE) e Actor-Critic (A2C/A3C/PPO) e variações para aprendizado *offline* livre e baseados em modelos (Conservative Q-learning (CQL), Policy Constrained Gradients (AWR, AWAC, ABM), MOREL). Serão utilizados os livros [12, 18].
- **Atividade 5: Revisão de Literatura.** Nesta etapa, o aluno irá pesquisar trabalhos que aplicam RL para resolver problemas de otimização combinatória, em particular o TSP, buscando identificar as abordagens mais promissoras, os desafios comuns e

as possibilidades ainda em aberto. Enfoque adicional em modelos de rede neural para representação de grafos (Graph Neural Networks - GNNs) se o tempo permitir. Pontos de partida serão os *surveys*: [4, 11, 21].

- **Atividade 6: Formulação do Problema.** Aqui, ele irá (guiado pelo orientador) definir matematicamente o TSP como um MDP, definindo precisamente os estados (parcialmente construídos ou visitados), ações (próxima cidade a visitar) e recompensas (decremento no custo total ou bônus por completude). Fontes de apoio são: [4, 11, 21].
- **Atividade 7: Implementações.** Seleção e implementação de alguns algoritmos de RL apropriados (como um algoritmo de gradiente de política como REINFORCE ou PPO, e um valor-baseado como DQN, com redes neurais para generalização). A escolha dependerá da facilidade de adaptação ao TSP e da complexidade das implementações iniciais. Desenvolvimento de estratégias para representar estados e ações de forma eficiente para a rede neural subjacente, considerando a natureza gráfica do problema. Realização de testes e treinamentos iniciais do algoritmo de RL em instâncias pequenas do TSP para verificar a validade da abordagem e a capacidade de aprendizado do agente.
- **Atividade 8: Experimentação, Análise de Desempenho e Avaliação Empírica.** Utilização de bibliotecas (TSPLIB) geradores (aleatórios) de instâncias para o TSP para criar um conjunto de testes diversificado em tamanho e características (e.g., cidades em linha, cidades agrupadas, etc.). Realização de experimentos comparativos entre os algoritmos implementados. Análise da qualidade das soluções obtidas (custo total da rota, *gap* em relação ao ótimo/melhor conhecido), tempo de execução para encontrá-las, tempo de treinamento do modelo de RL e estabilidade da política aprendida e sua generalização para instâncias não vistas.

  Contando ainda com o fato de que alguns dos algoritmos que ele irá implementar estão disponíveis em pacotes especializados em Python (como ScikitLearn, pyTorch,

TensorFlow) e poderão servir de base de comparação (*benchmarks*) para as implementações realizadas.

- **Atividade 9: Elaboração de relatórios.** Esta atividade se refere a elaboração do relatório parcial e do relatório final.

4.1. **Cronograma.** Este projeto tem 12 meses de duração (01/09/2025 a 31/08/2026).

Ativ.	Título	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago
1	Revisão de Python	*	*										
2	Introdução a grafos		*	*									
3	O caixeixo viajante (TSP)		*	*	*								
4	Aprendizado por reforço			*	*	*	*	*	*	*	*		
5	Revisão da literatura				*	*							
6	Formulação do problema					*	*						
7	Implementações						*	*	*	*	*		
8	Experimentação / Análise								*	*	*	*	
9	Elaboração de relatórios						*						*

5. Viabilidade

Consideramos que, em linhas gerais, este projeto é bastante ambicioso para um aluno no início do segundo ano de graduação. No entanto, o aluno já possui uma familiaridade com a linguagem Python, obteve bons resultados nos cursos de programação e de matemática que já cursou na UFABC, e está bastante empolgado com o tema deste projeto (algo que é simplesmente, fundamental). Os resultados e algoritmos a serem estudados e implementados situam-se em nível introdutório de suas respectivas áreas e foram planejados em números grandes o suficiente para que possam ser escolhidos de acordo com o avanço do aluno. Logo, expressamos um alto grau de confiança na viabilidade da execução do projeto.

Referências

[1] Arora, S., and Barak, B. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009. → 2

[2] Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research 290*, 2 (2021), 405–421. → 2

[3] Boussaïd, I., Lepagnot, J., and Siarry, P. A survey on optimization metaheuristics. *Information Sciences 237* (2013), 82–117. Prediction, Control and Diagnosis using Advanced Neural Computations. → 2

[4] Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research 24*, 130 (2023), 1–61. → 8

[5] Conforti, M., Cornuéjols, G., and Zambelli, G. *Integer Programming*. Graduate Texts in Mathematics. Springer International Publishing, 2014. → 2

[6] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. *Combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1998. A Wiley-Interscience Publication. → 1 and 7

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2009. → 7

[8] Garey, M. R., and Johnson, D. S. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Co., San Francisco, Calif., 1979. A Series of Books in the Mathematical Sciences. → 2

[9] Gasse, M., Bowly, S., Cappart, Q., Charfreitag, J., Charlin, L., Chételat, D., Chmiela, A., Dumouchelle, J., Gleixner, A., Kazachkov, A. M., Khalil, E., Lichocki, P., Lodi, A., Lubin, M., Maddison, C. J., Christopher, M., Papageorgiou, D. J., Parjadis, A., Pokutta, S., Prouvost, A., Scavuzzo, L., Zarpellon, G., Yang, L., Lai, S., Wang, A., Luo, X., Zhou, X., Huang, H., Shao, S., Zhu, Y., Zhang, D., Quan, T., Cao, Z., Xu, Y., Huang, Z., Zhou, S., Binbin, C., Minggui, H., Hao, H., Zhiyu, Z., Zhiwu, A., and Kun, M. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track* (06–14 Dec 2022), D. Kiela, M. Ciccone, and B. Caputo, Eds., vol. 176 of *Proceedings of Machine Learning Research*, PMLR, pp. 220–231. → 2

[10] Korte, B., and Vygen, J. *Combinatorial optimization: Theory and algorithms*, sixth ed., vol. 21 of *Algorithms and Combinatorics*. Springer, Heidelberg, 2018. → 1, 2, and 7

[11] Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research 134* (2021), 105400. → 2 and 8

[12] Murphy, K. Reinforcement learning: An overview, 2025. → 7

[13] Nemhauser, G., and Wolsey, L. Maximizing submodular set functions: formulations and analysis of algorithms. *In Studies of Graphs and Discrete Programming* (1981), 279–301. → 2

[14] Powell, W. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. Wiley, 2022. → 2 and 4

[15] Resende, M., and Ribeiro, C. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer New York, 2016. → 2

[16] Sedgewick, R. *Algorithms*. Addison-Wesley Professional, 2011. → 7

[17] Sedgewick, R., Wayne, K., and Dondero, R. *Introduction to Programming in Python: An Interdisciplinary Approach*, 1 ed. Addison-Wesley Professional, June 2015. → 7

[18] Sutton, R., and Barto, A. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. → 2, 4, and 7

[19] Vazirani, V. V. *Approximation algorithms*. Springer, 2001. → 2

[20] Williamson, D. P., and Shmoys, D. B. *The design of approximation algorithms*. Cambridge University Press, Cambridge, 2011. → 2

[21] Yang, Y., and Whinston, A. B. A survey on reinforcement learning for combinatorial optimization. *CoRR abs/2008.12248* (2020). → 2 and 8