Uma imagem com texto, Gráficos, Tipo de letra, design gráfico

Descrição gerada automaticamente

**Inteligência Computacional e Otimização**

**Estudo do problema de *Timetabling***

**Apresentação de potencial solução de mitigação**

António Nanita nº 122240

Gonçalo Botelho nº 98893

Miguel Valadares nº 98245

Rodrigo Alves nº 121700

Índice

[Introdução 3](#_Toc170513618)

[Problema do *Timetabling* 3](#_Toc170513619)

[Possível Solução 4](#_Toc170513620)

[Input 4](#_Toc170513621)

[Visualização & Análise dos Horários 5](#_Toc170513622)

[Output 5](#_Toc170513623)

[Contexto 5](#_Toc170513624)

[Utilização de Algoritmos de Otimização 6](#_Toc170513625)

[Frameworks Utilizadas 6](#_Toc170513626)

[JMetalPy 7](#_Toc170513627)

[Flask 7](#_Toc170513628)

[Implementação 7](#_Toc170513629)

[Avaliação da Solução 8](#_Toc170513630)

[Criação da Solução Inicial 8](#_Toc170513631)

[Justificações Técnicas 9](#_Toc170513632)

[Dificuldades no Projeto 9](#_Toc170513633)

[Escolha dos Algoritmos 10](#_Toc170513634)

[Definição dos Objetivos e Criação da Solução Inicial 11](#_Toc170513635)

[Análise de performance dos diferentes tipos de otimização 11](#_Toc170513636)

[Sobrelotações 11](#_Toc170513637)

[Requisitos Não Cumpridos 12](#_Toc170513638)

[Alunos a Mais (Sobrelotações) 12](#_Toc170513639)

[Conclusão 13](#_Toc170513640)

[Bibliografia 13](#_Toc170513641)

## Introdução

O problema de *timetabling* é um desafio comum em diversas áreas, especialmente em instituições de ensino, empresas e organizações que têm a necessidade de organizar eventos ou atividades em períodos específicos. Este problema envolve a atribuição de recursos limitados (como salas, professores ou equipamentos) a um conjunto de atividades (como aulas, exames ou reuniões) de modo a respeitar um conjunto de restrições.

No contexto educacional, por exemplo, o *timetabling* consiste na criação de horários para aulas de modo que não haja sobreposições de professores, estudantes ou salas, e que se respeitem as preferências e restrições de todos os envolvidos. As restrições podem ser classificadas em duras (hard constraints) e suaves (soft constraints). As restrições duras são aquelas que não podem ser violadas, como a disponibilidade de uma sala ou a necessidade de um professor estar presente numa determinada aula. Já as restrições suaves são preferências que se procura atender, como a preferência de um professor por não lecionar em certos horários.

A complexidade do problema de *timetabling* decorre da necessidade de equilibrar múltiplas restrições e preferências, o que frequentemente resulta num espaço de soluções vasto e complicado. Além disso, este problema é um exemplo clássico de um problema de otimização combinatória, onde a procura pela melhor solução requer métodos sofisticados e, muitas vezes, computacionalmente intensivos, como algoritmos heurísticos, metaheurísticas ou técnicas de programação matemática.

A resolução eficaz do problema de *timetabling* é crucial para o bom funcionamento das instituições, garantindo que os recursos são utilizados de maneira eficiente e que os horários resultantes são convenientes e equitativos para todos os envolvidos.

## Problema do *Timetabling*

O problema do *timetabling* é intrinsecamente combinatório, o que significa que envolve a combinação de elementos de diferentes maneiras para encontrar uma solução ótima. À medida que o número de eventos e restrições aumenta, o espaço de busca cresce exponencialmente.

As restrições no problema de *timetabling* podem ser categorizadas em restrições rígidas (*hard constraints*) e restrições flexíveis (*soft constraints*). Os objetivos geralmente incluem otimizar a utilização dos recursos disponíveis e minimizar a violação das restrições flexíveis.

Neste trabalho, apresentamos um problema de *timetabling* universitário, que é mais complexo do que em um âmbito escolar. Ele envolve a alocação de aulas em diferentes cursos e departamentos, considerando a disponibilidade de salas específicas, os horários preferenciais dos professores e a necessidade de evitar conflitos entre disciplinas comuns a diferentes cursos.

