Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Integração de Estratégias Meta-heurísticas na Plataforma ItPlatPSP para Problemas de Escalonamento de Projetos com Restrições de Recursos.

Kassio Rodrigues Ferreira

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Samuel Souza Brito

COORIENTAÇÃO:
Janniele Aparecida Soares Araujo

Fevereiro, 2026
João Monlevade–MG

Kassio Rodrigues Ferreira

Integração de Estratégias Meta-heurísticas na Plataforma ItPlatPSP para Problemas de Escalonamento de Projetos com Restrições de Recursos.

Orientador: Samuel Souza Brito

Coorientador: Janniele Aparecida Soares Araujo

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro de 2026

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

F383i Ferreira, Kassio Rodrigues.
Integração de estratégias meta-heurísticas na plataforma ItPlatPSP para problemas de escalonamento de projetos com restrições de recursos. [manuscrito] / Kassio Rodrigues Ferreira. - 2026.
75 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Samuel Brito.
Coorientadora: Profa. Dra. Janniele Araujo.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Sistemas de Informação .

1. Desenvolvimento de software. 2. Engenharia de software. 3. Heurística. 4. Otimização combinatória. 5. Projetos - Planejamento. I. Brito, Samuel. II. Araujo, Janniele. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004.41:005.81

Bibliotecário(a) Responsável: Flavia Reis - CRB6/2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Kassio Rodrigues Ferreira

Integração de Estratégias Meta-heurísticas na Plataforma ItPlatPSP para Problemas de Escalonamento de Projetos com Restrições de Recursos

Monografia apresentada ao Curso de Sistemas de Informação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em 05 de março de 2026.

Membros da banca

Dr. Samuel Souza Brito - Orientador - Universidade Federal de Ouro Preto
Dra. Janniele Aparecida Soares Araújo - Coorientadora - Universidade Federal de Ouro Preto
Dr. Rafael Frederico Alexandre - Universidade Federal de Ouro Preto
Dr. Roberto Gomes Ribeiro - Universidade Federal de Ouro Preto

Samuel Souza Brito, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 06/05/2026.

Documento assinado eletronicamente por **Samuel Souza Brito**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 06/05/2026, às 13:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1103060** e o código CRC **D81F2D3A**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Dedico este trabalho aos meus pais, Elenilda e Juacir, e aos meus irmãos Síria e Italo.

Agradecimentos

A realização deste trabalho e de toda a minha jornada acadêmica só foi possível graças ao apoio de diversas pessoas, às quais expresso minha profunda gratidão.

Primeiramente, agradeço à minha família: aos meus pais, Elenilda e Juacir, e aos meus irmãos, Síria e Italo, pelo apoio e incentivo ao longo de tantos anos.

Agradeço aos meus professores, Janniele e Samuel, pela orientação, apoio, parceria e paciência ao longo de tantos anos, desde os projetos de iniciação científica até o presente projeto.

Aos meus colegas de curso, agradeço a parceria, a troca de experiências e o companheirismo ao longo de nossa jornada.

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para minha trajetória até aqui.

"And no one knows the where's or whys
But something stirs and something tries
And starts to climb towards the light."

— Pink Floyd,
in: Echoes.

Resumo

Este trabalho aborda a lacuna existente entre métodos de resolução para os Problemas de Escalonamento de Projetos com Restrição de Recursos (RCPSPs) e sua aplicação prática por usuários não especialistas, bem como a escassez de ambientes exploratórios para a experimentação dessas abordagens. O objetivo deste trabalho foi expandir as capacidades de resolução da plataforma ItPlatPSP por meio da integração de meta-heurísticas baseadas nos métodos *Variable Neighborhood Descent* (VND) e *Variable Neighborhood Search* (VNS). Foram implementadas diferentes versões do VNS, cada uma com estratégias específicas de exploração de estruturas de vizinhança, compatíveis com as distintas variantes do problema suportadas pela plataforma. Experimentos computacionais foram conduzidos utilizando instâncias padrão na avaliação de RCPSPs. Os resultados obtidos indicaram que os métodos *General* VNS e VND apresentam desempenho mais equilibrado entre qualidade de solução e robustez, enquanto *Skewed* VNS e *Smart* VNS demonstraram maior sensibilidade em instâncias de maior complexidade. Os experimentos também evidenciaram relação direta entre as estratégias de troca de vizinhança e a participação dos operadores ao longo da busca. Como resultado, a plataforma passa a oferecer um conjunto mais robusto e parametrizável de ferramentas de apoio à decisão, além de ampliar sua modularidade por meio da padronização de componentes meta-heurísticos baseados em contratos e interfaces.

Palavras-chaves: Escalonamento de Projetos. Otimização Combinatória. Heurísticas. Meta-heurísticas. Projeto de Software. Plataforma Web.

Abstract

This work addresses the gap between solution methods for Resource-Constrained Project Scheduling Problems (RCPSPs) and their practical application by non-specialist users, as well as the scarcity of exploratory environments for experimenting with such approaches. The objective of this study was to expand the solving capabilities of the ItPlatPSP platform by integrating metaheuristics based on the Variable Neighborhood Descent (VND) and Variable Neighborhood Search (VNS) methods. Different VNS versions were implemented, each employing specific neighborhood structure exploration strategies, compatible with the different problem variants supported by the platform. Computational experiments were conducted using standard benchmark instances for RCPSP evaluation. The results indicated that the General VNS and VND methods achieved a more balanced performance in terms of solution quality and robustness, whereas Skewed VNS and Smart VNS showed greater sensitivity when applied to higher-complexity instances. The experiments also revealed a direct relationship between neighborhood change strategies and the participation of operators throughout the search process. As a result, the platform now offers a more robust and parameterizable set of decision-support tools, while further enhancing its modularity through the standardization of metaheuristic components based on contracts and interfaces.

Key-words: Project Scheduling. Combinatorial Optimization. Heuristics. Metaheuristics. Software Design. Web Platform.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACO Otimização da colônia de formigas (*Ant Colony Optimization*)

BKS Melhor Solução Conhecida *(Best Known Solution)*

BVNS Busca em Vizinhança Variável Básica (*Basic Variable Neighborhood Search*)

CPD Duração do Caminho Crítico (*Critical Path Duration*)

CPM Método do Caminho Crítico (*Critical Path Method*)

GRASP Procedimento de Busca Adaptativa Aleatória Gulosa GRASP *(Greedy Randomized Adaptive Search Procedure*)

GVNS Busca em Vizinhança Variável Geral (*General Variable Neighborhood Search*)

ItPlatPSP Plataforma Interativa para Problemas de Escalonamento de Projetos (*Interactive Platform for Project Scheduling Problems*)

LAHC Método da Subida com Aceitação Tardia (*Late Acceptance Hill-Climbing*)

MILP Programação Linear Inteira Mista (*Mixed Integer Linear Programming*)

MMLIB *Multiple Modes Library*

MMRCMPSP Problema de Escalonamento de Projetos com Restrição de Recursos em Múltiplos Modos e Múltiplos Projetos (*Multi-Mode Resource-Constrained Multi-Project Scheduling Problem*)

MMRCPSP Problema de Escalonamento de Projetos com Restrição de Recursos com Múltiplos Modos (*Multi Mode Resource Constrained Project Scheduling Problem*)

MS tempo total de execução de um projeto (*Makespan*)

MS-RCPSP Problema de Escalonamento de Projetos com Recursos Multi-Objetivo (*Multi-Skill Resource-Constrained Project Scheduling Problem*)

PSPLIB *Project Scheduling Problem Library*

RCPSP Problemas de Escalonamento de Projetos com Restrição de Recursos (*Resource Constrained Project Scheduling Problem*)

RVNS Busca em Vizinhança Variável Reduzida (*Reduced Variable Neighborhood Search*)

SGS Esquema de Geração de Cronograma (*Schedule Generation Scheme*)

SILS Busca Local Iterada Inteligente (*Smart Iterated Local Search*)

Smart VNS Busca em Vizinhança Variável Inteligente (*Smart Variable Neighborhood Search*)

SMRCPSP Problema de Escalonamento de Projetos com Restrição de Recursos com Modo Único (*Single Mode Resource Constrained Project Scheduling Problem*)

SVNS Busca em Vizinhança Variável Inclinada (*Skewed Variable Neighborhood Search*)

TMS tempo término do último projeto (*Total Makespan*)

TPD tempo de atraso de todos os projetos (*Total Projects Delay*)

VND Descida em Vizinhança Variável (*Variable Neighborhood Descent*)

VNS Busca em Vizinhança Variável (*Variable Neighborhood Search*)

Sumário

1 Introdução

Classificados como NP-difícil (Blazewicz, Lenstra e Kan, 1983), os Problemas de Escalonamento de Projetos com Restrição de Recursos (*Resource Constrained Project Scheduling Problem*) (RCPSP), são amplamente discutidos na literatura dada a sua natureza complexa e relevância em diversos setores que dependem do planejamento e gestão de projetos. A aplicação prática destes problemas está presente em setores como engenharia civil, planejamentos industriais, desenvolvimento de software, planejamento de eventos entre outros (Demeulemeester e Herroelen, 2002).

Embora os RCPSP possuam uma literatura ampla com diversas abordagens de resolução consolidadas, a aplicação prática dos métodos mais sofisticados ainda é restrita a ambientes acadêmicos. Indivíduos não especialistas no tema, mas que trabalham ou desejam explorar técnicas de escalonamento no campo de planejamento e gestão de projetos, podem se deparar com barreiras como a necessidade de compreender formulações avançadas e conhecimento prévio em técnicas de modelagem computacional.

A Plataforma Interativa para Problemas de Escalonamento de Projetos (*Interactive Platform for Project Scheduling Problems*) (ItPlatPSP), concebida ao longo do projeto de iniciação científica "Uma plataforma interativa para os problemas de escalonamento de projetos com restrição de recursos" (elaborado ao longo dos programas PIP-1S-2022-23, PIVIC-1S-2023/24 e PIVIC-2S-2024/25) foi proposta com o objetivo de reduzir essas barreiras, oferecendo um ambiente acessível que integra recursos de modelagem e métodos de resolução em uma ambiente unificado dedicado aos RCPSP.

Até então, a plataforma integrava métodos de resolução exatos baseados em Programação Linear Inteira Mista (*Mixed Integer Linear Programming*) (MILP) (Araujo et al., 2020) e uma meta-heurística baseada no Método da Subida com Aceitação Tardia (*Late Acceptance Hill-Climbing*) (LAHC) (Burke e Bykov, 2017). A proposta deste trabalho foi ampliar as abordagens resolução disponíveis na plataforma por meio da integração de novas meta-heurísticas baseadas nos métodos de Descida em Vizinhança Variável (*Variable Neighborhood Descent*) (VND) e Busca em Vizinhança Variável (*Variable Neighborhood Search*) (VNS) (Mladenović e Hansen, 1997).

1.1 O problema de pesquisa

Segundo Araujo (2019) uma solução para um RCPSP consiste em encontrar um cronograma de execução viável para as atividades de um projeto respeitando as restrições impostas ao problema. Conforme Kolisch, Sprecher e Drexl (1995) o grau de complexidade

de um RCPSP está associado a dois fatores principais. O primeiro diz respeito a densidade das relações de precedência entre as atividades do projeto enquanto o segundo está relacionado com a disponibilidade e consumo de recursos disponíveis para o projeto.

As relações de precedência determinam que uma atividade não pode ser iniciada até que todas as atividades predecessoras tenham sido concluídas. A representação mais comum para relações de precedência nos RCPSP é dada meio de grafos acíclicos $G = (N, A)$, onde cada nó $n \in N$ está associado a um ou mais arcos $a \in A$.

A realização de projetos demanda recursos de diferentes tipos como financeiros, mão de obra, maquinário, equipamento, energia e espaço (Demeulemeester e Herroelen, 2002). No contexto dos RCPSP abordados neste trabalho, três tipos de recursos são considerados:

- recursos renováveis: as quantidades disponíveis se renovam periodicamente (hora, dia, semana, mês), e a disponibilidade por período é constante, exemplo: mão de obra, maquinário, equipamentos e espaço;
- recursos não renováveis: o volume de recursos disponíveis é limitado para todo o projeto, exemplo: dinheiro, energia e matéria prima;
- recursos globais: recursos renováveis cuja disponibilidade é compartilhada entre múltiplos projetos.

Diferentes variantes do problema são exploradas pela literatura, cada uma com características próprias. Este trabalho se concentra nas seguintes:

- Problema de Escalonamento de Projetos com Restrição de Recursos com Modo Único (*Single Mode Resource Constrained Project Scheduling Problem*) (SMRCPSP) (Kolisch e Sprecher, 1997): admite um único modo de execução por atividade, com duração e alocação de recursos fixas;
- Problema de Escalonamento de Projetos com Restrição de Recursos com Múltiplos Modos (*Multi Mode Resource Constrained Project Scheduling Problem*) (MMRCPSP) (Kolisch e Sprecher, 1997): uma generalização do SMRCPSP que permite múltiplos modos de execução para cada atividade, cada qual com requisitos específicos de tempo e recursos;
- Problema de Escalonamento de Projetos com Restrição de Recursos em Múltiplos Modos e Múltiplos Projetos (*Multi-Mode Resource-Constrained Multi-Project Scheduling Problem*) (MMRCMPSP) (Wauters et al., 2016): versão que abrange múltiplos projetos, com a possibilidade de compartilhamento de recursos renováveis entre eles.

1.2 Objetivos

O objetivo geral deste trabalho foi ampliar os métodos de resolução da plataforma ItPlatPSP por meio da integração de novas abordagens meta-heurísticas. Para alcançar o objetivo geral foram definidos os seguintes objetivos específicos:

a) Selecionar, com base na literatura, abordagens meta-heurísticas com potencial de diversificar as já existentes na plataforma;

b) Implementar protótipos para validar as abordagens selecionadas;

c) Incorporar as abordagens validadas na prototipação à biblioteca de resolvedores da plataforma;

d) Realizar experimentos computacionais para análise de desempenho;

e) Incorporar o novo método de resolução à plataforma.

1.3 Organização do trabalho

O restante deste trabalho é organizado como se segue. O Capítulo 2 apresenta a fundamentação teórica e os trabalhos relacionados ao problema abordado. O Capítulo 3 descreve os procedimentos metodológicos adotados, incluindo os ambientes e a configuração dos experimentos. O Capítulo 4 detalha o desenvolvimento das abordagens propostas e sua implementação. O Capítulo 5 apresenta e discute os resultados obtidos, bem como a integração das soluções à plataforma desenvolvida. Por fim, o Capítulo 6 sintetiza as considerações finais, destacando as principais contribuições, as limitações identificadas e as propostas para trabalhos futuros.

2 Revisão bibliográfica

Este capítulo apresenta a revisão bibliográfica que fundamenta o desenvolvimento deste trabalho. Inicialmente, são discutidos aspectos relacionados à representação do problema, ao método do caminho crítico, à definição da função objetivo e às formas de representação de soluções. Em seguida, são abordadas meta-heurísticas aplicadas a problemas de otimização combinatória. Por fim, são apresentados trabalhos relacionados, incluindo bibliotecas de instâncias de referência e projetos de software voltados à modelagem e resolução dos RCPSP.

2.1 Representação do problema

A representação computacional do problema neste trabalho é inspirada nas abordagens propostas por Araujo et al. (2020) e Asta et al. (2016), devido à sua versatilidade, que viabiliza o uso da mesma representação para diferentes variantes do problema. Essa formulação caracteriza uma instância do problema por meio dos seguintes elementos:

- $\mathcal{P}$: Conjunto de projetos que compõem a instância.
- $\mathcal{J}$: Conjunto de atividades que constituem cada projeto.
- $\mathcal{M}_j$: Conjunto de modos de execução disponíveis para a atividade j.
- $\mathcal{R}$: Conjunto de recursos do tipo renovável.
- $\mathcal{K}$: Conjunto de recursos do tipo não renovável.
- $\mathcal{S}$: Conjunto de relações de precedência entre as atividades.
- d_{jm}: Duração da atividade j quando executada no modo m.
- q_{rjm}: Demanda de recursos renováveis da atividade j no modo m.
- q_{kjm}: Demanda de recursos não renováveis da atividade j no modo m.
- σ_p: Instante de tempo de início do projeto p.
- a_p: Atividades artificiais (fictícias) que indicam o término do projeto p.
- $\mathcal{T}$: Conjunto de instantes de tempo discretos que definem o horizonte temporal.

O conjunto de projetos $\mathcal{P}$, cada um composto por atividades $\mathcal{J}$, que por sua vez podem ser executadas em diferentes modos $\mathcal{M}_j$; os recursos, classificados como renováveis $\mathcal{R}$ ou não renováveis $\mathcal{K}$; e as relações de precedência $\mathcal{S}$ entre as atividades.

Cada atividade j, quando executada no modo m, está associada a uma duração d_{jm} e a um consumo de recursos de q_{rjm} e q_{kjm}, correspondentes às demandas de recursos renováveis e não renováveis, respectivamente. A representação também inclui os tempos de início do projeto σ_p e atividades artificiais a_p (fictícias) que indicam o fim de cada projeto, juntamente com o conjunto $\mathcal{T} \subset \mathbb{Z}^+$ de instantes de tempo discretos que definem o horizonte temporal estimado o cronograma de execução das atividades da instância.

2.2 Método do Caminho Crítico

O Método do Caminho Crítico (*Critical Path Method*) (CPM), originalmente proposto por Kelley e Walker (1959), é uma técnica utilizada para identificar atividades cujos atrasos afetam diretamente o tempo total de conclusão do projeto. Por meio do CPM, o limite inferior teórico para o tempo de conclusão de um projeto p pode ser obtido assumindo que todas as atividades críticas sejam executadas sem atraso, ou seja, considerando apenas as relações de precedência. O valor deste limite inferior é denotado como a Duração do Caminho Crítico (*Critical Path Duration*) (CPD).

2.3 Função objetivo

A função objetivo mais comum para o RCPSP busca minimizar o tempo total de execução de um projeto (*Makespan*) (MS)(Brucker et al.; Demeulemeester e Herroelen; Kolisch, Sprecher e Drexl, 1999, 2002, 1995). Formalmente, o cálculo do MS é demonstrado pela Equação 2.1, onde σ_p é o tempo de inicio do cronograma e f_p e o tempo de conclusão.

$$MS = f_p - \sigma_p \tag{2.1}$$

Para definir um objetivo generalizado para os MMRCMPSP, Wauters et al. (2016) propôs uma função objetivo com dois componentes: minimizar o tempo de atraso de todos os projetos (*Total Projects Delay*) (TPD) e o tempo término do último projeto (*Total Makespan*) (TMS). O TPD é obtido conforme 2.3, onde PD_p é o atraso de um projeto em relação ao seu caminho crítico CPD_p. O TMS, por sua vez é dado por 2.4, onde é calculada a diferença entre o maior tempo de conclusão e o menor tempo de início entre os projetos $p \in P$.

$$PD_p = MS_p - CPD_p \tag{2.2}$$

$$TPD = \sum_{p \in P} PD_p \tag{2.3}$$

$$TMS = \max_{p \in P} f_p - \min_{p \in P} \sigma_p \tag{2.4}$$

Essa abordagem unificada é definida pela competição MISTA 2013 (Wauters et al., 2016), onde o TPD é o principal e o TMS é usado para desempate.

