Projeto 2
Problema de Alocação de Estudantes em Projetos (SPA)

Élvis Miranda, 24/1038700
Gustavo Alves, 24/1020779
Pedro Marcinoni, 24/1002396

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0199 - Teoria e Aplicação de Grafos
11 de dezembro de 2025

elviscorreamirandajr@gmail.com, gusfring.a@gmail.com, pedroextrer@gmail.com

Abstract. *This paper presents the implementation of a solution for the Student-Project Allocation problem (SPA), applied to a scenario involving 200 students and 50 research projects, modeled through Graph Theory. The main objective was to develop a maximum stable matching algorithm that adheres to project capacity constraints (minimum and maximum quotas) and student qualification requirements (grades from 3 to 5), ensuring an impersonal and competitive selection process. Furthermore, two variations of the algorithm (student-proposing vs. project-proposing) were implemented and compared regarding their impact on stability and satisfaction.*

Resumo. *Este trabalho apresenta a implementação de uma solução para o Problema de Alocação de Estudantes em Projetos (SPA – Student-Project Allocation problem), aplicado a um cenário com 200 alunos e 50 projetos de pesquisa, modelado através da Teoria dos Grafos. O objetivo principal foi desenvolver um algoritmo de emparelhamento estável máximo que respeitasse as restrições de capacidade (mínima e máxima) dos projetos e os requisitos de qualificação dos alunos (notas de 3 a 5), garantindo uma seleção impessoal e competitiva. Além disso, duas variações do algoritmo (proposta pelo estudante vs. proposta pelo projeto) foram implementadas e comparadas em relação ao seu impacto na estabilidade e satisfação.*

1. Introdução

1.1. Objetivos

O objetivo deste projeto é projetar e implementar uma solução algorítmica para o Problema de Alocação de Estudantes em Projetos (SPA), modelando o cenário como um grafo bipartido. O trabalho visa garantir a alocação estável máxima entre 200 alunos e 50 projetos, respeitando restrições de capacidade e requisitos de qualificação. O programa deve ser capaz de:

- Representar o cenário de alocação através de um grafo bipartido, onde um conjunto de vértices representa os alunos e o outro os projetos ofertados;

- Implementar variações do algoritmo de Gale-Shapley, conforme proposto por Abraham, Irving Manlove (2007), para encontrar um emparelhamento estável máximo;
- Simular e visualizar a evolução do algoritmo, exibindo iterações do processo de propostas, emparelhamentos temporários e rejeições;
- Calcular e exibir métricas de satisfação, especificamente comparando a ordem de preferência dos alunos com o rank de classificação atribuído pelos projetos;
- Gerar uma matriz final de emparelhamento que evidencie o ”ganho”ou ”perda”de cada parte envolvida na alocação.

1.2. Métodos

Os métodos utilizados neste projeto envolvem a modelagem de grafos bipartidos e a aplicação de algoritmos de correspondência estável (stable matching) para a resolução de problemas de alocação com restrições. Para isso, o programa desenvolvido em **Python** realiza as seguintes etapas:

- Processamento de Dados: Leitura e estruturação do arquivo de entrada, contendo as capacidades dos projetos (vagas mínimas e máximas), requisitos de notas e as listas de preferências dos alunos;
- Aplicação Algorítmica: Execução da lógica de emparelhamento baseada em propostas sucessivas, onde alunos aplicam para projetos e estes aceitam provisoriamente ou rejeitam com base em capacidade e qualificação (notas 3, 4 ou 5);
- Visualização Dinâmica: Renderização gráfica do grafo bipartido para monitorar a evolução das iterações e a comparação visual entre os resultados finais das duas variações propostas, utilizando cores distintas para diferenciar o estado das arestas (proposta ativa, emparelhamento temporário e rejeição);
- Análise de Métricas: Cálculo de índices de preferência e geração de uma matriz de resultados que cruza o rank do aluno na lista do projeto com o rank do projeto na lista do aluno, permitindo a validação da estabilidade e qualidade da solução encontrada.