## Possível Solução

Uma possível solução consiste em criar e otimizar os horários das aulas da universidade utilizando dois ficheiros de entrada, com o intuito de melhorar a qualidade dos horários segundo critérios específicos.

### Input

#### 1. CarateristcasDasSalas

Este ficheiro contém a descrição das salas disponíveis na universidade e suas respetivas características. Cada linha deste ficheiro, representa uma sala e inclui as seguintes colunas:

* **Edifício**: Nome ou identificação do edifício onde a sala está localizada.
* **Nome sala**: Nome ou número da sala.
* **Capacidade Normal**: Capacidade padrão da sala para aulas normais.
* **Capacidade Exame**: Capacidade da sala para exames, que é geralmente menor devido ao espaçamento extra entre os alunos.
* **Nº características**: Número total de características específicas que a sala possui.
* **Características específicas**: Diversas colunas indicando se a sala possui certas características, como "Anfiteatro aulas", "Apoio técnico eventos", "Laboratório de Informática", etc.

#### 2. HorarioDeExemplo

Este ficheiro contém os horários das aulas já programadas e as salas que já foram atribuídas. Cada linha representa um *slot* de horário e inclui as seguintes colunas:

* **Curso**: Nome do curso.
* **Unidade de execução**: Nome da disciplina ou unidade curricular.
* **Turno**: Turno em que a aula ocorre (por exemplo, manhã, tarde, noite).
* **Turma**: Turma associada ao turno.
* **Inscritos no turno**: Número de alunos inscritos no turno.
* **Dia da Semana**: Dia da semana em que a aula ocorre.
* **Início**: Hora de início da aula.
* **Fim**: Hora de término da aula.
* **Dia**: Data específica da aula (se aplicável).
* **Características da sala pedida para a aula**: Características necessárias para a sala onde a aula será realizada.
* **Sala da aula**: Sala que foi atribuída para a aula.
* **Lotação**: Capacidade da sala atribuída.
* **Características reais da sala**: Características da sala atribuída.

### Visualização & Análise dos Horários

O programa, após carregar os dois ficheiros introduzidos, analisa o estado dos horários com base em critérios de avaliação específicos. Os critérios são os seguintes:

* **Sobrelotações**: Evitar a alocação de aulas em salas cuja capacidade é insuficiente para o número de alunos inscritos.
* **Sobreposições**: Garantir que não haja conflitos de horários entre aulas programadas para a mesma sala.
* **Requisitos não cumpridos**: Assegurar que as salas atribuídas possuam todas as características necessárias para as aulas.
* **Aulas sem sala**: Garantir que todas as aulas sejam alocadas a uma sala adequada.

A partir da visualização dos resultados para cada um dos critérios de avaliação, com o auxílio das diferentes tabelas e gráficos disponibilizados pelo programa desenvolvido, é possível compreender detalhadamente a estrutura do horário submetido. Esta análise permite ao utilizador identificar de forma clara e objetiva os pontos críticos que necessitam de ajustes, promovendo uma gestão mais eficiente dos recursos e espaços disponíveis.

### Output

O programa gera como output um novo horário otimizado, proporcionando ao utilizador uma solução que resolve os seguintes critérios de avaliação, como a sobrelotação ou requisitos não cumpridos.

## Contexto

Existem duas abordagens principais para abordar o problema:

* ***Room Usage Problem***: Foca-se na utilização eficiente das salas disponíveis, maximizando o uso das salas e minimizando o tempo em que permanecem vazias.
* ***Timetable Creation Problem***: Foca-se na criação de horários otimizados que atendam aos requisitos dos alunos e professores, além das características específicas das salas.

Neste trabalho, tratamos o problema como um *Room Usage Problem*, onde o nosso algoritmo não altera a data ou horas das aulas já definidas no horário, mas distribui as aulas entre as salas disponíveis de forma otimizada.