2.4 Representação de soluções

A representação computacional da solução de um RCPSP é um componente fundamental para a implementação de métodos de resolução, pois define a forma na qual os cronogramas devem ser construídos. De acordo com Demeulemeester e Herroelen (2002), a representação mais adequada pode variar dependendo do tipo de abordagem de solução adotada.

As representações diretas, baseadas em cronogramas, associam cada atividade a seu modo de execução e tempo de início. Elas são mais comuns em métodos exatos por apresentarem um formato próximo à definição das variáveis de decisão do problema (Asta et al., 2016).

Para abordagens heurísticas e meta-heurísticas, normalmente são utilizadas representações baseadas em sequências, que permitem maior flexibilidade na construção e avaliação de soluções. Tais representações são especialmente adequadas para tarefas intensivas, como a geração de vizinhanças a partir de uma solução de entrada (Asta et al., 2016). Entretanto, para avaliar tais representações, é necessária uma conversão para um formato baseado em cronogramas, que pode ser realizada usando algoritmos como o Esquema de Geração de Cronograma (*Schedule Generation Scheme*) (SGS) (Węglarz e Hillier, 1999).

Os resolvedores da ItPlatPSP adotam ambos os formatos no processo de resolução. Nos métodos exatos, o formato de solução padrão é o baseado em cronograma, representado como uma lista $S = \{(p, j, m, t)...\}$, onde p denota o projeto, j a atividade, m o modo selecionado e t o tempo de início do cronograma. No método heurístico, ambos os formatos são utilizados. O formato baseado em sequência é representado como $S = (\pi, \mathcal{M})$, onde π é a sequência de alocação de atividades e $\mathcal{M}$ é o conjunto de modos atribuídos a cada atividade em π. Os operadores de vizinhança aplicam os movimentos sobre o formato em sequência e, em seguida, para que a nova solução seja avaliada é feita a conversão para o formato baseado em cronograma por meio de um algoritmo SGS.

2.5 Meta-heurísticas aplicadas a problemas de otimização combinatória

Segundo Papadimitriou e Steiglitz (1982), problemas de Otimização Combinatória têm como finalidade identificar um objeto pertencente a um conjunto finito ou infinitamente enumerável, sendo esse objeto frequentemente representado por um número inteiro, um subconjunto, uma permutação ou uma estrutura de grafo. Nesse contexto, a busca consiste em selecionar, entre todas as configurações possíveis, aquela que melhor satisfaz o critério estabelecido.

Conforme Blum e Roli (2003) um problema de Otimização Combinatória $P = (S, f)$ pode ser definido como:

- um conjunto de variáveis $X = \{x_1, ..., x_n\}$;
- domínios associados a essas variáveis $D_1, ..., D_n$;
- um conjunto de restrições;
- uma função objetivo f a ser minimizada (ou maximizada) onde $f : D_1 \times \cdots \times D_n \to \mathbb{R}^+$;
- e o conjunto de soluções factíveis é dado por: $S = \{\{(x_1, v_1), ..., (x_n, v_n)\} \mid v_i \in D_i\}$.

Considerando um problema de minimização, uma solução $s^* \in S$ onde $f(s^*) \leq f(s)$ $\forall_s \in S$ é chamada de solução ótima global de (S, f), e o conjunto $S^* \subseteq S$ é chamado de conjunto de soluções ótimas.

Problemas de otimização combinatória frequentemente exigem a avaliação de um número elevado de possibilidades. De modo que, para problemas de larga escala, a busca pela solução ótima em tempo hábil se torna inviável dada a vasta dimensão do espaço de busca. Nestes casos, heurísticas são utilizadas para produzir soluções viáveis em prazos aceitáveis, oferecendo alternativas eficazes quando a busca exata se torna impraticável (Pearl; Rardin e Uzsoy, 1984, 2001).

Contudo, a aplicação de procedimentos heurísticos apresenta limitações fundamentais. Demeulemeester e Herroelen (2002) destaca como principais desvantagens a dificuldade de validação formal de desempenho e a forte dependência de análises empíricas. Tais avaliações, baseadas no comportamento médio, não capturam adequadamente cenários de pior caso, os quais podem apresentar resultados significativamente inferiores e comprometer a confiabilidade na consistência do método.

É comum que meta-heurísticas sejam empregadas como estruturas algorítmicas de alto nível que organizam e orientam o uso de heurísticas básicas, visando explorar o espaço de busca de forma mais eficiente e eficaz. Sob essa perspectiva, uma meta-heurística

fornece diretrizes gerais e independentes do problema, atuando como um arcabouço para o desenvolvimento de métodos heurísticos robustos e adaptáveis (Toulouse, Thulasiraman e Glover, 1999).

A busca local constitui a base de muitos métodos heurísticos empregados em problemas de otimização combinatória, sendo utilizada como um procedimento iterativo simples para obter boas soluções aproximadas. Sua eficácia depende de elementos como a estrutura de vizinhança, a função a ser minimizada e a solução inicial adotada (Martí, Sevaux e Sörensen, 2025). De modo geral, a busca local parte de uma solução arbitrária e realiza movimentos sucessivos para soluções vizinhas que apresentem melhoria, encerrando quando nenhum vizinho fornece redução adicional no valor da função objetivo.

De acordo com Johnson, Papadimitriou e Yannakakis (1988) uma estrutura de vizinhança define, para cada solução, um conjunto de soluções próximas que podem ser alcançadas por pequenas modificações. Formalmente, para um problema de minimização $g(s)$, uma vizinhança $N(s)$ contém todas as soluções acessíveis a partir de s, sendo um mínimo local caracterizado pela condição $g(s^*) \leq g(s'), \forall_{s'} \in N(s)$ (Voudouris e Tsang, 2003).

O Método da Subida com Aceitação Tardia (*Late Acceptance Hill-Climbing*) (LAHC) (Burke e Bykov, 2017) é um método de busca local que introduz um mecanismo simples de aceitação tardia, permitindo que soluções piores sejam aceitas quando o valor da solução candidata é melhor que o valor registrado algumas iterações atrás. Esse valor anterior é determinado por um único parâmetro ajustável do algoritmo, que define o tamanho do histórico considerado. Assim, em contraste com métodos de busca local tradicionais, nos quais o candidato é comparado apenas com a solução atual, o LAHC compara o valor da solução candidata com o valor de soluções anteriores, favorecendo a diversificação e reduzindo a probabilidade de estagnação em ótimos locais (Burke e Bykov, 2017).

A Busca em Vizinhança Variável (*Variable Neighborhood Search*) (VNS), proposta por Mladenović e Hansen (1997), fundamenta-se na exploração sistemática de diferentes estruturas de vizinhança ao longo do processo de busca. A formulação do método é baseada na premissa de que o ótimo local de uma vizinhança pode não o ser em outra, e um ótimo global é, necessariamente, um ótimo local em todas as vizinhanças possíveis. Para explorar essas propriedades, o algoritmo alterna entre três etapas principais: perturbação, busca local (melhoria) e mudança de vizinhança. Esse ciclo se repete até que um critério de parada predefinido seja atingido, estabelecendo um equilíbrio dinâmico entre a diversificação para escapar de ótimos locais e a intensificação em regiões promissoras do espaço de soluções.

Meta-heurísticas baseadas em busca local têm se mostrado uma alternativa eficaz para resolver variantes mais complexas do RCPSP, especialmente aquelas que envolvem múltiplos projetos. No Desafio MISTA 2013 (Wauters et al., 2016), diversas dessas estratégias demonstraram bom desempenho diante da complexidade do problema. Entre

os estudos que exploram essas estratégias, Asta et al. (2016) apresenta uma abordagem híbrida que combina busca em árvore de Monte Carlo com operadores de vizinhança e mecanismos de seleção adaptativa. Araujo (2019), que também aborda essa linha de investigação, analisa o impacto de diferentes vizinhanças em vários estágios do processo de busca e destaca como a escolha dos movimentos pode influenciar o desempenho heurístico ao longo da busca.

2.6 Trabalhos relacionados

Esta seção revisa as principais bibliotecas de instâncias usadas na literatura, e projetos de software que abordam os RCPSP.

2.6.1 Bibliotecas de instâncias e conjuntos de dados de referência

Diversos conjuntos de instâncias são empregados como base para a avaliação de abordagens de resolução para os RCPSP. Entre os mais conhecidos, destaca-se a *Project Scheduling Problem Library* (PSPLIB) (Kolisch e Sprecher, 1997), que reúne alguns dos conjuntos mais utilizados na literatura. Esses conjuntos contemplam instâncias dos tipos SMRCPSP e MMRCPSP, compostas por problemas reais e gerados artificialmente.

O conjunto de dados *Multiple Modes Library* (MMLIB) (Peteghem e Vanhoucke, 2014) adota o formato convencional da PSPLIB porém é exclusivo para a variante MMRCPSP e inclui problemas de maior complexidade comparado aos disponíveis na PSPLIB. Esta versão inclui mais tarefas, modos de execução e recursos, oferecendo um ambiente mais desafiador para a avaliação de métodos de resolução.

Outro conjunto de dados relevante é o proposto originalmente para o Desafio MISTA 2013 (Wauters et al., 2016), que continua sendo utilizado como referência para análises de desempenho. Composto exclusivamente por instâncias do tipo MMRCMPSP, esse acervo deriva da PSPLIB, mas introduz características adicionais, como a definição de múltiplos projetos e o compartilhamento de recursos entre eles. O formato adotado para esses dados também fornece uma base sólida para a definição de representações computacionais padronizadas, aspecto fundamental no desenvolvimento de ferramentas voltadas à resolução do RCPSP.

2.6.2 Projetos de software que abordam a modelagem e resolução de RCPSP

Em relação softwares abertos voltados aos RCPSP, os projetos disponíveis variam significativamente em termos de usabilidade, arquitetura e suporte a resolvedores. Alguns focam no ensino, enquanto outros oferecem alto desempenho e flexibilidade para pesquisa acadêmica através de ferramentas modulares. Contudo, a ausência de interfaces gráficas

interativas e a dependência de ambientes de programação nessas ferramentas criam lacunas de acessibilidade.

O software RESCON Deblaere, Demeulemeester e Herroelen (2011) é uma ferramenta educacional desenvolvida em C++ para ilustrar conceitos de escalonamento de projetos. A aplicação apresenta uma interface gráfica que inclui a visualização de grafos de precedência, perfis de consumo de recursos e gráficos de *Gantt*, e também conta com a compatibilidade nativa para instâncias no formato da PSPLIB. Em termos algorítmicos, suporta métodos exatos (*branch-and-bound*) e heurísticos, como Escalonamento de Lista e Busca Tabu. Apesar de permitir a integração de novos métodos, sua última atualização ocorreu em 2010, o que limita sua aplicação em contextos mais atuais.

Voltada ao Problema de Escalonamento de Projetos com Recursos Multi-Objetivo (*Multi-Skill Resource-Constrained Project Scheduling Problem*) (MS-RCPSP), a biblioteca de código aberto iMOPSE Gmyrek et al. (2024) oferece uma arquitetura modular em C++ que facilita a implementação de abordagens personalizadas. O ambiente disponibiliza meta-heurísticas clássicas, como Procedimento de Busca Adaptativa Aleatória Gulosa GRASP *(Greedy Randomized Adaptive Search Procedure*) (GRASP), Otimização da colônia de formigas (*Ant Colony Optimization*) (ACO) e Busca Tabu. Embora seja uma plataforma robusta para a comparação de algoritmos, o iMOPSE exige compilação local e carece de uma interface gráfica, operando exclusivamente via linha de comando ou integração com scripts C++, o que pode restringir sua acessibilidade.

Recentemente, a biblioteca PyJobShop Lan e Berkhout (2025) introduziu uma ferramenta implementada em Python para a modelagem e resolução de variantes dos RCPSP (incluindo MMRCPSP e MMRCMPSP) via Programação por Restrições. A ferramenta foca na flexibilidade de modelagem e experimentação sistemática, com suporte nativo aos resolvedores OR-Tools[1] e IBM CP Optimizer[2]. Embora gere visualizações estáticas de cronogramas e recursos, seu uso é estritamente orientado a *scripts*, sendo, portanto, uma ferramenta voltada ao desenvolvimento e à pesquisa, orientada a usuários especialistas e com conhecimento em programação.

[1] <github.com/google/or-tools>
[2] <ibm.com/docs/en/icos/22.1.0?topic=cp-optimizer>

3 Metodologia

A metodologia adotada compreende duas etapas principais: (i) o desenvolvimento e integração dos métodos à arquitetura existente e (ii) a avaliação experimental das abordagens implementadas.

3.1 Ambiente de desenvolvimento e integração

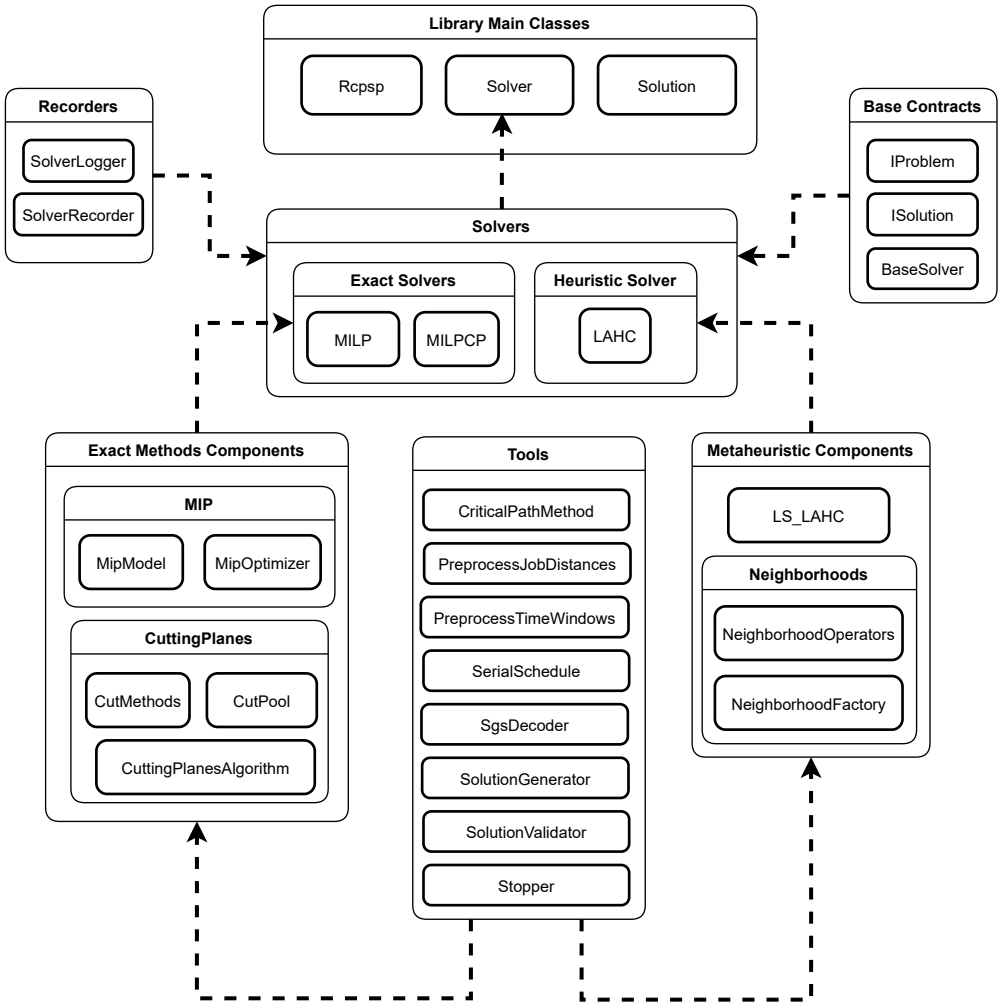
Considerando que o objetivo deste trabalho consistiu na integração de novos métodos de resolução a uma plataforma já existente, a metodologia adotada no âmbito do desenvolvimento concentrou-se na definição do ambiente de implementação, na utilização dos componentes estruturais da biblioteca de resolvedores e na adaptação das interfaces necessárias. Todas as etapas foram conduzidas de modo a preservar a compatibilidade com a infraestrutura previamente consolidada da plataforma.

3.1.1 Biblioteca de resolvedores *LaIC RCPSP Solvers*

Os resolvedores da ItPlatPSP são implementados em uma biblioteca desacoplada dos demais módulos da plataforma, denominada *LaIC RCPSP Solvers*. Essa biblioteca constitui o núcleo responsável pela implementação e execução dos métodos de resolução. A Figura 1 apresenta uma visão geral de sua arquitetura antes da integração dos métodos desenvolvidos neste trabalho.

De forma geral, a biblioteca é estruturada nas seguintes camadas: (i) contratos base e abstrações centrais, (ii) componentes específicos de métodos exatos e meta-heurísticos, (iii) ferramentas utilitárias, (iv) resolvedores em nível de orquestração e (v) mecanismos de registro e monitoramento da execução.

Figura 1 – Diagrama de componentes da biblioteca de resolvedores da ItPlatPSP antes das implementações deste trabalho.



Fonte: Elaborado pelo autor.

3.1.2 Ambiente de Prototipação

As etapas iniciais de prototipação e validação dos métodos foram conduzidas em ambiente *Jupyter Notebook*[1], em conjunto com recursos já presentes na biblioteca, conforme as necessidades de implementação e experimentação. Os componentes e camadas utilizados são apresentados a seguir.

Da camada de **Contratos Base**, foram empregadas as interfaces de Problema e Solução, bem como a classe abstrata que define o comportamento mínimo esperado para todos os resolvedores. A utilização desses elementos desde a fase de prototipação garantiu a padronização das implementações e a compatibilidade com a arquitetura geral da biblioteca.

[1] <https://jupyter.org/>

Da camada de **Ferramentas Utilitárias**, foram utilizados o módulo responsável pela geração de soluções iniciais, o validador de soluções e a classe de representação no formato baseado em sequência, que encapsula o procedimento de decodificação para o formato baseado em cronograma.

Por fim, da camada de **Componentes Meta-heurísticos**, foram empregados os módulos relacionados aos operadores de vizinhança.

3.1.3 Ambiente de Integração

O processo de integração dos novos métodos ao código-fonte da plataforma foi conduzido por meio de controle de versão utilizando a ferramenta *Git*[2], em conjunto com o repositório remoto da ItPlatPSP hospedado no *GitHub*[3].