2. Procedimentos e Resultados

2.1. Modelagem e Estrutura de Dados

O problema foi modelado utilizando Programação Orientada a Objetos em Python. Foram instanciadas duas classes principais: *Project* e *Student*. A leitura dos dados (`entradaProj2.25TAG.txt`) gerou 50 objetos de projeto e 200 objetos de aluno.

Para garantir a integridade da simulação, as listas de preferência dos projetos foram pré-processadas, ordenando os candidatos aptos primeiramente por nota (decrescente) e, secundariamente, por ordem de chegada (ID).

2.2. Algoritmos de Emparelhamento Implementados

Foram desenvolvidas e comparadas duas variações do algoritmo SPA (*Student-Project Allocation*), ambas visando a estabilidade, mas com lógicas de proposição opostas:

2.2.1. Variação 1: SPA Orientado ao Estudante (Student-Proposing)

Esta variação prioriza a escolha ativa dos alunos. O algoritmo itera enquanto houver alunos livres que ainda não propuseram para todas as opções de sua lista.

- **Mecanismo de Proposta:** O aluno propõe para o seu projeto preferido disponível.
- **Critério de Aceitação e Substituição (”Mogging”):** Se o projeto tem vagas, o aluno entra. Se o projeto está cheio, ocorre uma competição com o pior candidato atual. O novo aluno substitui o atual se:

    1. Sua nota for estritamente maior; OU
    2. As notas forem iguais, mas o novo aluno tiver listado aquele projeto em uma posição prioritária mais alta (menor índice de preferência) do que o aluno atual.

- **Objetivo:** Tende a produzir o emparelhamento estável ”ótimo para o estudante”.

2.2.2. Variação 2: SPA Orientado ao Projeto (Project-Proposing)

Nesta variação, os projetos realizam as propostas ativas para os alunos em suas listas de preferência.

- **Mecanismo de Proposta:** Projetos com vagas ociosas propõem para o próximo aluno qualificado de sua lista.
- **Decisão do Aluno:**

    1. Se o aluno estiver livre, aceita provisoriamente.
    2. Se o aluno já estiver alocado, ele compara a nova proposta com a atual. Se o novo projeto for ”melhor”(estiver acima na sua lista pessoal de preferências), ele troca e libera a vaga no projeto anterior.

- **Objetivo:** Tende a produzir o emparelhamento estável ”pessimal para o estudante”(ou ótimo para o projeto).

2.3. Resultados Visuais e Comparativos

A visualização gráfica permite observar a evolução das propostas e a saturação das vagas.

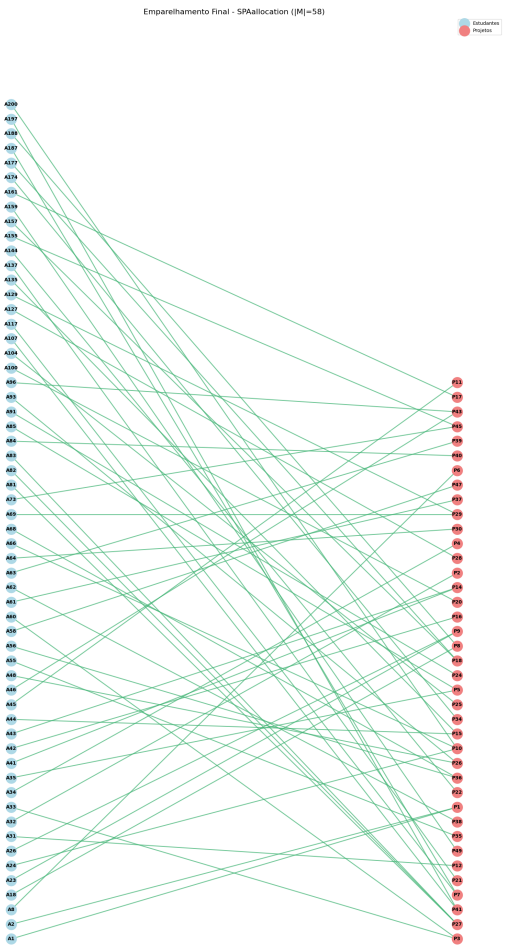

Figura 1. Grafo Final: Variação Orientada ao Estudante.