## Utilização de Algoritmos de Otimização

Para resolver o problema descrito, é comum utilizar algoritmos de otimização, especialmente em contextos onde há múltiplos objetivos a serem otimizados simultaneamente. Deste modo, foram aplicadas algumas técnicas relevantes para a resolução do problema mencionado anteriormente:

* ***Objetivo únicoOptimization***: Foca-se na otimização de um único objetivo. Neste contexto, utilizamos:
  + **Algoritmos Genéticos (Genetic Algorithms)**: Adequados para explorar soluções num espaço de pesquisa complexo, como o problema do número de sobrelotações ou de requisitos não cumpridos. Estes algoritmos são capazes de encontrar soluções próximas do ótimo global, mesmo em problemas com múltiplas restrições e objetivos conflitantes.
* ***Multi-objective Optimization:*** Este tipo de otimização considera mais do que um objetivo simultaneamente, assim desta forma é possível minimizar o número de sobrelotações e ao mesmo tempo de garantir que todas as características necessárias das salas sejam atendidas. Esta abordagem é essencial para equilibrar diferentes necessidades e restrições, oferecendo opções de compromisso para decisões mais informadas. Neste caso, aplicamos:
  + **NSGA-II (*Nondominated Sorting Genetic Algorithm II*)**: Este algoritmo é eficiente para resolver problemas de otimização *multi-objective*, pois pode encontrar um conjunto de soluções ótimas que equilibram os diferentes objetivos. Desta forma, permite ao utilizador escolher a solução que melhor atende às suas necessidades específicas, considerando todos os critérios de avaliação.

## Frameworks Utilizadas

Neste trabalho, utilizámos diversas bibliotecas que foram fundamentais para desenvolver este trabalho, tais como *pandas*, *numpy*, *matplotlib*, entre outras.

A biblioteca *pandas* foi utilizada para a manipulação e análise de dados, especialmente no carregamento e processamento dos ficheiros de entrada, criação de tabelas de horários e manipulação de grandes conjuntos de dados.

A biblioteca *numpy* desempenhou um papel importante na computação numérica eficiente, permitindo realizar operações matemáticas e estatísticas complexas necessárias para a análise de dados e implementação dos algoritmos de otimização.

Para a visualização dos dados, utilizámos a biblioteca *matplotlib*, que nos permitiu criar gráficos e visualizações para apresentar os resultados da análise e da otimização dos horários.

### JMetalPy

Para este trabalho decidimos utilizar a biblioteca *JMetalPy* para a implementação do problema de otimização devido à sua capacidade de lidar com problemas complexos de otimização multi-objetivo de forma eficiente e estruturada. A JMetalPy oferece suporte para a resolução de problemas que envolvem múltiplos critérios de avaliação simultaneamente, com algoritmos otimizados que garantem um desempenho rápido e soluções de alta qualidade. Além disso, esta biblioteca possui ferramentas que permitem personalizar a abordagem de otimização conforme as necessidades especificadas.

### Flask

De forma a realizar a ligação/conexão do algoritmo de otimização com a interface de utilizador, utilizámos a *microframework Flask*. O *Flask* permite a criação de aplicações web de maneira simples e eficiente, facilitando a comunicação entre o *frontend* e o *backend*.

Desta forma, a arquitetura do sistema desenvolvido é definida pelo *backend* que é responsável por executar o algoritmo de otimização, processando os dados e gerando o novo horário otimizado. O *frontend*, por sua vez, é a interface de utilizador desenvolvida que permite a recolha dos dados e a visualização dos resultados. O *Flask* lida com as requisições HTTP, enviando os dados do *frontend* para o *backend* e retornando os resultados da otimização para serem apresentados ao utilizador.

## Implementação

Definição do Problema de Otimização como Integer Problem:

* **Variáveis**: Correspondem a slots de horários para cada aula.
* **Valor atribuídos**: o valor atribuído às variáveis corresponde ao seu índice da sala atribuída que vai de 0 até n-1, em que n corresponde ao número de salas disponíveis no “caratericasdassalas”. Além disso, pode ser atribuído o valor -1, caso a aula não tenha sido atribuída uma sala.
* **Objetivos**: Dois objetivos que correspondem aos critérios de avaliação mencionados:

1. Minimizar sobrelotações.
2. Minimizar o incumprimento dos requisitos das salas.

* **Restrições**: Para simplificação do algoritmo inicial, não consideramos restrições explícitas.