As implementações foram realizadas, inicialmente, em um ramo de desenvolvimento isolado do código-fonte principal. A validação dos métodos integrados foi conduzida por meio de testes de integração já existentes na biblioteca de resolvedores, implementados com a suíte padrão de testes do Python e orquestrados com o *pytest*[4]. Esses testes avaliam a execução dos resolvedores sob diferentes combinações de parâmetros, considerando tanto cenários válidos quanto situações inválidas, como a inicialização com dados inconsistentes, além da verificação do comportamento esperado nos casos de sucesso.

Para cada novo resolvedor implementado neste trabalho, foram definidos testes seguindo o mesmo padrão estrutural dos já existentes, mantendo a consistência com os critérios já estabelecidos na biblioteca. Após a validação por meio dessa suíte de testes, os novos métodos foram incorporados ao ramo principal do repositório.

3.2 Ambiente de Experimentos

Esta seção descreve o ambiente computacional adotado, a configuração dos experimentos realizados e o conjunto de instâncias utilizado na avaliação dos métodos implementados. Também são apresentadas as métricas empregadas para a análise dos resultados.

3.2.1 Ambiente Computacional

Os experimentos foram executados em um computador com processador Intel Xeon E5-2650 v2 (2.60 GHz, 8 núcleos e 16 threads) e 16 GB de memória RAM DDR3. O sistema operacional utilizado foi o Debian 13 Trixie.

[2] <git-scm.com/>
[3] <github.com>
[4] <https://docs.pytest.org>

3.2.2 Configuração dos Experimentos

Os critérios de parada adotados para todos os métodos avaliados foram baseados no tempo máximo de execução e no número máximo de iterações consecutivas sem melhoria. O tempo limite foi fixado em 300 segundos, enquanto o limite de iterações sem melhoria foi estabelecido em 5.000.000. Optou-se por um valor elevado para este segundo critério a fim de priorizar o tempo de execução como principal mecanismo de encerramento das buscas.

Para cada instância e para cada método, foram realizadas 5 rodadas independentes de execução. Cada rodada foi iniciada a partir de uma solução inicial gerada aleatoriamente. Considerando a natureza estocástica da geração das soluções iniciais e de determinados procedimentos internos dos métodos, foi utilizado um *seed* fixo por rodada, de modo a garantir a reprodutibilidade dos experimentos.

3.2.3 Conjunto de Instâncias

O conjunto de instâncias selecionado é composto por amostras das bibliotecas das bibliotecas mencionadas na Seção 2.6.1, e abrangendo as variantes SMRCPSP, MMRCPSP e MMRCMPSP.

Na etapa de seleção das amostras, foram priorizadas instâncias com tempo de execução estimado entre 5 e 10 minutos e maior desvio em relação à melhor solução conhecida. Essas estimativas foram obtidas a partir dos resultados publicados por Araujo et al. (2020).

A composição do conjunto experimental, apresentada na Tabela 1, compreende 11 grupos com 10 instâncias cada, totalizando 110 instâncias avaliadas. Nessa tabela, as colunas indicam, respectivamente: o subconjunto de instâncias (“Grupo”), a quantidade de amostras (“n”), a variante do problema (“Variante”), conforme apresentado na Seção 1.1, e a biblioteca ou repositório de origem dos dados (“Origem”).

Tabela 1 – Resumo das instâncias selecionadas para os experimentos.

Grupo	n	Variante	Origem
j30	10	MMRCPSP	PSPLIB
j50	10	MMRCPSP	MMLIB
j60	10	SMRCPSP	PSPLIB
j90	10	SMRCPSP	PSPLIB
j100	10	MMRCPSP	MMLIB
j120	10	SMRCPSP	PSPLIB
jall50	10	MMRCPSP	MMLIB
jall100	10	MMRCPSP	MMLIB
A	10	MMRCMPSP	MISTA
B	10	MMRCMPSP	MISTA
X	10	MMRCMPSP	MISTA

Fonte: Elaborado pelo Autor.

3.2.4 Métricas de Avaliação

A forma mais convencional de avaliar a qualidade das soluções obtidas para os RCPSPs é por meio do *gap* (δ) em relação a Melhor Solução Conhecida *(Best Known Solution)* (BKS) (Demeulemeester e Herroelen; Asta et al.; Araujo, 2002, 2016, 2019). O *gap* é definido pela Equação 3.1:

$$\delta = \frac{\text{valor da solução encontrada} - BKS}{BKS} \tag{3.1}$$

Para a média do δ obtido na resolução de uma mesma instância ao longo das R rodadas, considera-se:

$$\bar{\delta}_i^{(m)} = \frac{1}{R} \sum_{r=1}^{R} \delta_{i,r}^{(m)} \tag{3.2}$$

onde $\delta_{i,r}^{(m)}$ representa o *gap* obtido pelo método m na resolução da instância i durante a rodada r.

Para a média do δ nos resultados obtidos em cada grupo g, define-se:

$$\bar{\delta}_g^{(m)} = \frac{1}{N_g \cdot R} \sum_{i \in N_g} \sum_{r=1}^{R} \delta_{i,r}^{(m)} \tag{3.3}$$

onde N_g representa o número de instâncias pertencentes ao grupo g.

Para a média do δ nos resultados obtidos em cada variante do problema v, define-se:

$$\bar{\delta}_v^{(m)} = \frac{1}{N_v \cdot R} \sum_{i \in N_v} \sum_{r=1}^{R} \delta_{i,r}^{(m)} \tag{3.4}$$

onde N_v corresponde ao número de instâncias pertencentes à variante v.

Por fim, a média global considerando todas as instâncias de todas as variantes é definida por:

$$\bar{\delta}^{(m)} = \frac{1}{N \cdot R} \sum_{i=1}^{N} \sum_{r=1}^{R} \delta_{i,r}^{(m)} \tag{3.5}$$

onde N representa o número total de instâncias do conjunto experimental.

Nos resultados apresentados no capítulo 5, o símbolo δ é utilizado como notação padrão para representar os valores médios calculados nos diferentes contextos (por instância, grupo, variante ou global), sendo a distinção explicitada sempre que necessário.

4 Desenvolvimento

As meta-heurísticas selecionadas neste trabalho foram escolhidas com o objetivo de ampliar e diversificar o conjunto de métodos de resolução disponíveis na ItPlatPSP. Até então, a plataforma dispunha apenas do LAHC como abordagem meta-heurística, na qual tanto a escolha dos operadores de vizinhança quanto a seleção das soluções vizinhas são realizadas de forma aleatória.

Embora abordagens estocásticas apresentem bom desempenho em diferentes contextos, a adoção exclusiva desse tipo de estratégia limita a análise comparativa entre paradigmas distintos de exploração do espaço de soluções da plataforma. Nesse sentido, optou-se pela incorporação de meta-heurísticas baseadas em VNS, as quais se caracterizam por uma exploração sistemática das estruturas de vizinhança. Diferente do LAHC, métodos derivados do VNS permitem definir explicitamente regras para a seleção e ordenação dos operadores de vizinhança, bem como políticas determinísticas para a transição entre diferentes estruturas de vizinhança ao longo do processo de busca.

Conforme Mladenović e Hansen (1997), o VNS é composto por três etapas principais: a perturbação da solução atual, etapa responsável pela diversificação da busca, cujo objetivo é escapar de ótimos locais por meio da aplicação de um movimento de vizinhança, conduzindo a exploração para uma nova região do espaço de soluções; a rotina de melhoria, fase de intensificação na qual as soluções vizinhas associadas a um operador específico são avaliadas de maneira sistemática, usualmente por meio de estratégias clássicas de busca local, como primeira melhoria ou melhor melhoria; e a troca de vizinhança, mecanismo que define qual estrutura será utilizada na iteração subsequente, com base no desempenho da solução gerada na etapa de melhoria.

Diferentes variações do VNS derivam do comportamento adotado em cada uma dessas três etapas fundamentais. Neste trabalho, tais configurações serviram de base para a implementação de métodos que abrangem desde abordagens clássicas até estratégias mais sofisticadas. As seções a seguir detalham a construção dos componentes comuns aos algoritmos, os métodos derivados do VNS selecionados e as técnicas adotadas para mitigar o custo computacional da exploração sistemática de vizinhanças.

4.1 Operadores de vizinhança

Os operadores de vizinhança utilizados neste trabalho são os já implementados na plataforma. Ao todo, são empregados dez operadores, sendo que seis aplicam movimentos na sequência de alocação das atividades π, e quatro atuam sobre o conjunto de modos de

execução $\mathcal{M}$. A descrição geral de cada operador é dada a seguir.

O operador ***Squeeze Project on Extreme* (SPE)** agrupa atividades de um mesmo projeto p em torno de uma atividade de referência selecionada a partir dos índices da sequência π. O movimento preserva a ordem relativa interna das atividades do projeto, alterando apenas sua posição global na sequência. A Figura 2 ilustra a aplicação do operador considerando como referência a posição π_4, correspondente à atividade 14.

Figura 2 – Exemplo de vizinhança gerada pelo operador SPE, agrupando as atividades de um projeto em torno de uma atividade de referência.

s'=SPE(s,4)

s	**2**	**3**	**5**	**8**	**14**	**10**	**7**	**9**	**12**
s'	**3**	**8**	**2**	**5**	**14**	**10**	**9**	**7**	**12**

Fonte: Araujo (2019)

O operador ***Swap and Compact Two Projects* (SCTP)** troca a posição relativa de dois projetos na sequência de atividades. Os parâmetros de entrada são dois projetos p_a e p_b. As atividades dos projetos são agrupadas, e então é feita a troca de suas posições relativas na sequência de alocação. A Figura 3 ilustra o processo, onde os parâmetros de entrada são a solução atual s e os projetos p_2 e p_3.

Figura 3 – Exemplo de vizinhança gerada pelo operador SCTP, mostrando a troca de posição relativa entre dois projetos na sequência de alocação.

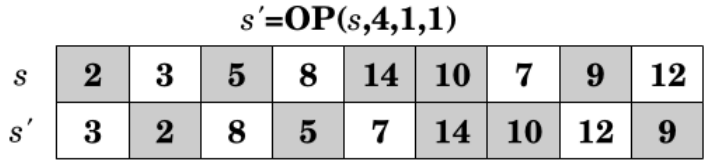
s'= SCTP (s,2,3)

s	**5**	**9**	**3**	**6**	**13**	**2**	**1**	**7**	**25**
s'	**5**	**9**	**6**	**7**	**3**	**13**	**1**	**2**	**25**

Fonte: Araujo (2019)

O operador ***Offset Project* (OP)** desloca todas as atividades de um projeto em k posições. O operador atua sobre um único projeto p, realizando o deslocamento de n atividades à esquerda ($k < 0$) ou à direita ($k > 0$). Na Figura 4, o movimento é ilustrado, onde todas as atividades de p_4 são deslocadas uma posição à direita.

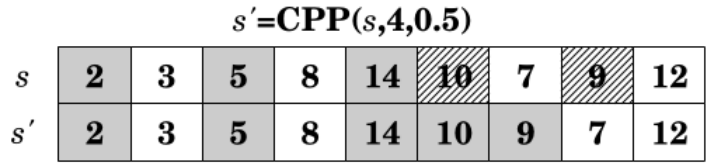
Figura 4 – Exemplo de vizinhança gerada pelo operador OP, deslocando todas as atividades de um projeto em uma posição à direita.



Fonte: Araujo (2019)

O operador ***Compact Project on Percentage* (CPP)** compacta parcialmente as atividades de um projeto em direção ao início da sequência, com base em um fator percentual. O movimento ocorre sobre um projeto p, deslocando para a esquerda uma parcela de suas atividades definida por $perc \in [0, 1]$. A Figura 5 ilustra o movimento do operador recebendo como entrada o projeto p_4 com $perc = 0.5$.

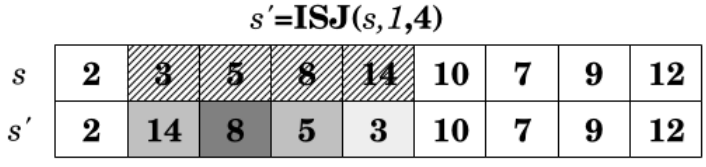
Figura 5 – Exemplo de vizinhança gerada pelo operador CPP, compactando parcialmente as atividades de um projeto com base em um fator percentual.



Fonte: Araujo (2019)

O operador ***Invert Sequence of Jobs* (ISJ)** inverte a ordem de uma subsequência interna de atividades. Essa subsequência é definida pelos parâmetros i, que indica a posição inicial, e k, que determina seu tamanho. A Figura 6 apresenta o movimento, invertendo uma subsequência de tamanho 4 a partir da posição π_1.

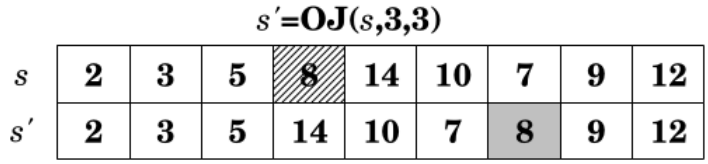
Figura 6 – Exemplo de vizinhança gerada pelo operador ISJ, invertendo a ordem de uma subsequência interna de atividades.



Fonte: Araujo (2019)

O operador ***Offset Job* (OJ)** desloca uma única atividade em k posições na sequência. O operador move uma atividade j à esquerda ($k < 0$) ou à direita ($k > 0$). A Figura 7 ilustra esse movimento aplicado à atividade π_3 na sequência, sendo deslocada em 3 posições.

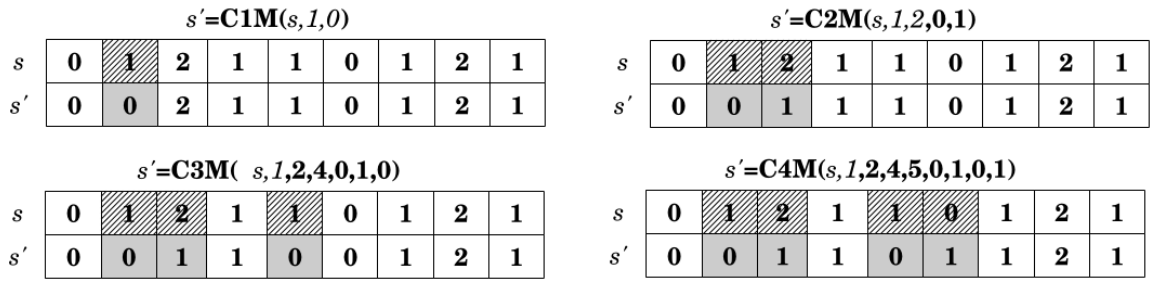
Figura 7 – Exemplo de vizinhança gerada pelo operador OJ, deslocando uma única atividade para outra posição na sequência.



Fonte: Araujo (2019)

Os operadores ***Change "n" Modes* (C1M a C4M)** alteram o modo de execução de uma a quatro atividades, priorizando as que estão conectadas diretamente no grafo de precedência quando o movimento é realizado para três ou quatro atividades. A Figura 8 ilustra as vizinhanças geradas pelos movimentos desse operador, onde o parâmetro de entrada que varia para cada um é a quantidade de pares (j, m) indicando a atividade e o modo a ser aplicado.

Figura 8 – Exemplo de vizinhanças geradas pelos operadores C1M a C4M, alterando o modo de execução de uma a quatro atividades conforme os pares (j, m) informados.



Fonte: Araujo (2019)

Por fim, é importante considerar que na plataforma, os operadores aplicam apenas o movimento; soluções infactíveis são tratadas pelo método que os utiliza, o qual decide se aplica um novo movimento ou se troca de operador.

4.1.1 Geração de soluções vizinhas sob demanda

Dado um operador de vizinhança N_k e uma solução atual s, a busca local integrada ao VNS pressupõe a avaliação das soluções pertencentes ao conjunto:

$$N_k(s) = \{s_i \mid s_i \text{ é obtida pela aplicação de } N_k \text{ sobre } s\}.$$

Idealmente, o conjunto $N_k(s)$ deve ser explorado priorizando os vizinhos mais próximos de s, normalmente os gerados por movimentos de menor intensidade.

Contudo, a enumeração completa de $N_k(s)$ pode ser computacionalmente inviável para a maioria dos RCPSP. Isso ocorre devido à explosão combinatória do espaço de busca, como em operadores que atuam sobre os modos de execução, podendo tornar inviável o simples armazenamento dessas soluções em memória.

Para contornar essa limitação, foram adotadas duas estratégias neste trabalho. A primeira consiste em abrir mão da exploração exaustiva por meio da seleção aleatória de vizinhos. A segunda, mais estruturada, baseia-se na geração de soluções sob demanda utilizando o conceito de avaliação tardia (*lazy evaluation*).

Essa técnica foi implementada por meio de geradores[1] da linguagem Python. Em vez de instanciar o conjunto completo de vizinhos, o gerador fornece os parâmetros de entrada necessários para o operador N_k apenas quando solicitados, mantendo a ordem de exploração definida nas regras de enumeração. O Algoritmo 1 detalha essa estratégia aplicada à busca local.

Algorithm 1 Exemplo de uma Busca local com geração de vizinhanças sob demanda

Entrada: problema p, solução corrente s, operador de vizinhança N_k

1: $s^* \leftarrow s$
2: $G_{sk} \leftarrow G(p, s, N_k)$
3: **enquanto** não atingir critério de parada **faça**
4: $\quad\alpha \leftarrow$ próximo conjunto de parâmetros de G_{sk}
5: $\quad$**se** $\alpha =$ NULL **então**
6: $\quad\quad$**break** {Fim da vizinhança}
7: $\quad$**fim se**
8: $\quad s' \leftarrow N_k(s, \alpha)$
9: $\quad$**se** $f(s') < f(s^*)$ **então**
10: $\quad\quad s^* \leftarrow s'$
11: $\quad$**fim se**
12: **fim enquanto**
13: **retorna** s^*

Ainda assim, o uso da exploração sistemática deve contar com critérios de parada adequados para garantir tempos de execução aceitáveis, como limite de tempo global ou

[1] <wiki.python.org/moin/Generators>

número máximo de iterações consecutivas sem melhoria.

4.2 Procedimento de perturbação

O procedimento de perturbação tem como objetivo explorar diferentes regiões no espaço de solução durante a busca, favorecendo o escape de ótimos locais. Conforme apresentado no Algoritmo 2, o procedimento recebe como entradas uma solução corrente s e o índice do operador de vizinhança k, a partir dos quais gera solução vizinha $s' \in N_k(s)$ mediante a aplicação de um movimento aleatório.