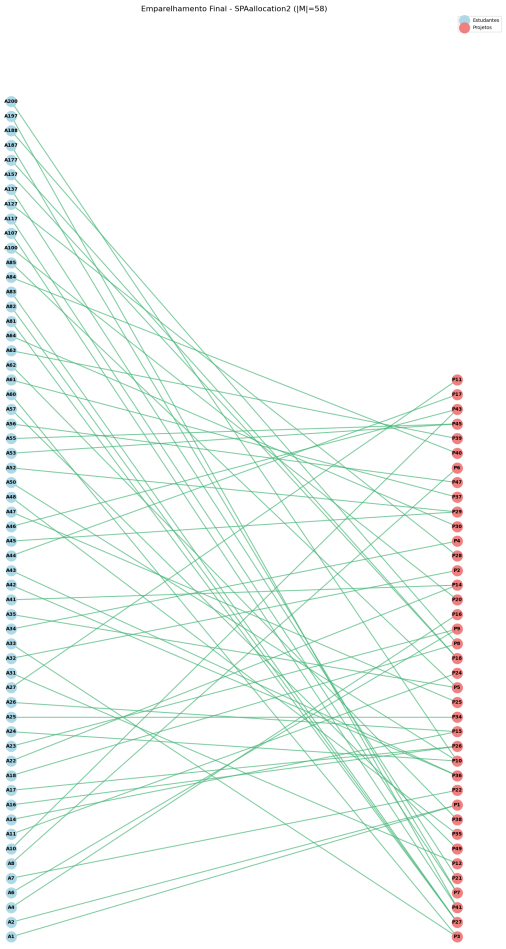

Figura 2. Grafo Final: Variação Orientada ao Projeto.

3. Conclusões

A implementação das duas variações do algoritmo SPA evidenciou o impacto de quem detém a iniciativa da proposta na satisfação final dos agentes.

Na **Variação 1 (Orientada ao Estudante)**, observou-se que os alunos tendem a conseguir vagas mais próximas ao topo de suas listas de preferências. O critério de desempate implementado (priorizando quem ”quer mais”o projeto em caso de empate de notas) serviu como um fator decisivo para maximizar a utilidade percebida pelos discentes.

Na **Variação 2 (Orientada ao Projeto)**, a dinâmica favoreceu os projetos, que conseguiram preencher suas vagas seguindo rigorosamente sua ordem de classificação de notas. Embora estável (sem pares bloqueadores), essa solução tende a alocar os alunos em opções menos prioritárias de suas listas pessoais quando comparada à primeira variação.

Essa dualidade confirma a teoria de Gale-Shapley: embora a estabilidade seja garantida em ambos os casos, o lado que propõe (*proposing side*) geralmente obtém o melhor resultado possível dentro do conjunto de soluções estáveis.

Referências

[Abraham et al. 2007] Abraham, D. J., Irving, R. W., and Manlove, D. F. (2007). Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5(1):73–90.

[Bondy and Murty 1976] Bondy, J. A. and Murty, U. S. R. (1976). *Graph Theory with Applications*. The Macmillan Press Ltd, London, 1st edition.

[Foundation 1991] Foundation, P. S. (1991). Python programming language. https://www.python.org/.

[Hagberg et al. 2008] Hagberg, A. A., Swart, P. J., and Sculley, D. (2008). Networkx. https://networkx.org/.

[Hunter 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. https://matplotlib.org/.

[Team 2020] Team, T. P. D. (2020). Pandas. https://pandas.pydata.org/.