### Avaliação da Solução

A função evaluate() tem como objetivo avaliar uma solução com base em dois critérios: excesso de capacidade e necessidades não satisfeitas. O processo é o seguinte:

* Itera sobre cada turma do calendário;
* Verifica se a sala atribuída atende às necessidades da turma;
* Calcula o número de instâncias de excesso de capacidade;
* Calcula o número de requisitos não satisfeitos;
* Atualiza os valores do objetivo da solução com base no tipo de otimização selecionado (objetivo único ou multi-objetivo);

### Criação da Solução Inicial

A função create\_solution() tem como objetivo criar uma solução inicial:

* Ordena as salas com base na capacidade;
* Itera sobre cada turma no horário;
* Tenta atribuir uma sala adequada com base no tamanho da turma e requisitos;
* Se não encontrar uma sala adequada, atribui uma sala aleatória;
* Se a turma não precisar de uma sala, atribui o valor –1;

## Justificações Técnicas

### Dificuldades no Projeto

Inicialmente, tentámos implementar todo o projeto em JavaScript, desde a parte *frontend* e *backend*, pois a recolha dos dados e a visualização dos resultados paera cada um dos critérios de avaliação de um dado horário introduzido pelo utilizador, eram feitas nesta linguagem. No entanto, enfrentámos dificuldades significativas, pois não íamos conseguir aceder aos algoritmos de otimização necessários para resolver o problema de forma eficiente.

#### Implementação em JavaScript

A primeira abordagem foi desenvolver a solução completamente em JavaScript. Esta escolha foi baseada na facilidade de integração com as ferramentas de recolha de dados e visualização que já estavam estabelecidas no programa. No entanto, percebemos que a implementação de algoritmos de otimização complexos em JavaScript seria inviável devido à falta de bibliotecas robustas e adequadas para esse propósito.

#### JMetal em Java

Após uma longa pesquisa, explorámos a utilização da biblioteca JMetal em Java, pois encontramos uma *framework* que possibilitava a ligação entre JavaScript e Java. A ideia era usar Java para executar os algoritmos de otimização e JavaScript para a interface do utilizador e visualização dos resultados. Contudo, esta abordagem também apresentou problemas pois a integração entre as duas linguagens não foi tão fluida quanto esperávamos, resultando em problemas de comunicação e sincronização de dados.

Depois de descobrirmos a biblioteca *JMetal* em Java, encontramos também a sua versão em Python, a *JMetalPy*. Assim, decidimos procurar uma *framework* que permitisse a ligação entre JavaScript e Python.

#### Biblioteca JMetalPy

A *JMetalPy* é uma biblioteca que se destaca pela sua capacidade de lidar com problemas complexos de otimização multi-objetivo de maneira eficiente e estruturada.

Esta bibliotecaoferece um conjunto de funcionalidades que a tornam uma escolha robusta para problemas de otimização. Algumas de suas principais características incluem:

* **Suporte a vários algoritmos de otimização**:
  + Algoritmos evolutivos como NSGA-II, NSGA-III, SPEA2, MOEA/D, entre outros.
  + Algoritmos baseados em enxame, como PSO (P*article Swarm Optimization*).
  + Algoritmos híbridos e customizáveis, permitindo aos utilizadores combinar diferentes técnicas de otimização.
* **Facilidade de integração**:
  + Integra-se facilmente com outras bibliotecas de Python, como NumPy e Pandas, que são amplamente utilizadas para análise de dados e manipulação numérica.
  + Ferramentas de visualização de fronteiras de Pareto e outros resultados de otimização, facilitando a análise dos resultados.

#### Adoção do Flask e do JMetalPy

Perante os desafios encontrados, decidimos utilizar a *framework Flask* para criar uma API que permitisse a comunicação entre o *backend*, onde os algoritmos de otimização seriam executados, e o *frontend*, desenvolvido em JavaScript. Esta decisão foi motivada pela flexibilidade e eficiência do *Flask* na criação de aplicações web.

#### Compreensão do Funcionamento dos Algoritmos

Compreender a natureza dos algoritmos foi uma das nossas principais dificuldades no início, pois cada algoritmo tem características específicas. Assim, foi crucial estudar e entendê-los para garantir a sua correta implementação e aplicação. Dedicamos tempo significativo para entender como funcionavam e como poderíamos implementá-los de maneira eficiente.

### Escolha dos Algoritmos

Para este trabalho, decidimos que uma abordagem eficaz para determinar a melhor forma de resolver o problema seria testar algoritmos relevantes, eficientes e de diferentes tipos de otimização. Desta forma, foi possível avaliar o desempenho e comportamento de cada algoritmo em relação às especificidades do problema de otimização.