Algorithm 2 Procedimento de perturbação

Entrada: Solução atual s, índice da vizinhança k, operadores de vizinhança N
1: Selecionar aleatoriamente $s' \in N_k(s)$
2: **retorna** s'

4.3 Rotina de melhoria (busca local)

As rotinas de busca local atuam na intensificação da busca, explorando sistematicamente vizinhanças para refinar a solução obtida após a etapa de perturbação.

No contexto dos RCPSP considerados neste trabalho, a escolha da estratégia de exploração é determinante em razão da dimensão do espaço de busca desses problemas. Optou-se, portanto, pela estratégia de primeira melhoria (*first improvement*), que interrompe a busca ao encontrar o primeiro vizinho com valor melhor do que à solução atual. Essa abordagem reduz significativamente o tempo de processamento por iteração, permitindo a exploração de um número maior de regiões do espaço de soluções dentro dos limites impostos pelos critério de parada (tempo e número de iterações).

O Algoritmo 3 detalha este procedimento. A partir de uma solução atual s e uma vizinhança N_k, os vizinhos são avaliados sequencialmente até que ocorra uma melhora.

Algorithm 3 Busca local com estratégia de primeira melhoria

Entrada: Solução inicial s, estrutura de vizinhança N_k
1: $N_k(s) \leftarrow \{s'_1, s'_2, \ldots, s'_n\}$
2: $i \leftarrow 0$
3: **repita**
4: $\quad i \leftarrow i + 1$
5: $\quad$ **se** $f(s'_i) < f(s)$ **então**
6: $\quad\quad s \leftarrow s'_i$
7: $\quad\quad$ **break**
8: $\quad$ **fim se**
9: **até** $i = n$
10: **retorna** s

4.4 Troca de vizinhança

As políticas de troca de vizinhança determinam qual operador $N_k \in N$ será utilizado na iteração subsequente. A decisão considera o valor da solução candidata s' em relação a solução atual s, definindo qual será o operador utilizado no próximo ciclo. As três estratégias implementadas neste trabalho são descritas a seguir.

A troca de vizinhança sequencial (*Sequential Neighborhood Change*), detalhada no Algoritmo 4, retorna ao primeiro operador sempre que uma melhoria é identificada. Caso contrário, o índice k é incrementado para que a exploração utilize próximo operador em N.

Algorithm 4 Troca de vizinhança sequencial

Entrada: Solução atual s, solução candidata s', índice da vizinhança k
1: **se** $f(s') < f(s)$ **então**
2: $\quad s \leftarrow s'$
3: $\quad k \leftarrow 1$
4: **senão**
5: $\quad k \leftarrow k + 1$
6: **fim se**
7: **retorna** k

A troca de vizinhança cíclica (*Cyclic Neighborhood Change*), apresentada no Algoritmo 5, sempre aplica a alternância entre os operadores independente do sucesso da busca. O índice k é incrementado a cada iteração, enquanto a solução atual s é atualizada apenas se $f(s') < f(s)$.

Algorithm 5 Troca de vizinhança cíclica

Entrada: Solução atual s, solução candidata s', índice da vizinhança k
1: $k \leftarrow k + 1$
2: **se** $f(s') < f(s)$ **então**
3: $\quad s \leftarrow s'$
4: **fim se**
5: **retorna** k

Por fim, a troca de vizinhança *Pipe Neighborhood Change* (Algoritmo 6) prioriza a intensificação no operador corrente, mantendo o índice k inalterado enquanto houver melhoria da solução atual. A transição para o próximo operador de vizinhança é feita apenas quando não ocorre melhoria da solução atual.

Algorithm 6 Troca de vizinhança do tipo *pipe*

Entrada: Solução atual s, solução candidata s', índice da vizinhança k
1: **se** $f(s') < f(s)$ **então**
2: $\quad s \leftarrow s'$
3: **senão**
4: $\quad k \leftarrow k + 1$
5: **fim se**
6: **retorna** k

4.5 Descida em vizinhança variável

A Descida em Vizinhança Variável (*Variable Neighborhood Descent*) (VND) Mladenović e Hansen (1997) consiste em um método de exploração sistemática de múltiplas vizinhanças, sendo amplamente empregada como mecanismo de intensificação em metaheurísticas baseadas em VNS.

O método foi implementado conforme sua definição clássica (Algoritmo 7), na qual a busca é organizada a partir de um conjunto ordenado de operadores de vizinhança, os quais são aplicados sequencialmente ao procedimento de busca local até que seja atingido um ótimo local em relação a todas as estruturas consideradas.

Algorithm 7 Algoritmo clássico do VND

Entrada: Solução inicial s_0, operadores de vizinhança N
1: $s \leftarrow s_0$ {Solução atual}
2: $r \leftarrow |N|$
3: $k \leftarrow 1$ {Índice da estrutura de vizinhança}
4: **enquanto** $k \leq r$ **faça**
5: $\quad s' \leftarrow \text{PrimeiraMelhoria}(s, N_k)$
6: $\quad s' \leftarrow \text{TrocaVizinhança}(s, s', k)$
7: **fim enquanto**
8: **retorna** s

Uma característica relevante da implementação realizada é a sua flexibilidade, permitindo que a política de troca de vizinhança (conforme descrito na Seção 4.4) seja definida via parâmetros de inicialização, adaptando o comportamento do algoritmo conforme desejado.

O VND integrado à ItPlatPSP neste trabalho é utilizado tanto como resolvedor independente quanto como componente de intensificação em uma variante específica do VNS (detalhada na próxima Seção).

4.6 Variantes do VNS implementadas

Esta seção apresenta as variantes do VNS selecionadas e implementadas neste trabalho. Para cada método, são destacados os mecanismos de perturbação, busca local e troca de operador de vizinhança, evidenciando como cada variante trata o equilíbrio entre intensificação e diversificação no processo de busca.

4.6.1 *Basic* VNS

A Busca em Vizinhança Variável Básica (*Basic Variable Neighborhood Search*) (BVNS) (Hansen et al., 2017) corresponde à versão clássica da meta-heurística. O método segue o fluxo iterativo padrão do VNS, composto pelas etapas de perturbação, busca local e troca de vizinhança.

O Algoritmo 8 detalha a implementação deste método na plataforma. O procedimento inicia com a perturbação da solução atual para gerar uma solução candidata, a qual é submetida à rotina de primeira melhoria. Na sequência, o mecanismo de troca de vizinhança (conforme detalhado na Seção 4.4) é aplicado para decidir sobre a aceitação da nova solução e a atualização do índice k.

É importante notar que o algoritmo possui um laço externo controlado pelo critério de parada, reiniciando a busca pela primeira vizinhança ($k = 1$) sempre que todo o conjunto de operadores é percorrido (quando $k = r$).

Algorithm 8 *Basic Variable Neighborhood Search (BVNS)*

Entrada: Solução inicial s_0, operadores de vizinhança N

$s \leftarrow s_0$ {Solução atual}
$r \leftarrow |N|$
enquanto critério de parada não satisfeito **faça**
  $k \leftarrow 1$ {Índice da estrutura de vizinhança}
  enquanto $k \leq r$ **faça**
    $s' \leftarrow$ Perturbação(s, N_k)
    $s'' \leftarrow$ PrimeiraMelhoria(s', N_k)
    $s, k \leftarrow$ TrocaVizinhança(s, s'', k)
  fim enquanto
fim enquanto
retorna s

As variantes apresentadas nas seções seguintes derivam dessa estrutura fundamental, que também foi a base para a implementação concreta dos métodos integradas à plataforma.

4.6.2 *General* VNS

A Busca em Vizinhança Variável Geral (*General Variable Neighborhood Search*) (GVNS) (Hansen et al., 2017) utiliza o VND (Algoritmo 7) como rotina de intensificação, substituindo o método de busca local de primeira melhoria. A implementação desse método foi realizada conforme apresentado no Algoritmo 9.

Nessa estrutura, cada solução gerada na etapa de perturbação é submetida ao VND antes de sua eventual aceitação, o que promove uma intensificação mais profunda na região explorada. Em contrapartida, essa estratégia implica maior esforço computacional por iteração em comparação às demais variantes.

Algorithm 9 *General Variable Neighborhood Search (GVNS)*

Entrada: Solução inicial s_0, operadores de vizinhança N
1: $s \leftarrow s_0$ {Solução atual}
2: $r \leftarrow |N|$
3: **enquanto** critério de parada não satisfeito **faça**
4: $k \leftarrow 1$ {Índice da estrutura de vizinhança}
5: **enquanto** $k \leq r$ **faça**
6: $s' \leftarrow$ Perturbação(s, N_k)
7: $s'' \leftarrow$ VND(s', k, N)
8: $s, k \leftarrow$ TrocaVizinhança(s, s'', k)
9: **fim enquanto**
10: **fim enquanto**
11: **retorna** s

Uma particularidade em relação a inicialização do VND implementado neste trabalho é que, quando utilizado como componente interno do GVNS, o índice k inicial não é definido dentro do VND. Em vez disso, o procedimento recebe o índice do operador atual proveniente da etapa de perturbação que ocorre externamente.

4.6.3 *Reduced* VNS

A Busca em Vizinhança Variável Reduzida (*Reduced Variable Neighborhood Search*) (RVNS) (Hansen et al., 2017) é a única variante que não aplica a rotina de intensificação por meio de uma busca local. O método opera exclusivamente com as fases de perturbação e troca de operador de vizinhança, como mostra o Algoritmo 10.

Algorithm 10 *Reduced Variable Neighborhood Search (RVNS)*

Entrada: Solução inicial s_0, operadores de vizinhança N
1: $s \leftarrow s_0$ {Solução atual}
2: $r \leftarrow |N|$
3: **enquanto** critério de parada não satisfeito **faça**
4: $k \leftarrow 1$ {Índice da estrutura de vizinhança}
5: **enquanto** $k \leq r$ **faça**
6: $s' \leftarrow$ Perturbação(s, N_k)
7: $s, k \leftarrow$ TrocaVizinhança(s, s'', k)
8: **fim enquanto**
9: **fim enquanto**
10: **retorna** s

Devido a essa simplificação, o custo por iteração é significativamente menor, concentrando o esforço algorítmico na diversificação. Essa abordagem é particularmente adequada para cenários em que uma exploração ampla do espaço de busca é desejável ou quando tempos mais curtos de processamento são prioridade.

4.6.4 *Skewed* VNS

A Busca em Vizinhança Variável Inclinada (*Skewed Variable Neighborhood Search*) (SVNS) (Hansen et al., 2017) adota um viés no critério de aceitação como estratégia para escapar de ótimos locais.

No passo de troca de operador de vizinhança, detalhado no Algoritmo 12 e integrado ao fluxo principal no Algoritmo 11, soluções candidatas s'' de qualidade inferior podem ser aceitas caso estejam suficientemente distantes da solução s. Essa aceitação é ponderada por um parâmetro de penalização α e uma função de distância $\rho(s, s'')$, favorecendo a exploração de regiões mais distantes do espaço de busca.

Algorithm 11 *Skewed Variable Neighborhood Search (SVNS)*

Entrada: Solução inicial s_0, operadores de vizinhança N, fator de aceitação α

$s \leftarrow s_0$ {Solução atual}
$s^* \leftarrow s$ {Melhor solução encontrada}
$r \leftarrow |N|$
enquanto critério de parada não satisfeito **faça**
  $k \leftarrow 1$ {Índice da estrutura de vizinhança}
  enquanto $k \leq r$ **faça**
    $s' \leftarrow$ Perturbação(s, N_k)
    $s'' \leftarrow$ PrimeiraMelhoria(s', N_k)
    se $f(s'') < f(s^*)$ **então**
      $s^* \leftarrow s''$
    fim se
    TrocadeVizinhançaEnviesada(s, s'', k, α)
  fim enquanto
fim enquanto
retorna s^*

É importante ressaltar que, embora o Algoritmo 12 ilustre a troca enviesada utilizando o método sequencial, a implementação na plataforma foi adaptada para que a troca de operador seja feita conforme a política de troca definida nos parâmetros inicialização do método.

Algorithm 12 Troca de Vizinhança Enviesada

Entrada: Solução atual s, solução candidata s'', índice da vizinhança k, parâmetro de penalização α

se $f(s'') < f(s) + \alpha \cdot \rho(s, s'')$ **então**
  $s \leftarrow s''$
  $k \leftarrow 1$
senão
  $k \leftarrow k + 1$
fim se
retorna s, k

4.6.5 *Smart* VNS

O Busca em Vizinhança Variável Inteligente (*Smart Variable Neighborhood Search*) (Smart VNS), proposto por Souza et al. (2010), é uma variante adaptativa que estende o mecanismo de exploração ao controlar o número de perturbações consecutivas aplicadas ao mesmo operador de vizinhança (Algoritmo 13).

Inspirada nos princípios da Busca Local Iterada Inteligente (*Smart Iterated Local Search*) (SILS), esta abordagem permite ajustar a intensidade da diversificação dinamicamente antes de avançar para a etapa de melhoria.

Algorithm 13 *Smart Variable Neighborhood Search (Smart VNS)*

Entrada: Solução inicial s_0, operadores de vizinhança N, número máximo de perturbações $p_{\max}$

$s \leftarrow s_0$ {Solução atual}
$r \leftarrow |N|$
enquanto critério de parada não satisfeito **faça**
  $k \leftarrow 1$ {Vizinhança corrente}
  $p \leftarrow 1$ {Número de perturbações na mesma vizinhança}
  enquanto $k \leq r$ **faça**
    $j \leftarrow 1$
    enquanto $j \leq p$ **faça**
      $s' \leftarrow$ Perturbação(s, N_k) {Executa p perturbações}
    fim enquanto
    $s'' \leftarrow$ PrimeiraMelhoria(s', N_k)
    se $f(s'') < f(s)$ **então**
      $s \leftarrow s''$
      $p \leftarrow 1$
      $s, k \leftarrow$ TrocaVizinhança(s, s'', k)
    senão
      se $p \geq p_{\max}$ **então**
        $p \leftarrow 1$
        $s, k \leftarrow$ TrocaVizinhança(s, s'', k)
      senão
        $p \leftarrow p + 1$
      fim se
    fim se
  fim enquanto
fim enquanto
retorna s

Diferente das versões anteriores, o Smart VNS utiliza um contador p para gerenciar múltiplas perturbações sucessivas. Caso uma melhoria não seja encontrada, o algoritmo intensifica a diversificação naquela mesma vizinhança incrementando p até um limite $p_{\max}$, postergando a transição para o próximo operador até que o potencial de exploração da vizinhança atual seja exaurido.

5 Resultados

Este capítulo apresenta os resultados obtidos a partir da avaliação dos métodos implementados e do processo de integração à plataforma. Inicialmente, são descritos os experimentos computacionais e as análises decorrentes. Em seguida, são apresentados os resultados relacionados à integração dos resolvedores à biblioteca de resolvedores e, por fim, à sua incorporação à plataforma por meio da interface web.

5.1 Experimentos computacionais

Experimentos computacionais foram conduzidos com o objetivo de analisar a viabilidade dos métodos baseados em VNS na resolução das variantes dos RCPSP suportadas pela ItPlatPSP, bem como comparar o desempenho das abordagens implementadas com a versão do LAHC já existente na plataforma.

5.1.1 Calibragem de parâmetros

Conforme descrito no Capítulo 4 os métodos baseados em VND/VNS podem ser parametrizados em relação a estratégia de troca de vizinhanças, o critério de aceitação e na forma explorar os vizinhos de um operador (de forma sistemática ou aleatória). Além disso, os métodos SVNS e Smart VNS ainda possuem parâmetros específicos, como o fator de aceitação de soluções piores, no caso do SVNS, e o número máximo de perturbações consecutivas, no caso do Smart VNS.

Para avaliar a combinação de parâmetros mais promissora para cada método, um cenário de experimentos reduzido foi executado para obter estimativas do desempenho médio de cada método sob diferentes configurações de entrada. Nesta etapa, três instâncias de cada grupo foram selecionadas, com tempo limite de execução fixado em 5 minutos. Foram executadas três rodadas para a resolução de cada instância, onde cada rodada partiu de uma solução inicial diferente. A relação de parâmetros testados em cada resolvedor é apresentada na Tabela 2.

Tabela 2 – Parâmetros possíveis para os resolvedores baseados em VND/VNS.

Parâmetro	**Identificador**	**Entradas válidas**	**Resolvedores**

Troca de vizinhança	SEQUENTIAL
	CYCLIC
	PIPE
		`sequential`
		`cyclic`
		`pipe`
			Todos

Aceitar soluções com valor igual	ACCEQ	`ON` / `OFF`	Todos
Seleção aleatória de vizinhos	NBRD	`ON` / `OFF`	Todos (exceto RVNS)
Fator de aceitação para soluções piores	Sn	`10%; 25%; 50%; 75%`	SVNS
Número máximo de perturbações consecutivas	Pn	`3, 5, 10, 15, 20, 30`	Smart VNS

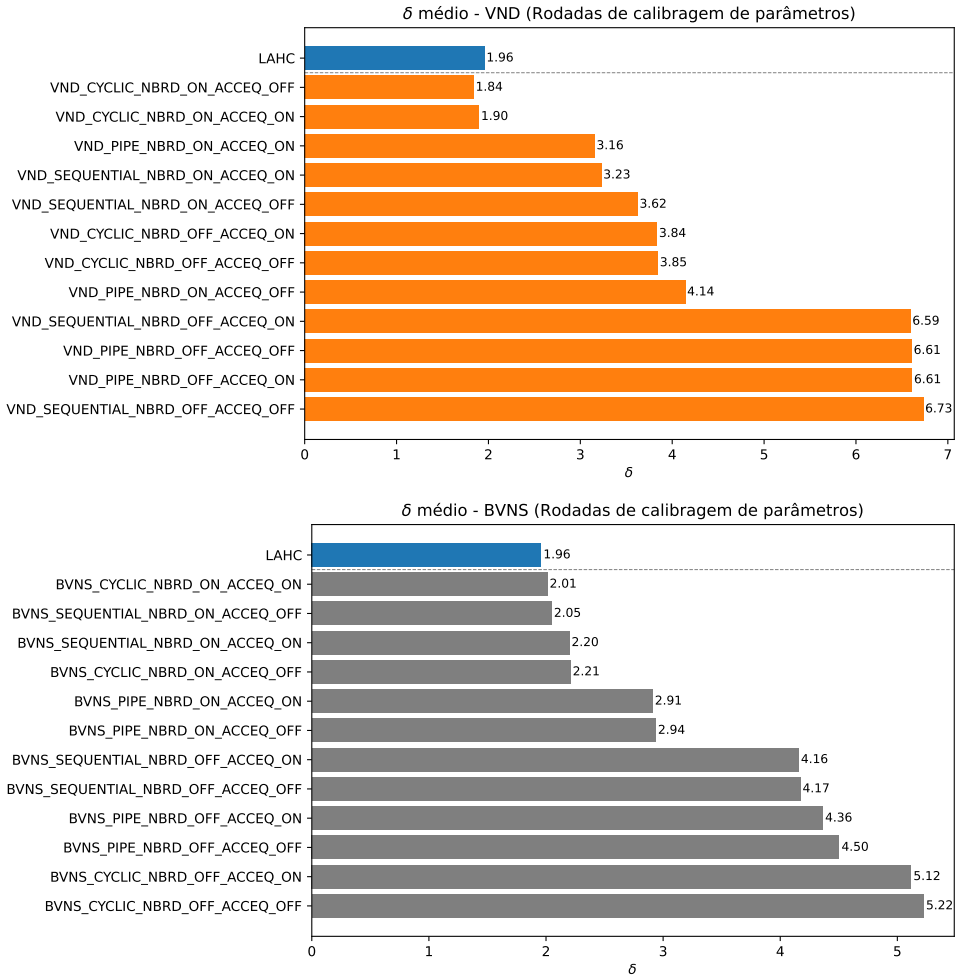
O LAHC também foi executado nesta etapa para servir como referência base de comparação com o desempenho dos demais métodos. O único parâmetro ajustável no LAHC é o tamanho do histórico de soluções melhoradas, que, neste experimento e nos demais foi fixado com tamanho 20.

As Figuras 9, 10 e 11 mostram o desempenho médio das três rodadas para cada método sob as diferentes configurações. O LAHC foi incluído no topo de cada gráfico, e os demais estão ordenados pelo valor médio do *gap* (δ) das execuções. Dado que os métodos SVNS e Smart VNS possuem um número maior de combinações possíveis, os registros no gráfico foram truncados para exibir os 25 melhores. Os demais apresentam todas as combinações avaliadas.

Os resultados da avaliação de parâmetros mostraram que todos os métodos obtêm melhores resultados quando a exploração dos vizinhos de um operador é realizada de forma aleatória (parâmetro `NBRD_ON`), com diferença significativa quando comparados às mesmas configurações com esse parâmetro desabilitado.

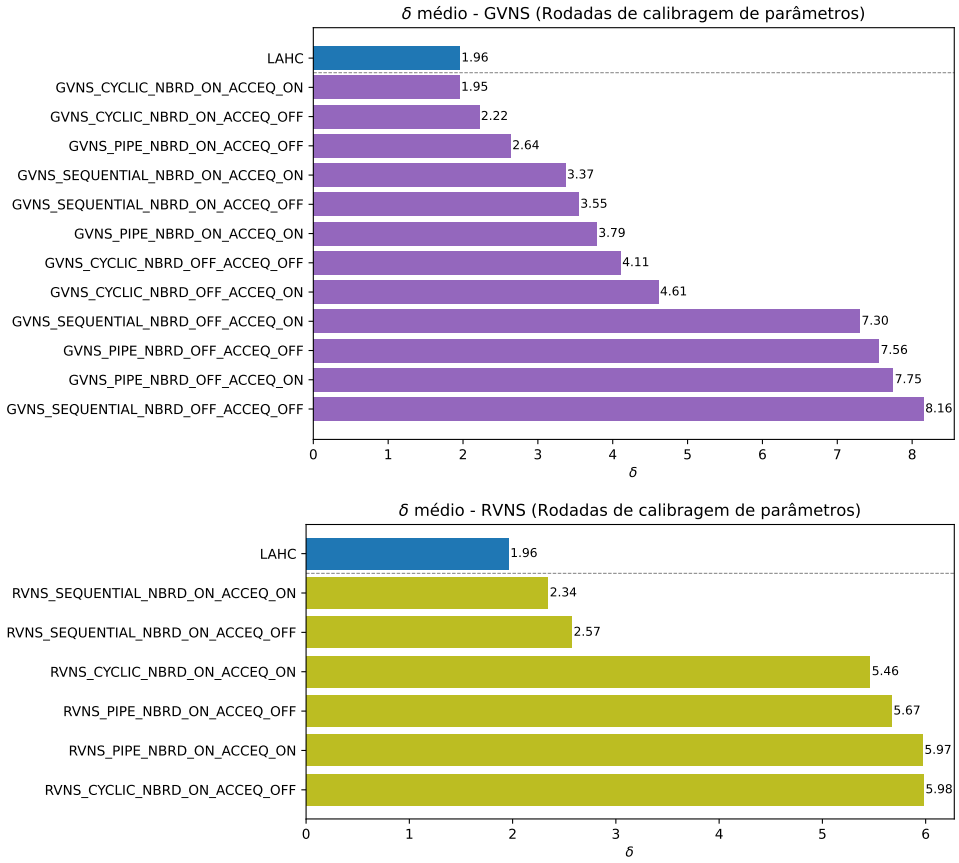
Quanto ao critério de aceitação de soluções com valor objetivo igual (`ACCEQ`), os resultados obtidos demonstraram que o impacto desse parâmetro pode variar dependendo do método e da estratégia de vizinhança selecionada. Observou-se, por exemplo, nos casos do GVNS e do VND, melhoria significativa nos resultados quando esse parâmetro está desabilitado, combinado com a troca de vizinhança do tipo *pipe*, em comparação às mesmas configurações com esse atributo habilitado.

Figura 9 – Valor médio de δ obtido nas rodadas de calibragem para os métodos de VND e BVNS.



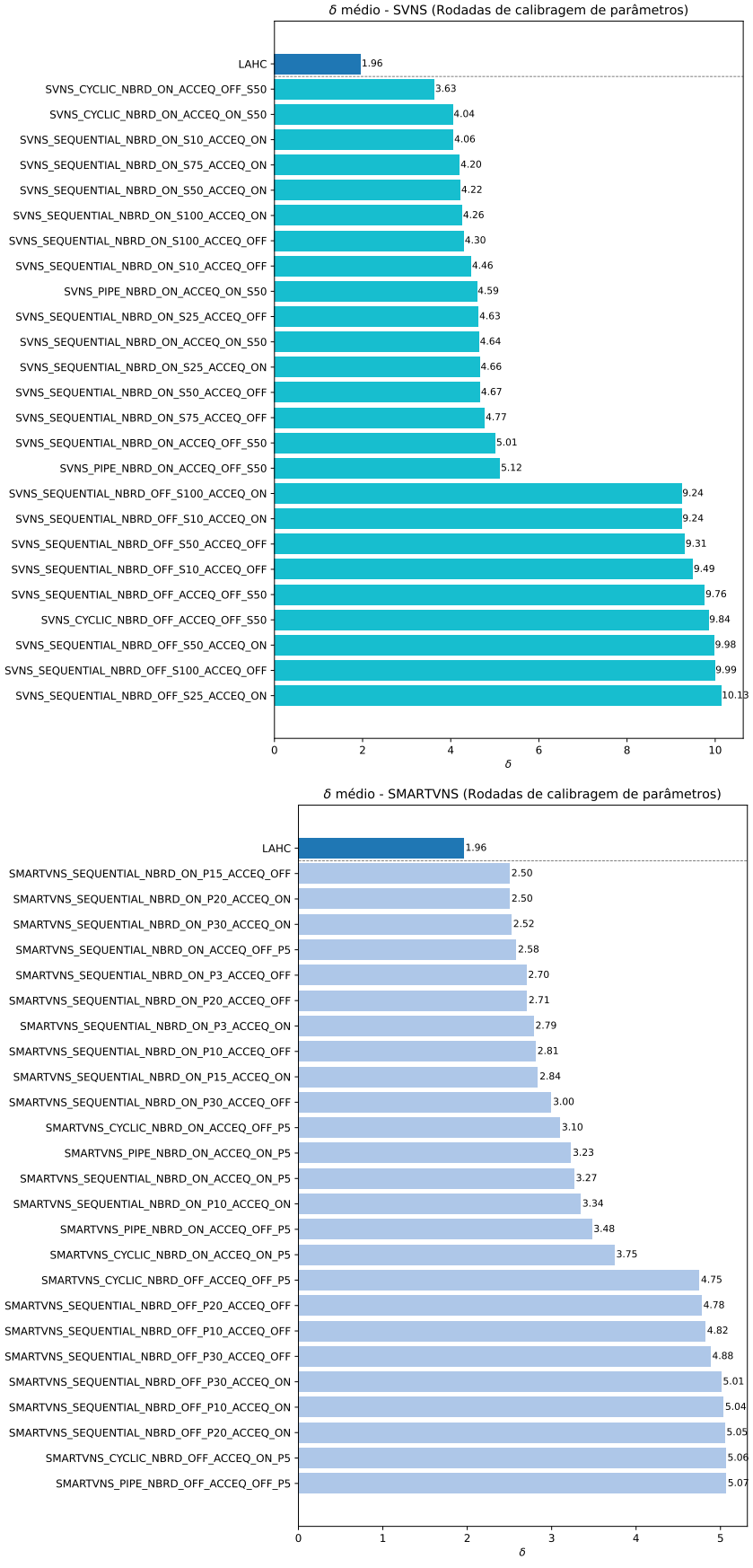
Fonte: Elaborado pelo autor.

Figura 10 – Valor médio de δ obtido nas rodadas de calibragem para os métodos de GVNS e RVNS.



Fonte: Elaborado pelo autor.

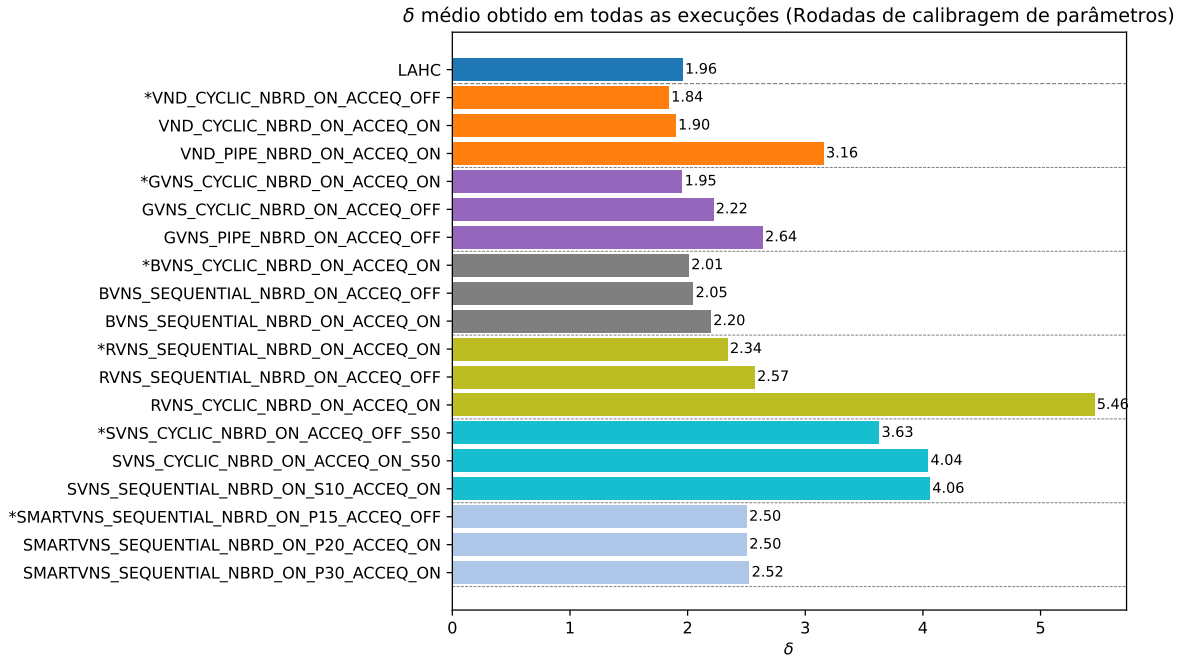
Figura 11 – Valor médio de δ obtido nas rodadas de calibragem para os métodos de SVNS e Smart VNS.



Fonte: Elaborado pelo autor.

A Figura 12 apresenta um resumo geral das combinações de parâmetros que produziram os melhores resultados para cada método nesta análise inicial. Com base nesses resultados, os parâmetros de entrada dos métodos para os experimentos computacionais foram fixados conforme apresentado na Tabela 3.

Figura 12 – Melhores resultados médios obtidos nas rodadas de calibragem dos métodos. (∗ Combinação de parâmetros com o melhor resultado para cada método.)



Fonte: Elaborado pelo autor.

Tabela 3 – Parâmetros selecionados para os experimentos com base nos melhores resultados obtidos na etapa de calibragem.

Parâmetro	**VND**	**BVNS**	**GVNS**	**RVNS**	**SVNS**	**SMART**
Troca de Vizinhança	`cyclic`	`cyclic`	`cyclic`	`sequential`	`cyclic`	`sequential`
Aceitar solução com valor igual	`OFF`	`ON`	`ON`	`ON`	`OFF`	`OFF`
Seleção aleatória de vizinhos	`ON`	`ON`	`ON`	-	`ON`	`ON`
Fator aceitação	-	-	-	-	50%	-
Máx. perturbações	-	-	-	-	-	15

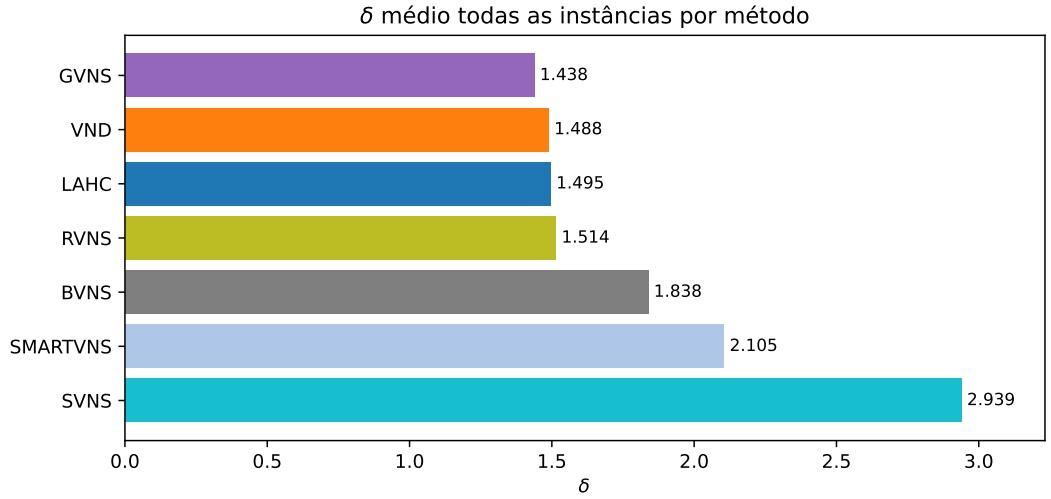
As subseções a seguir apresentam a análise e discussão dos resultados obtidos nos experimentos realizados sobre a totalidade do conjunto de amostras, considerando os resolvedores parametrizados conforme detalhado anteriormente.

5.1.2 Análise do desempenho médio dos novos métodos

A seguir são apresentados os resultados do desempenho médio dos métodos com base no *gap* (δ) em relação à melhor solução conhecida. As análises consideram os valores médios obtidos no conjunto completo de instâncias, bem como a segmentação por variante e por grupo, com o objetivo de identificar padrões de comportamento e diferenças de desempenho entre os métodos implementados.

A Figura 13 apresenta *gap* (δ) médio obtido por cada método considerando todo o conjunto de amostras. Os métodos com melhores resultados são o GVNS, VND, LAHC e RVNS, ambos bem próximos nos resultados médios. Por outro lado, BVNS, Smart VNS e SVNS apresentam os maiores desvios médios, com destaque para o SVNS, cuja média obtida é aproximadamente o dobro dos métodos com melhor desempenho.

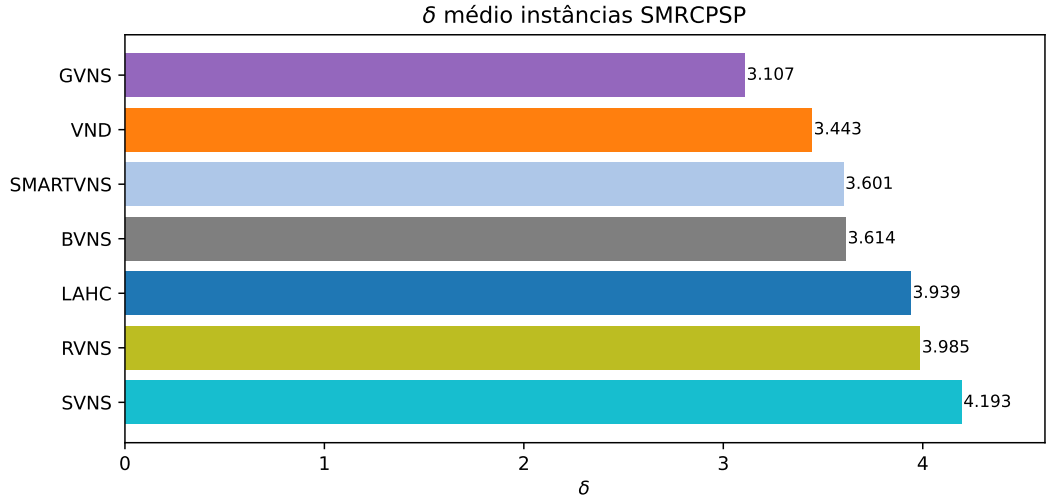
Figura 13 – δ médio dos métodos considerando todas as instâncias avaliadas.



Fonte: Elaborado pelo autor.

A Figura 14 mostra os resultados médios para as instâncias SMRCPSP. Os métodos GVNS e VND apresentam os melhores resultados, assim como observado na média global. Nessa variante, Smart VNS e BVNS apresentam desempenho intermediário, superando LAHC e RVNS, o que contrasta com a média geral. O SVNS novamente registra o maior δ entre os métodos avaliados.

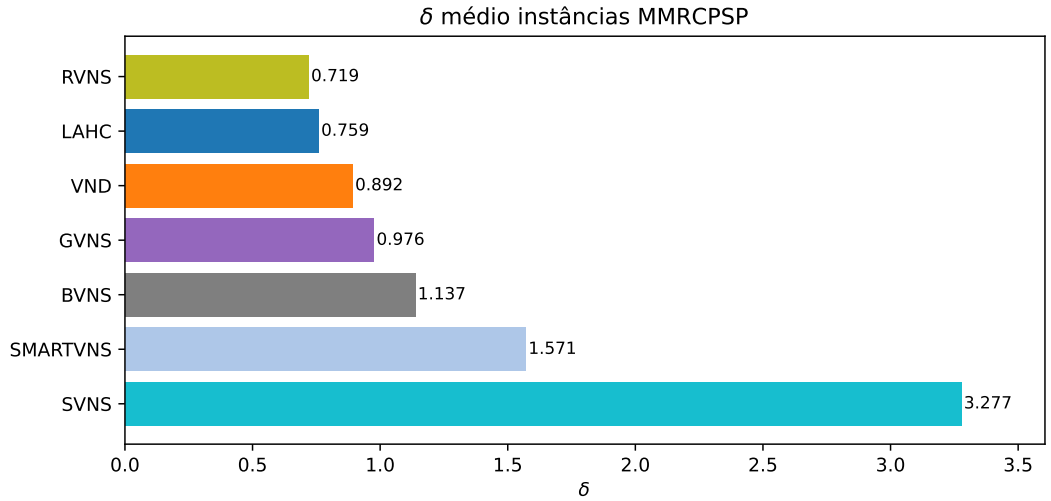
Figura 14 – δ médio dos métodos considerando apenas as instâncias SMRCPSP.