Optámos por implementar os algoritmos como *Integer Problems* para garantir precisão e viabilidade nas soluções. Além disso, para uma análise abrangente, selecionámos algoritmos representativos de diferentes paradigmas de otimização. Esta abordagem permitiu-nos comparar a eficácia de cada método e identificar quais técnicas são mais adequadas para lidar com as múltiplas restrições e objetivos conflitantes.

### Definição dos Objetivos e Criação da Solução Inicial

Anteriormente, havíamos definido quatro objetivos para o problema (correspondentes aos critérios utilizados para a avaliação dos horários), mas a criação da solução inicial (create\_solution) era feita de forma aleatória. Esta abordagem resultava numa convergência lenta do algoritmo para uma solução ótima. Para melhorar a eficiência e a qualidade das soluções, ajustamos o processo de criação da solução inicial.

## Análise de performance dos diferentes tipos de otimização

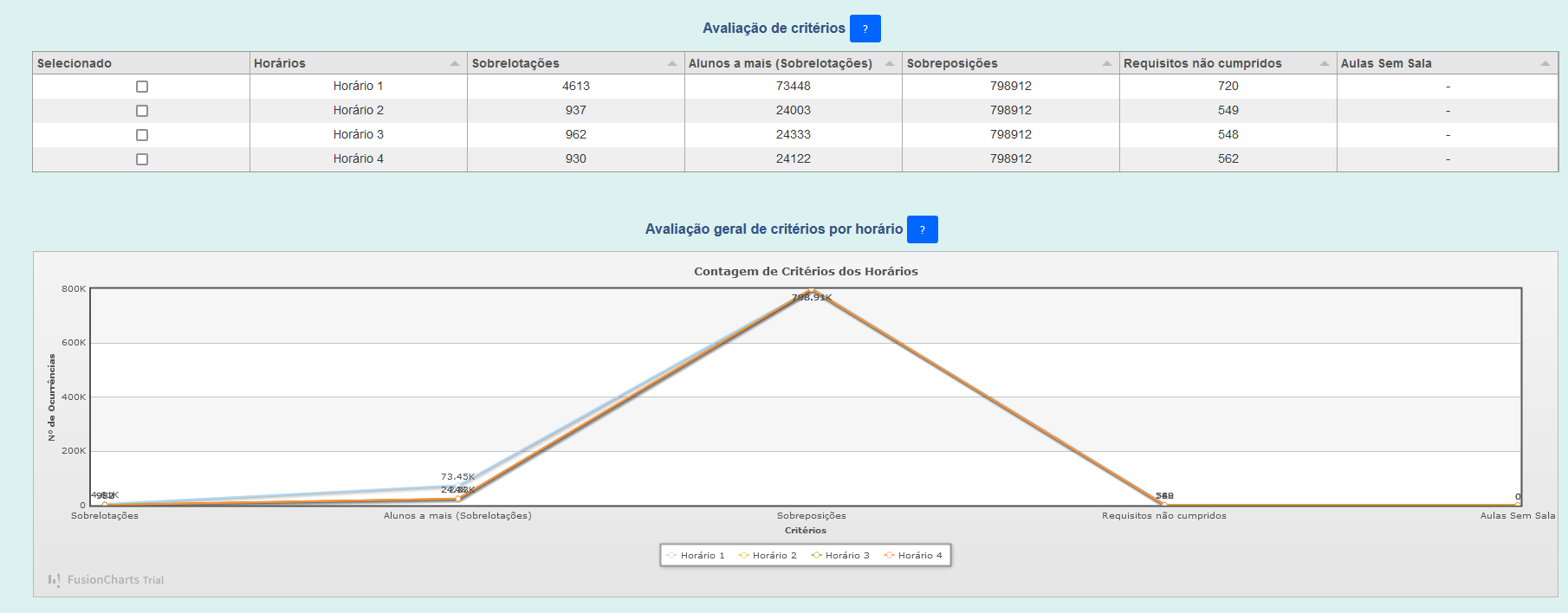


Figura 1 - Análise dos diferentes tipos de otimização

Para analisar a performance dos diferentes tipos de otimização, serão comparados quatro horários diferentes, onde "Horário 1" é o horário base e "Horário 2", "Horário 3" e "Horário 4" são horários otimizados (através da otimização do Horário 1 por diferentes tipos de otimização: multi-objetivo, Objetivo único por requisitos, e Objetivo único por sobrelotações, respectivamente.