Fonte: Elaborado pelo autor.

A Figura 15 mostra resultados obtidos considerando apenas as instâncias MMRCPSP. Para este conjunto os métodos RVNS e LAHC são os com melhor desempenho, seguidos do VND e GVNS com δ inferior a 1. O BVNS tem desempenho intermediário, enquanto o Smart VNS e SVNS aparecem com os maiores desvios, com destaque para o SVNS onde a diferença para os demais é mais expressiva.

Figura 15 – δ médio dos métodos considerando apenas as instâncias MMRCPSP.

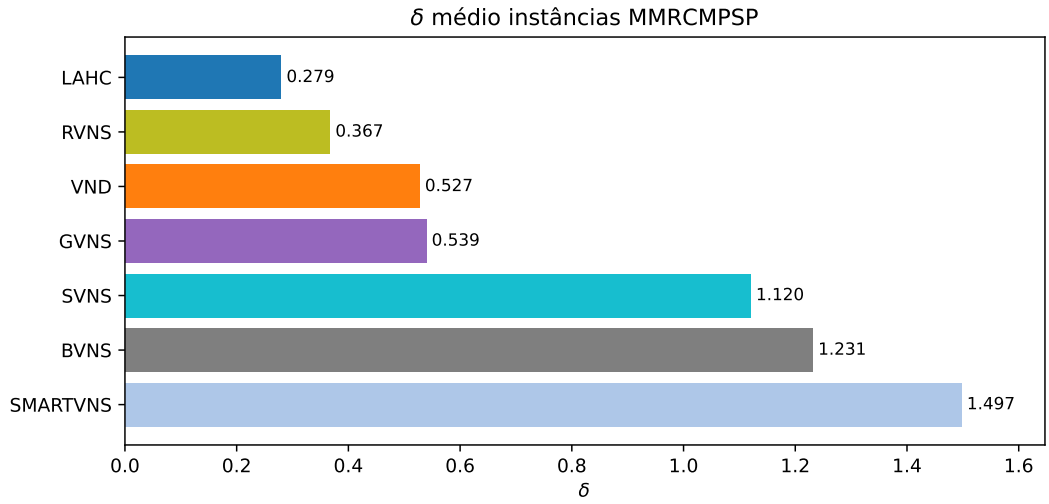


Fonte: Elaborado pelo autor.

Para as instâncias MMRCMPSP, as mais complexas do conjunto, os resultados indicam a superioridade do LAHC seguido pelo RVNS, conforme mostra a Figura 16.

VND e GVNS permanecem próximos entre si, porém com desempenho inferior aos dois primeiros. SVNS, BVNS e Smart VNS apresentam os maiores desvios médios. Observa-se, contudo, que o SVNS apresenta desempenho relativamente melhor nessa variante quando comparado às demais, superando o BVNS, embora permaneça com diferença significativa em relação aos métodos intermediários.

Figura 16 – δ médio dos métodos considerando apenas as instâncias MMRCMPSP.



Fonte: Elaborado pelo autor.

A Tabela 4 detalha o desempenho medido a partir do *gap* (δ) médio e desvio padrão ($\pm$) para cada grupo de instâncias presentes no conjunto de amostras.

De forma geral, LAHC e RVNS mantêm resultados próximos entre si, com desempenho mais consistente nas instâncias de maior complexidade, especialmente aquelas com múltiplos modos e múltiplos projetos. Esse comportamento é coerente com a estrutura semelhante dos métodos, ambos orientados a diversificação, com diferença apenas no uso dos operadores de vizinhança (troca sequencial no RVNS e aleatória no LAHC).

VND e GVNS também apresentam resultados próximos e demonstram comportamento mais equilibrado entre as diferentes variantes, mantendo desempenho competitivo tanto em instâncias mais simples quanto nas mais complexas.

Por fim, BVNS, SVNS e Smart VNS tendem a apresentar os maiores valores médios de δ na maioria dos grupos analisados. Ainda assim, em conjuntos específicos, como nas instâncias de menor porte do grupo J90 (SMRCPSP), esses métodos alcançam resultados relativamente competitivos.

De forma geral, a análise dos valores médios de δ, tanto no escopo global quanto por variante e por grupo, indica que métodos com estrutura mais simples e foco predominante

Tabela 4 – Média do valor de δ e respectivo desvio padrão para cada grupo de instâncias e método avaliado.

Grupo	LAHC		VND		BVNS		GVNS		RVNS		SVNS		SMARTVNS	
	δ	$\pm$	δ	$\pm$	δ	$\pm$	δ	$\pm$	δ	$\pm$	δ	$\pm$	δ	$\pm$
J60	0.649	2.492	0.679	2.489	0.906	3.015	**0.407**	1.793	1.441	4.574	0.637	2.495	0.632	2.495
J90	0.127	0.317	0.109	0.349	**0.091**	0.275	0.109	0.349	0.109	0.349	**0.091**	0.275	0.145	0.462
J120	11.042	13.350	9.543	11.288	9.844	11.861	**8.805**	10.463	10.406	12.828	11.852	13.545	10.027	11.751
J30	0.037	0.070	0.115	0.098	0.157	0.120	0.141	0.107	**0.020**	0.056	0.341	0.162	0.286	0.160
J50	0.231	0.261	0.257	0.263	0.266	0.257	0.272	0.260	**0.225**	0.264	0.497	0.236	0.363	0.347
J100	0.540	0.287	0.520	0.360	0.577	0.355	0.528	0.334	**0.508**	0.332	0.857	0.400	0.633	0.346
Jall50+	**0.811**	0.603	1.326	1.128	2.109	1.619	1.449	1.076	0.823	0.747	6.313	3.824	2.752	1.968
Jall100+	2.175	1.227	2.241	1.117	2.578	1.407	2.489	1.455	**2.021**	1.056	8.378	10.074	3.821	2.405
A	**0.207**	0.220	0.442	0.330	2.233	6.674	0.501	0.418	0.321	0.346	0.924	0.700	2.923	9.940
B	**0.313**	0.129	0.443	0.194	0.556	0.254	0.447	0.203	0.359	0.171	0.981	0.370	0.623	0.244
X	**0.317**	0.134	0.696	0.307	0.904	0.415	0.669	0.296	0.421	0.180	1.454	0.576	0.946	0.353
Global	1.495	5.109	1.488	4.364	1.838	4.992	**1.438**	4.035	1.514	5.000	2.939	6.486	2.105	5.509

Fonte: Elaborado pelo Autor.

Nota: Os valores destacados em negrito indicam o menor valor médio de δ em cada grupo.

na diversificação como LAHC e RVNS apresentam melhor desempenho em instâncias mais complexas, enquanto abordagens baseadas em intensificação estruturada, como VND e GVNS, se destacam por manter um desempenho estável em diferentes cenários.

Em termos médios, os métodos BVNS, SVNS e Smart VNS demonstraram maior sensibilidade às características estruturais das instâncias, apresentando variação mais acentuada de desempenho entre as variantes analisadas. Para o conjunto de amostras considerado nestes experimentos, os resultados sugerem que as estratégias adicionais de aceitação e de controle de perturbação empregadas no SVNS e no Smart VNS não resultaram em ganhos sistemáticos de qualidade de solução quando comparadas às demais abordagens.

5.1.3 Curvas de convergência

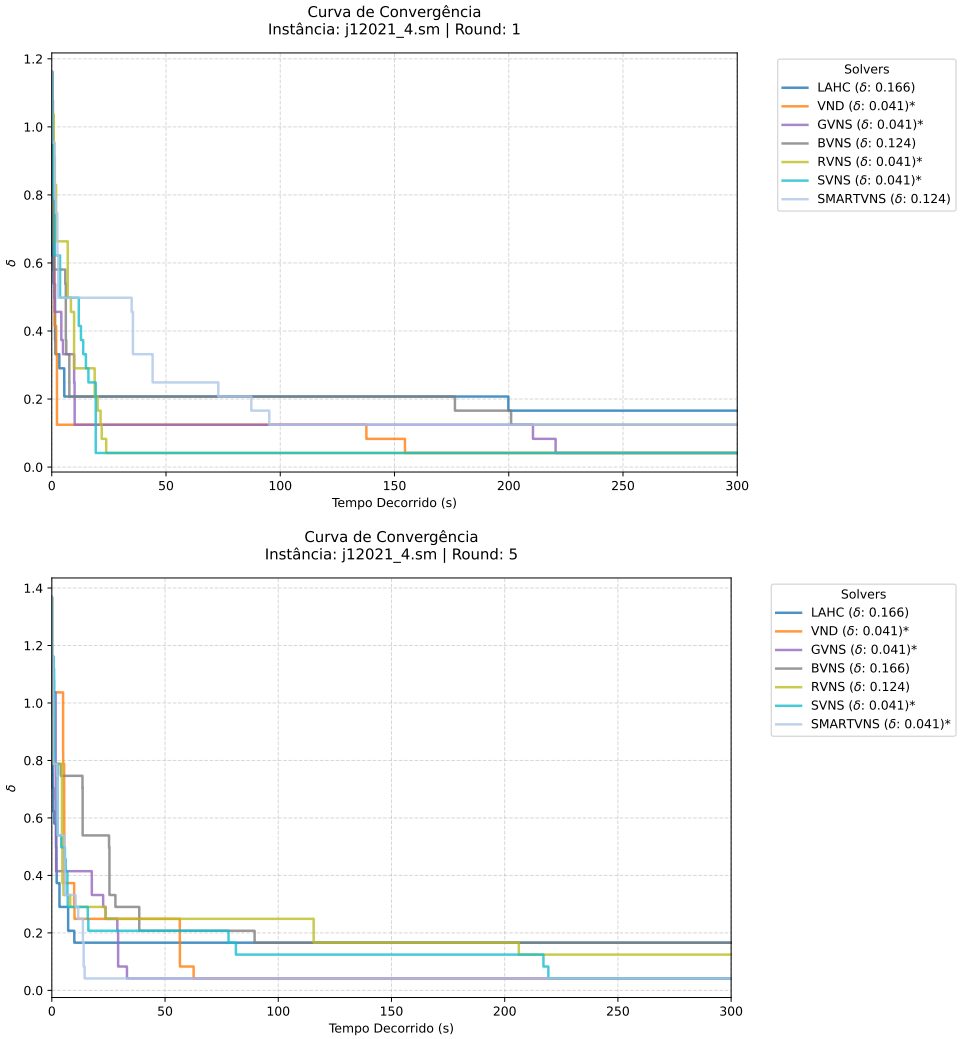
Esta seção apresenta a análise das curvas de convergência dos métodos de solução, considerando diferentes execuções para cada variante do problema.

Para essa análise, foram utilizados os logs de execução de uma instância de cada variante: j12021_4.sm (SMRCPSP), J10038_4.mm (MMRCPSP) e X-10 (MMRCMPSP). Em cada caso, são apresentados os gráficos referentes à primeira e à quinta rodada de execução, sendo que, em cada rodada, todos os métodos partem da mesma solução inicial.

Os gráficos foram gerados a partir dos *logs* dos resolvedores, que registram informações detalhadas sobre cada etapa da busca, incluindo as melhorias obtidas ao longo da execução.

Para a instância SMRCPSP (j12021_4.sm) (Figura 17), é possível observar a influência da solução inicial ao comparar os comportamentos do SVNS e do Smart VNS entre as rodadas. Na primeira, o SVNS converge rapidamente, enquanto o Smart VNS apresenta melhorias mais espaçadas e estabiliza precocemente. Na quinta rodada, esse comportamento se inverte, com o Smart VNS alcançando rapidamente o melhor valor e o SVNS apresentando convergência mais lenta. De modo geral, não são observadas melhorias relevantes no último minuto de execução. Destaca-se ainda que, nessa instância, o LAHC não ultrapassa $\delta = 0{,}166$, enquanto a maioria dos demais métodos atinge valores inferiores.

Figura 17 – Curvas de convergência na resolução da instância j12021_4.sm nas rodadas 1 e 5.

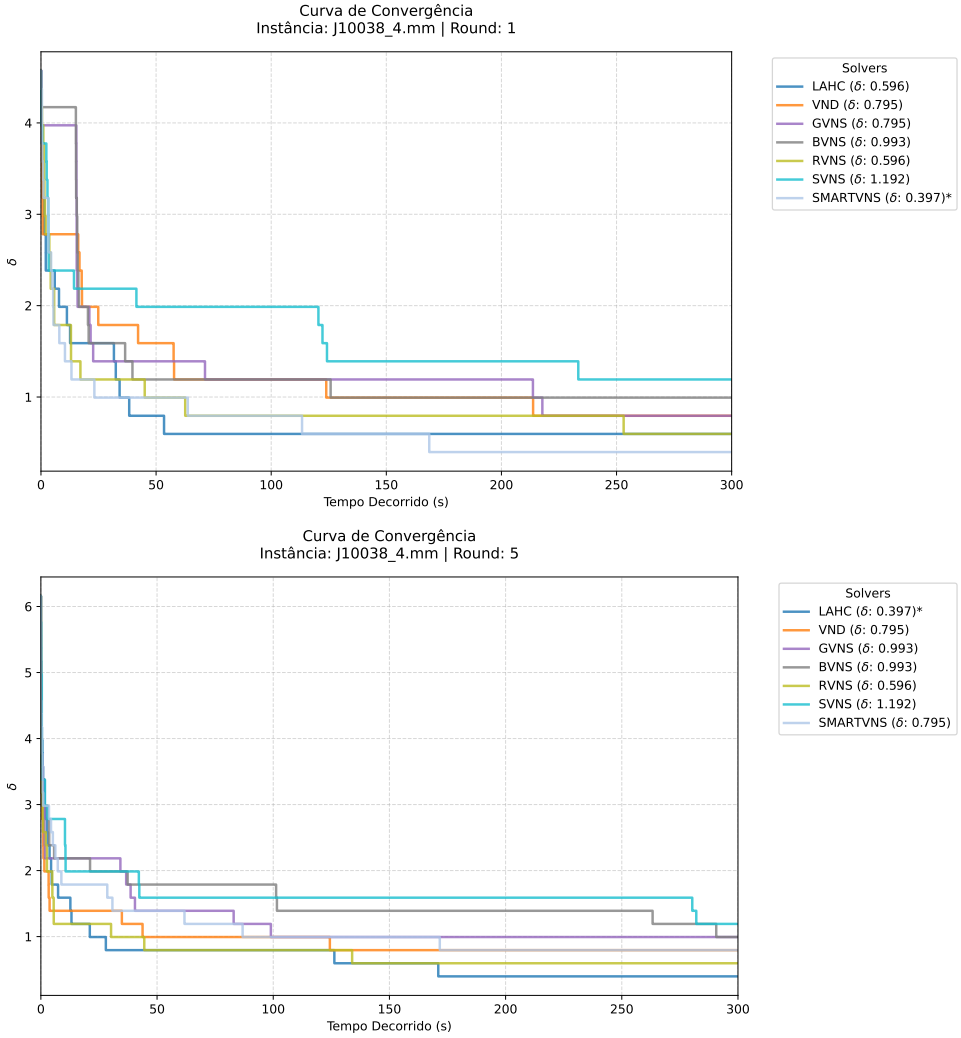


Fonte: Elaborado pelo autor.

Na instância MMRCPSP (J10038_4.mm) (Figura 19), a taxa de convergência inicial permanece mais elevada nos primeiros instantes, porém de forma menos acentuada do que na variante anterior. O SVNS apresenta maior dificuldade em escapar de platôs, permanecendo estagnado por períodos mais longos e finalizando com os piores resultados em ambas as rodadas. Em contraste, LAHC e RVNS apresentam comportamento mais consistente, com destaque para o LAHC na quinta rodada, que atinge o melhor resultado final. Os demais métodos mantêm desempenho intermediário, ficando entre os extremos observados para LAHC e SVNS.

Figura 18 – Curvas de convergência na resolução da instância J10038_4.mm nas rodadas 1 e 5.

Figura 19 – Curva convergencia MMRCPSP.



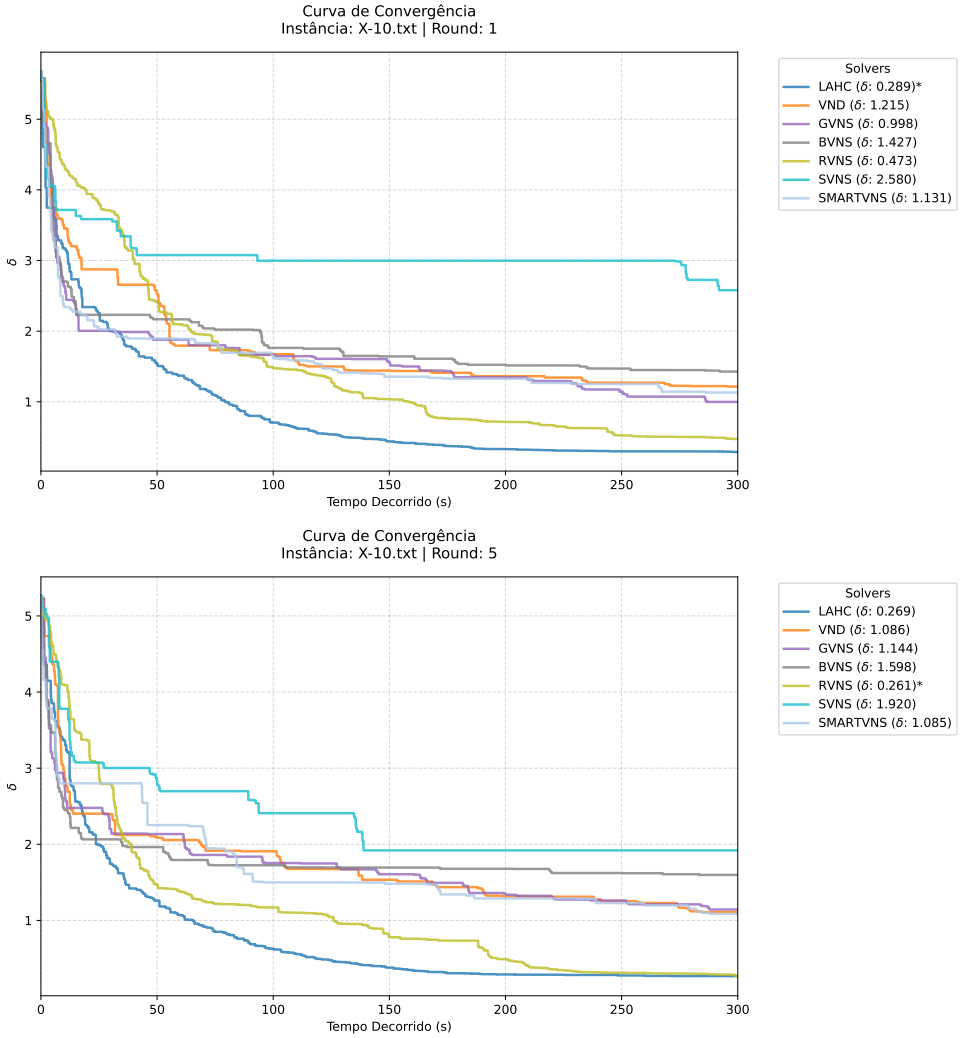
Fonte: Elaborado pelo autor.

Por fim, na instância MMRCMPSP (X-10) (Figura 20), as melhorias ocorrem de forma mais espaçada ao longo do tempo. Embora ainda se observe maior intensidade de melhorias nos instantes iniciais, os platôs tendem a ser mais curtos e sucessivos, refletindo a maior dimensão do espaço de soluções viáveis. O SVNS novamente apresenta longos períodos sem melhoria nas duas rodadas analisadas.

Em consonância com a análise do *gap* médio para essa variante, o LAHC demonstra convergência mais constante, alcançando melhores valores em estágios iniciais e mantendo um bom desempenho nas rodadas. O RVNS apresenta desempenho próximo ao LAHC, porém com quedas mais acentuadas associadas a escapes de platôs em estágios mais avançados. Os demais métodos exibem comportamento intermediário, com melhorias ocorrendo até instantes próximos ao limite de tempo, indicando convergência mais lenta

em instâncias de grande porte, possivelmente em razão do maior custo computacional associado às etapas de perturbação, busca local e troca sistemática de vizinhança.

Figura 20 – Curvas de convergência na resolução da instância X-10 nas rodadas 1 e 5.



Fonte: Elaborado pelo autor.

De forma geral, as curvas de convergência reforçam os resultados observados na análise do *gap* médio. Métodos como LAHC e RVNS apresentam comportamento mais estável e eficiente em instâncias de maior complexidade, enquanto VND e GVNS mantêm padrão equilibrado. Já SVNS, BVNS e Smart VNS tendem a apresentar maior variabilidade e maior permanência em platôs, especialmente nas variantes mais complexas, indicando menor robustez no ritmo de convergência para estes problemas.

5.1.4 Eficiência dos operadores de vizinhança

Sob a perspectiva dos operadores de vizinhança, foi realizada a análise do impacto de cada operador em cada método de resolução. A métrica adotada consiste na proporção,

em uma escala $[0, 1]$, em que cada operador foi responsável por melhorar uma solução ao longo da busca.

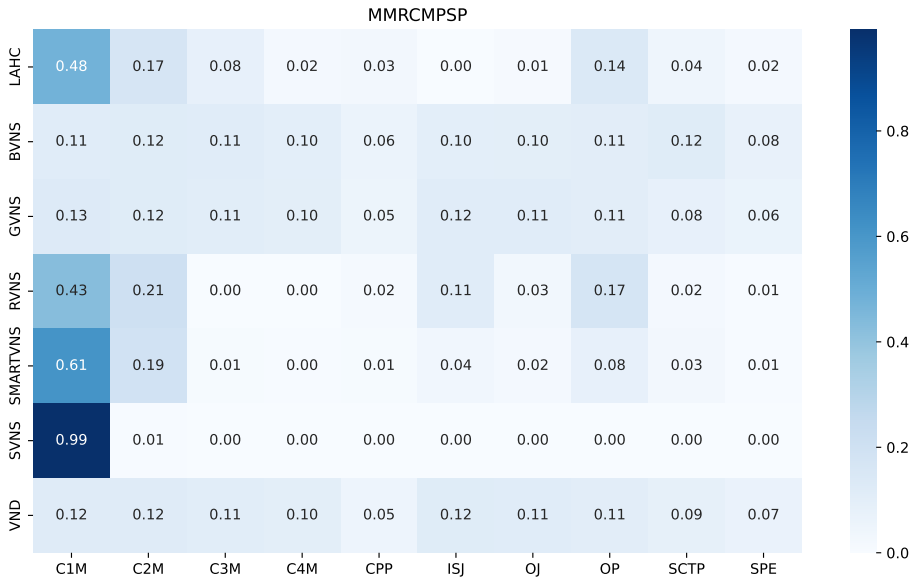
Considerando que alguns operadores utilizados neste trabalho (apresentados na subseção 4.1) são exclusivos de determinadas variantes do problema, as análises a seguir são organizadas por variante, levando em conta os operadores suportados por cada uma.

A Figura 21 apresenta a eficiência dos operadores na resolução das instâncias MMRCMPSP. É possível observar a influência da estratégia de troca de vizinhança adotada e a participação efetiva dos operadores ao longo da busca. Nos métodos que utilizaram a troca de vizinhança cíclica (VND, BVNS e GVNS), conforme descrito na Tabela 3), a eficiência dos operadores se distribui de maneira mais uniforme em comparação aos métodos que adotam troca sequencial.

Contudo, é possível observar uma exceção no caso do SVNS, que, apesar de utilizar a estratégia cíclica, apresenta melhorias quase exclusivamente por meio do operador C1M. Esse comportamento sugere que a aceitação de soluções piores pode induzir a busca a um padrão recorrente de exploração, no qual as melhorias passam a se concentrar predominantemente em um único operador.

Além disso, a influência da troca cíclica é perceptível nos operadores que atuam a nível de projetos (CPP, OP, SCTP e SPE). Ao contrário dos demais métodos, em que o operador C1M, responsável pela troca de modos de execução, predomina, esses operadores apresentam participação relevante nos métodos com troca cíclica.

Figura 21 – Eficiência dos operadores de vizinhança em cada método na resolução de instâncias MMRCMPSP

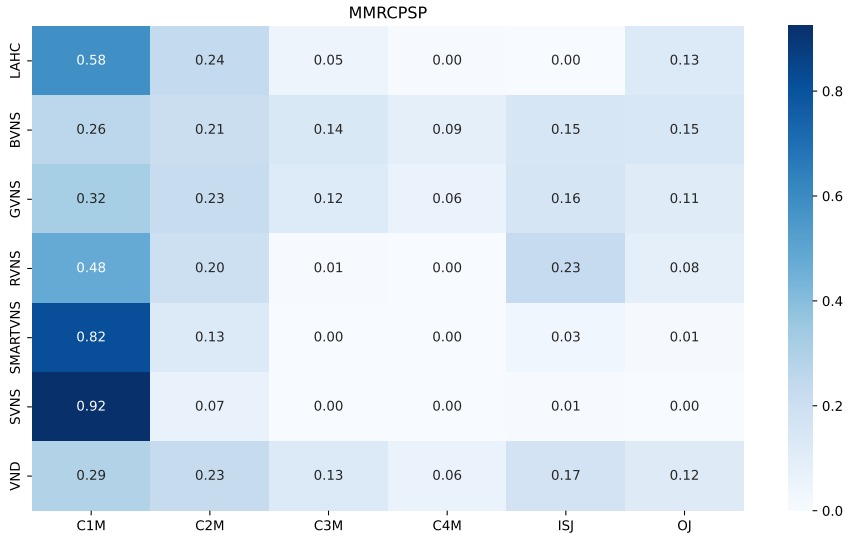


Fonte: Elaborado pelo autor.

A Figura 22 mostra a eficiência dos operadores para as instâncias MMRCPSP. Nesta variante, a diferença entre os métodos com troca cíclica e os demais aparece novamente. Entretanto, o operador C1M domina de forma mais consistente todos os métodos, refletindo a predominância de movimentos de troca de modos em instâncias com múltiplos modos.

Também é possível observar a baixa frequência de movimentos de maior intensidade, como C3M e C4M, que alteram simultaneamente três ou quatro atividades. Esse comportamento evidencia a tendência das instâncias RCPSP a apresentar melhorias a partir de movimentos mais sutis.

Figura 22 – Eficiência dos operadores de vizinhança em cada método na resolução de instâncias MMRCPSP

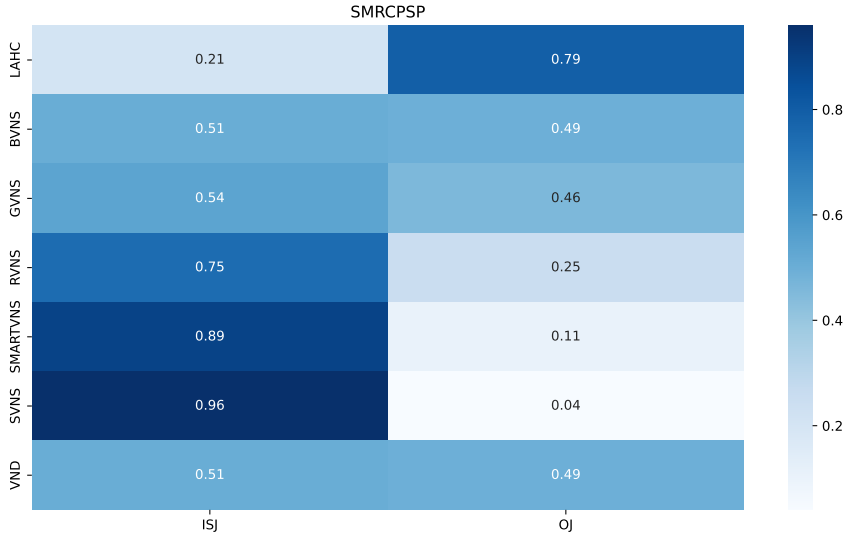


Fonte: Elaborado pelo autor.

Por fim, a Figura 23 apresenta a eficiência dos operadores nas instâncias SMRCPSP. Para esta variante, de projeto único e modo único, estão disponíveis apenas operadores que atuam sobre atividades (OJ e ISJ). É possível notar que o LAHC e RVNS apresentam aproveitamento distinto dos operadores, dado que o LAHC seleciona o operador de forma aleatória enquanto o RVNS utilizou a troca sequencial.

Nos demais métodos com troca cíclica, a distribuição da eficiência dos operadores permanece mais distribuída. Novamente, o SVNS se destaca por concentrar a maior parte das melhorias em um único operador.

Figura 23 – Eficiência dos operadores de vizinhança em cada método na resolução de instâncias SMRCPSP



Fonte: Elaborado pelo autor.

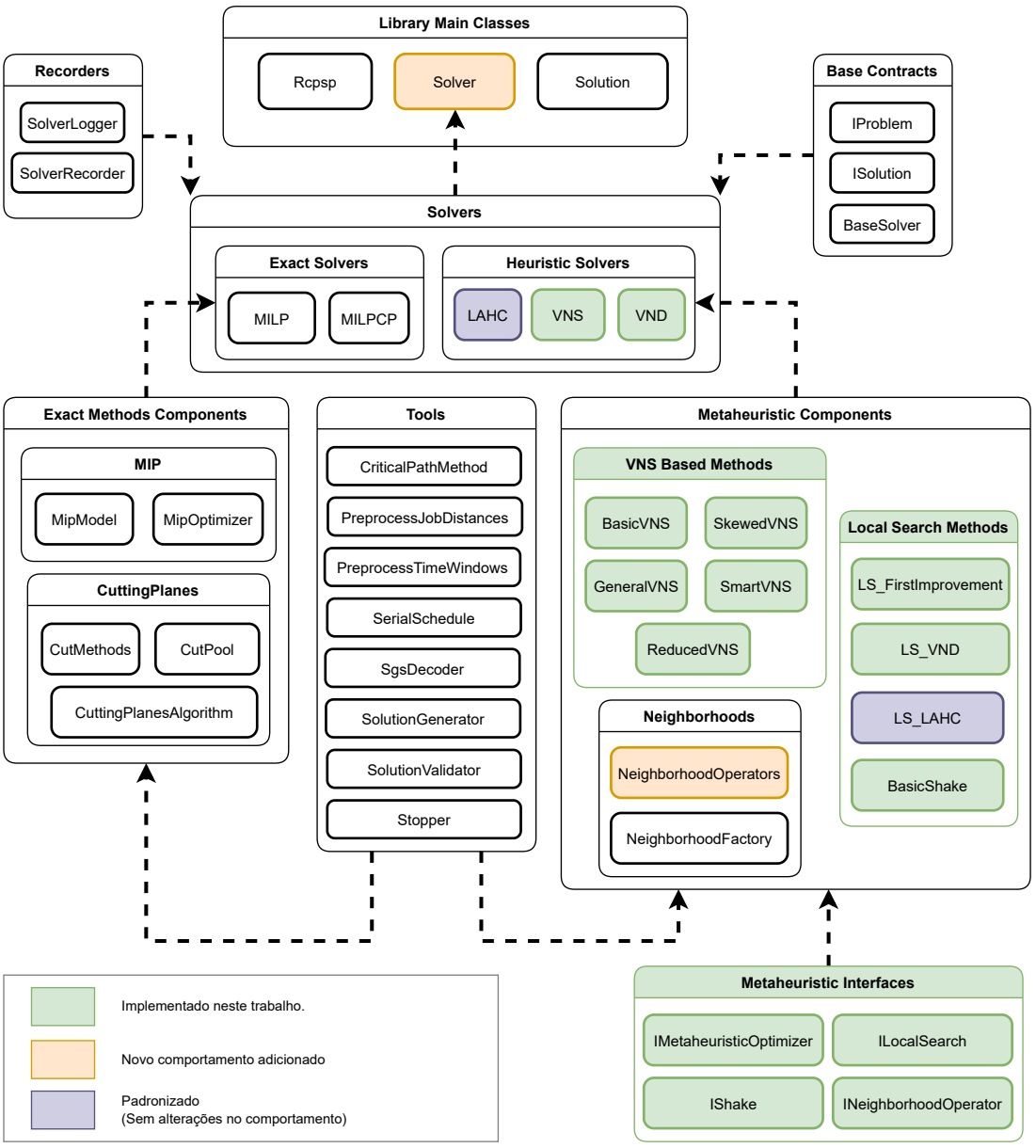
De forma geral foi possível observar que a estratégia de troca de vizinhança influencia diretamente a distribuição da eficiência dos operadores. Métodos com troca cíclica tendem a utilizar os operadores de maneira mais equilibrada, enquanto métodos com troca sequencial ou SVNS concentram melhorias em operadores específicos. Também é possível observar a predominância de melhorias geradas por meio de operadores com movimentos de menor amplitude, como o caso do C1M.

5.2 Integração dos resolvedores à biblioteca de resolvedores

A presente seção descreve o processo de integração dos métodos desenvolvidos neste trabalho à biblioteca de resolvedores da plataforma. O processo de foi conduzido de modo a preservar o alinhamento arquitetural da biblioteca, mantendo a separação de responsabilidades entre interfaces, componentes internos e resolvedores concretos.

A Figura 24 apresenta o diagrama de componentes após a inclusão dos novos métodos. Dois novos resolvedores base foram adicionados: o `VND` que aplica a resolução a partir do método de busca de mesmo nome, e o `VNS` que agrupa os resolvedores baseados em VNS. Na camada de componentes meta-heurísticos foram adicionadas as implementações concretas dos algoritmos, assim como os novos métodos de busca local (apresentados no Capítulo 4). Também foram definidas novas interfaces a fim de padronizar o comportamento de componentes meta-heurísticos.

Figura 24 – Diagrama de componentes da biblioteca de resolvedores após a integração dos métodos desenvolvidos neste trabalho.



Fonte: Elaborado pelo autor.

O processo de integração também demandou a atualização de alguns componentes, como os associados ao LAHC, que agora implementam as interfaces específicas dos componentes meta-heurísticos. Contudo, o comportamento destes componentes não foi alterado.

Novos comportamentos foram adicionados às classes `Solver` (classe externa da biblioteca) e `NeighborhoodOperators`. Na classe `Solver` foi adicionado o suporte para os parâmetros de entrada e inicialização concreta dos novos resolvedores internos. Na classe

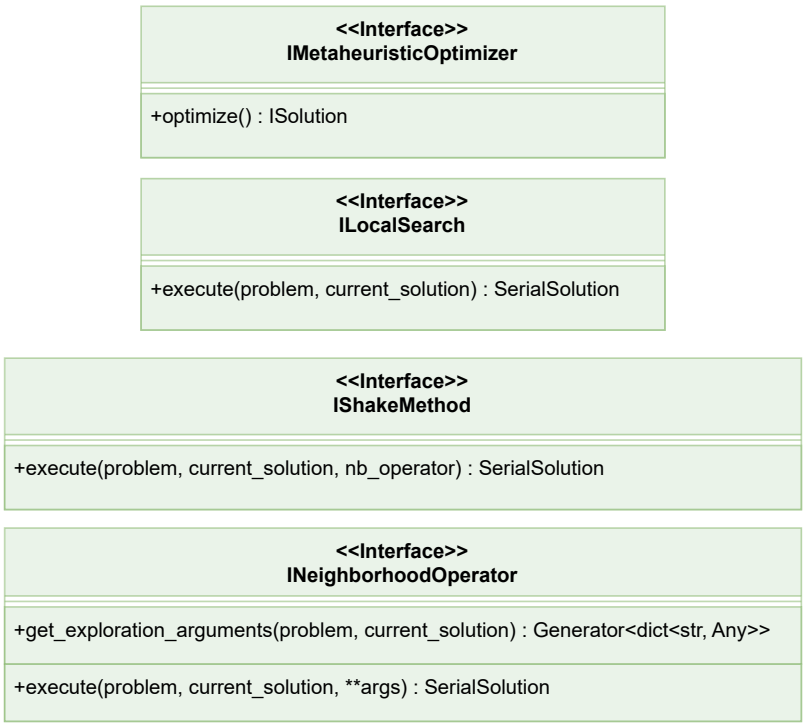
`NeighborhoodOperators` foi adicionado o recurso de geração de vizinhanças sob demanda, conforme mencionado em 4.1.1.

As subseções a seguir descrevem de forma detalhada cada componente adicionado.

5.2.1 Novas interfaces para componentes meta-heurísticos

As novas interfaces para os componentes meta-heurísticos definem os contratos formais para métodos de otimização, busca local, operadores de vizinhança e procedimentos de perturbação, conforme mostra a Figura 25. A adição destas interfaces simplifica a interoperabilidade entre componentes, permitindo que diferentes abordagens sejam combinadas ou ampliadas, desde que implementem os contratos estabelecidos.