### Sobrelotações

* Horário 1 (Base): 4613 sobrelotações.
* Horário 2 (Multi-objetivo): 937 sobrelotações.
* Horário 3 (Objetivo único por Requisitos): 962 sobrelotações.
* Horário 4 (Objetivo único por Sobrelotações): 930 sobrelotações.

O horário 4 apresenta o menor número de sobrelotações, seguido pelo horário 2 e pelo horário 3, indicando uma melhoria significativa em comparação com o horário base. Estes resultados alinham-se com o esperado, uma vez que foi a otimização apenas focada nas sobrelotações que obteve melhores resultados.

### Requisitos Não Cumpridos

* Horário 1 (Base): 720 requisitos não cumpridos.
* Horário 2 (Multi-objetivo): 549 requisitos não cumpridos.
* Horário 3 (Objetivo único por Requisitos): 548 requisitos não cumpridos.
* Horário 4 (Objetivo único por Sobrelotações): 562 requisitos não cumpridos.

O horário 3 tem o menor número de requisitos não cumpridos, seguido de perto pelo horário 2. O horário 4 ainda melhora em relação ao horário base, mas não tanto quanto os outros dois otimizados. Mais uma vez, estes resultados alinham-se com o esperado, uma vez que foi a otimização com foco apenas no cumprimento de requisitos que levou a melhores resultados neste campo.

### Alunos a Mais (Sobrelotações)

* Horário 1 (Base): 73.448 alunos.
* Horário 2 (Multi-objetivo): 24.003 alunos.
* Horário 3 (Objetivo único por Requisitos): 24.333 alunos.
* Horário 4 (Objetivo único por Sobrelotações): 24.122 alunos.

Os horários otimizados mostram uma redução substancial no número de alunos afetados pelas sobrelotações, com o horário 2 apresentando a maior redução.Uma imagem com texto, captura de ecrã, file, número

Descrição gerada automaticamente

Figura 2 - Fronteira de Pareto e dispersão dos pontos da Otimização multi-objetivo

A análise das métricas indica que os horários otimizados apresentam melhorias significativas em comparação com o horário base. Em termos de sobrelotações e alunos a mais, o horário 4 é o mais eficaz. No entanto, considerando um equilíbrio entre sobrelotações e requisitos não cumpridos, o horário 2 (multi-objetivo) parece ser a solução mais equilibrada e eficiente.

## Conclusão

Em suma, conseguimos definir e encontrar uma forma de mitigar o problema de otimização de horários da universidade de maneira eficiente e estruturada. Através da definição de variáveis, objetivos e métodos de avaliação, abordámos de forma holística os desafios associados, garantindo a melhor utilização dos recursos disponíveis. Para futuras implementações, podemos considerar a inclusão de restrições adicionais, ter em conta a atribuição das horas de início ou fim das aulas ou a utilização de técnicas avançadas de otimização, como algoritmos híbridos que combinam diferentes metodologias.

## Bibliografia

Lemos, A., Melo, F. S., Monteiro, P. T., & Lynce, I. (2019). Room usage optimization in timetabling: A case study at Universidade de Lisboa. Operations Research Perspectives, 6. <https://doi.org/10.1016/j.orp.2018.100092>

Alghamdi, H., Alsubait, T., Alhakami, H., & Baz, A. (2020). A Review of Optimization Algorithms for University Timetable Scheduling. Engineering, Technology and Applied Science Research, 10(6). <https://doi.org/10.48084/etasr.3832>

Bashab, A., Ibrahim, A. O., Hashem, I. A. T., Aggarwal, K., Mukhlif, F., Ghaleb, F. A., & Abdelmaboud, A. (2023). Optimization Techniques in University Timetabling Problem: Constraints, Methodologies, Benchmarks, and Open Issues. Computers, Materials and Continua, 74(3). https://doi.org/10.32604/cmc.2023.034051

Durillo, J. J., & Nebro, A. J. (n.d.). jMetalPy: Python version of the jMetal framework. Acedido em 28 de junho, 2024, de <https://jmetal.github.io/jMetalPy/index.html>