Figura 25 – Interfaces definidas para as meta-heurísticas.



Fonte: Elaborado pelo autor.

A interface `IMetaheuristicOptimizer` define o contrato para métodos de otimização meta-heurísticos. Seu método principal, `optimize()`, retorna um objeto do tipo `ISolution`. A ausência de parâmetros formais nesse método decorre do fato de que os resolvedores, localizados em nível superior de abstração, são responsáveis por encapsular o problema, condições de parada, inicializações e pré-processamentos necessários.

A interface `ILocalSearch` padroniza o comportamento dos algoritmos de busca local. O método `execute(...)` recebe como parâmetros obrigatórios o problema e a solução atual, e permite argumentos adicionais de acordo com o escopo da implementação concreta. O retorno é uma solução no formato sequencial.

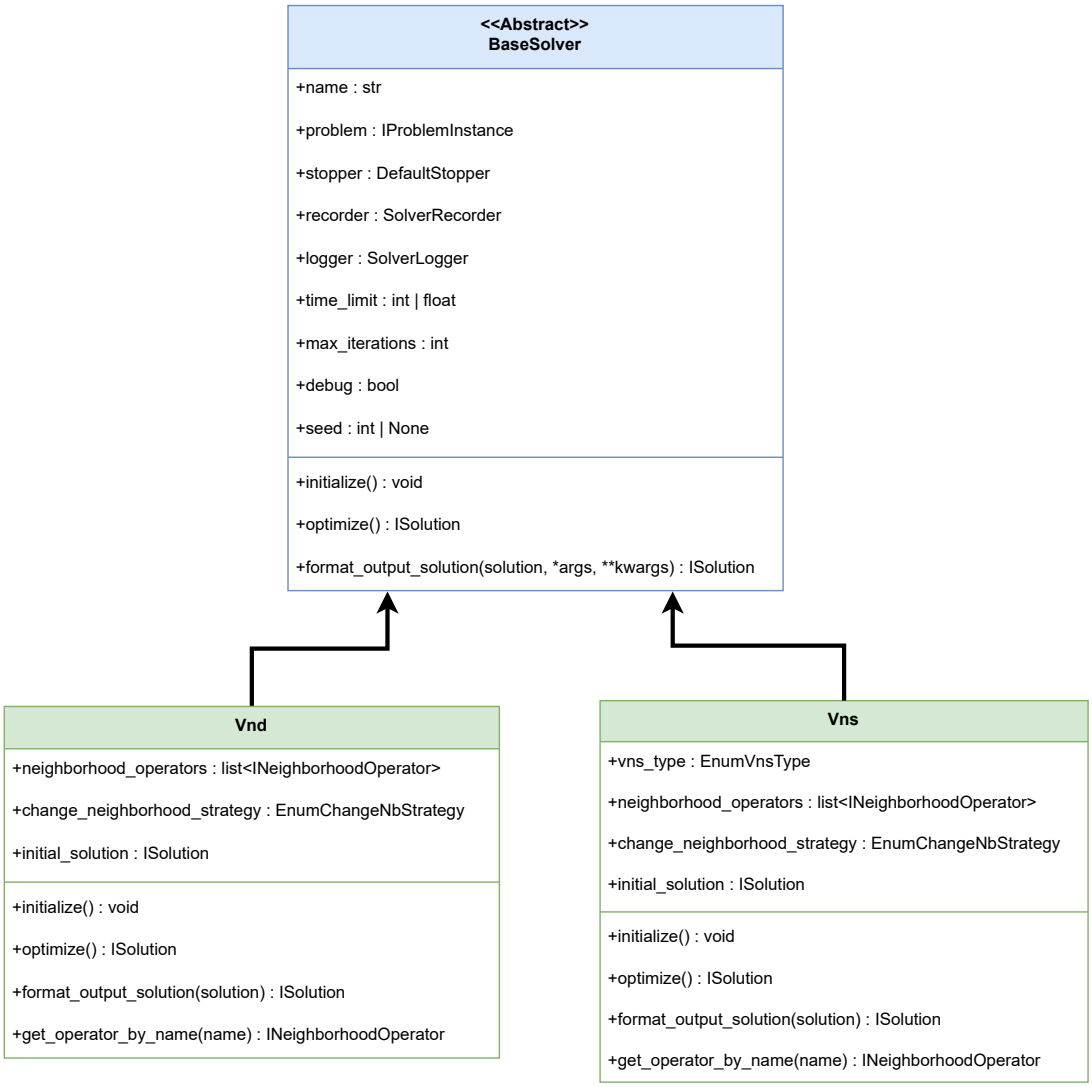
A interface `INeighborhoodOperator` estabelece o contrato para operadores de vizinhança. O método `get_exploration_arguments(...)` é responsável por gerar, sob demanda, os argumentos necessários à exploração da vizinhança. O método `execute(...)` aplica o movimento à solução corrente. Os argumentos extras são os parâmetros que informam como o operador aplicará o movimento (sobre quais elementos da solução e o nível de intensidade). Em casos de uso onde é necessária a exploração sistemática (como no VNS e VND), estes parâmetros são os obtidos pelo método `get_exploration_arguments(...)`.

5.2.2 Novas classes de resolvedores

A convenção na arquitetura da biblioteca é que cada método de resolução tenha uma classe base implementada na camada `Solvers`. As classes desta camada devem ser declaradas a partir da classe abstrata `BaseSolver`, que define as propriedades comuns entre os resolvedores, como o problema de entrada, os critérios de parada e a assinatura dos métodos responsáveis pela inicialização, execução do processo de otimização e formatação da solução final.

O resolvedor `VNS` atua como ponto unificado de entrada para as variantes do VNS, onde escolha da variante deve ser especificada via parâmetro de inicialização. O resolvedor `VND` utiliza exclusivamente o método VND como estratégia de resolução. A Figura 26 mostra a estrutura dessas classes por meio da especialização de `BaseSolver`.

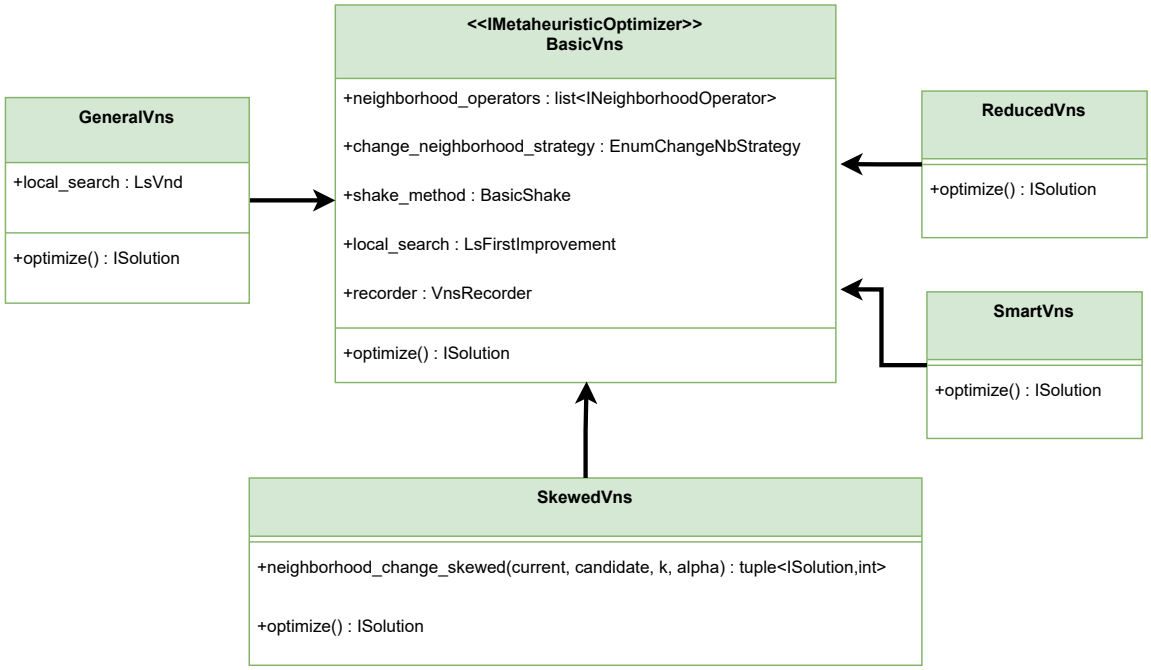
Figura 26 – Diagrama de classe dos Resolvedores VNS e VND. Ambos implementam a classe abstrata BaseSolver, que é padrão dos resolvedores da biblioteca.



Fonte: Elaborado pelo autor.

A implementação dos algoritmos baseados em VNS foi adicionada à camada `Metaheuristic Components`. Como essas variantes compartilham a mesma estrutura geral, a classe `BasicVNS` serve como base para as demais (Figura 27). Cada variante apenas sobrescreve o método `optimize()` com sua lógica específica, o que preserva a estrutura comum e garante a uniformidade no comportamento esperado para esses métodos.

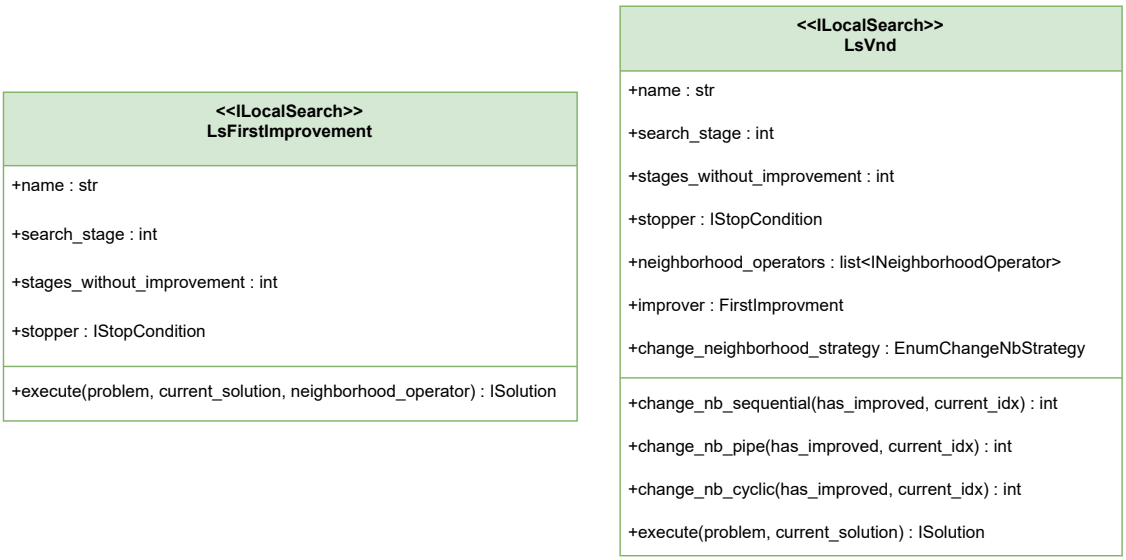
Figura 27 – Diagrama das classes que implementam os algoritmos baseados em VNS.



Fonte: Elaborado pelo autor.

Os novos métodos de busca local foram foram também integrados à camada `Metaheuristic Components`. As implementações da busca de Primeira Melhoria e VND seguem o contrato definido por `ILocalSearch`, conforme ilustrado na Figura 28.

Figura 28 – Diagrama de classe dos métodos de busca local VND e *FirstImprovement*.



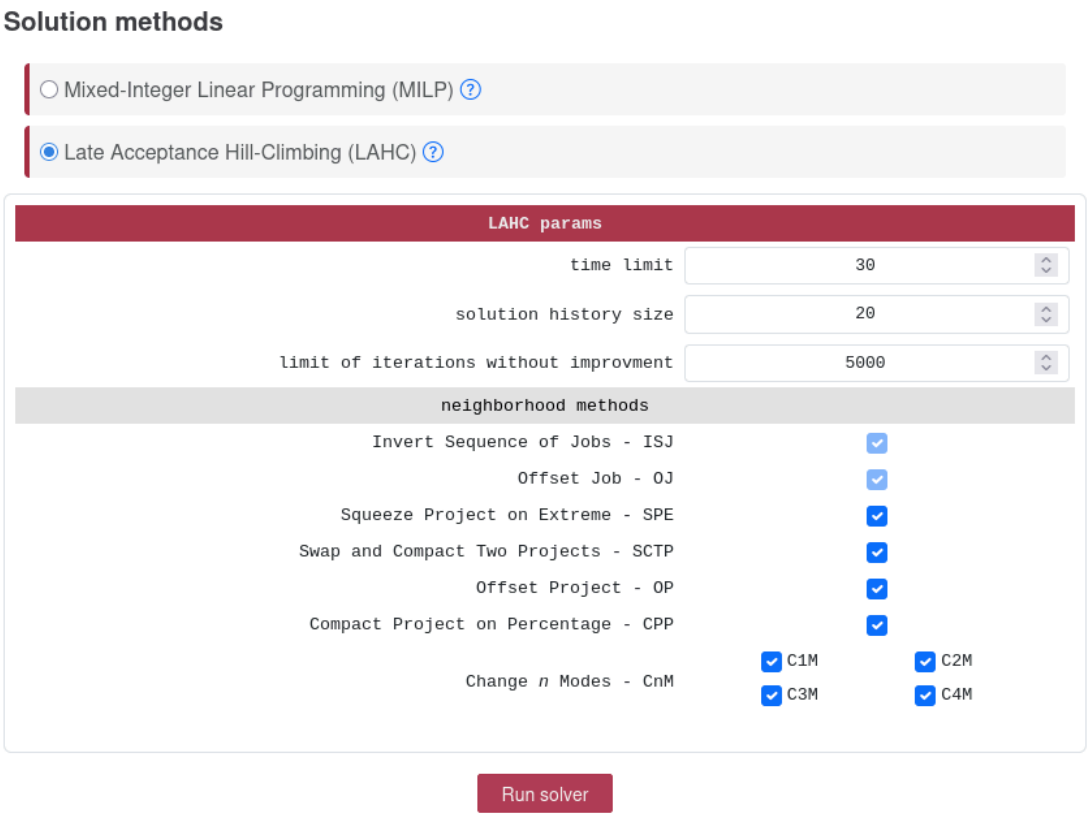
Fonte: Elaborado pelo autor.

5.3 Integração dos resolvedores à plataforma (Interface *Web*)

Nesta seção são apresentados os resultados obtidos com a integração das novas abordagens meta-heurísticas à ItPlatPSP, objetivo principal deste trabalho. A seguir, são descritas as alterações realizadas na interface gráfica da plataforma, bem como os novos recursos disponibilizados ao usuário em decorrência dessa integração.

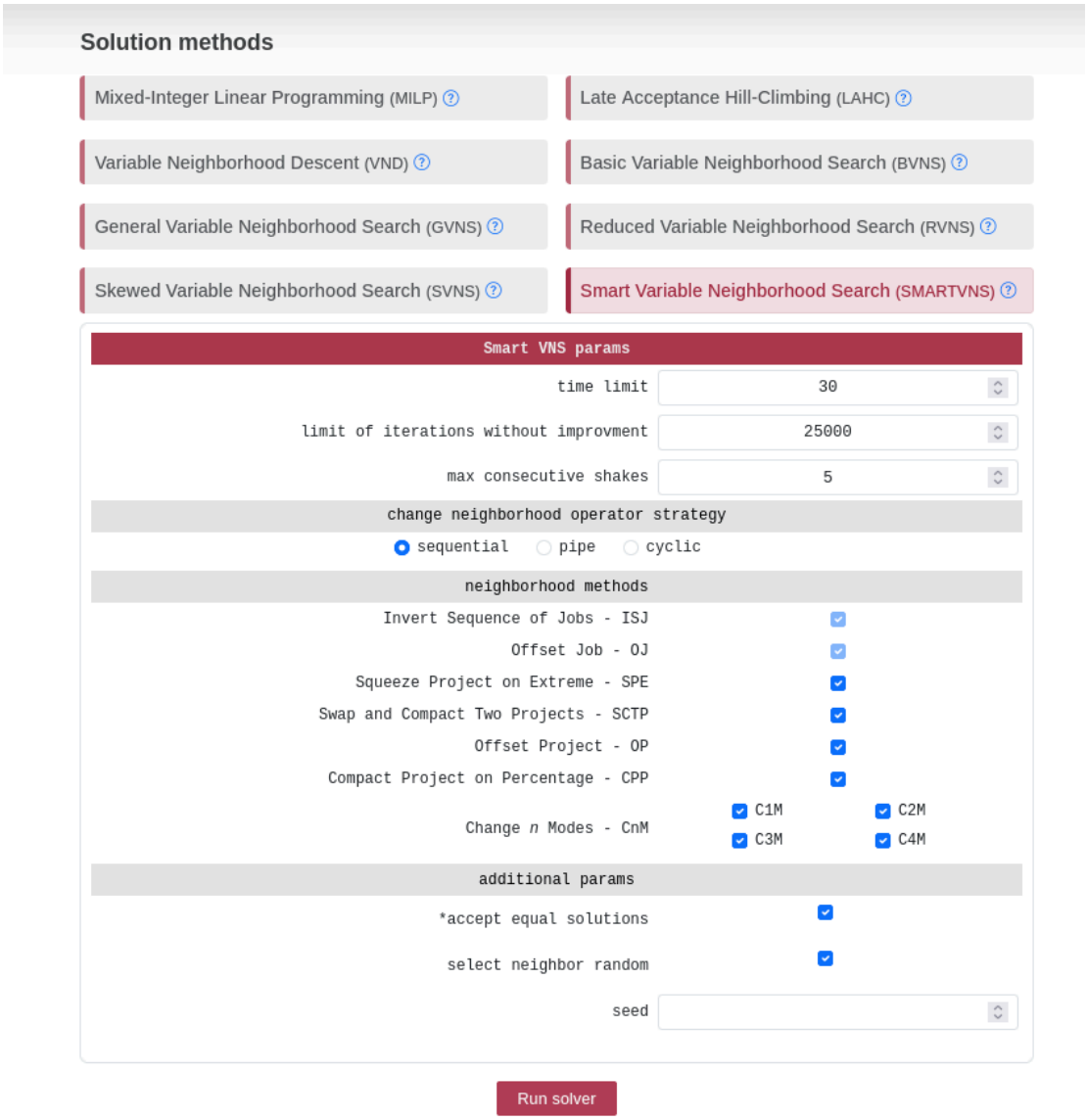
A Figura 29 apresenta o componente de seleção de resolvedores antes do desenvolvimento realizado neste trabalho, enquanto a Figura 30 mostra o mesmo componente após a atualização. Os novos métodos foram adicionados ao conjunto de resolvedores disponíveis no painel de resolução de instâncias (rota `/solver`). Ajustes pontuais de estilo foram realizados com o objetivo de preservar a organização visual e a usabilidade do sistema.

Figura 29 – Painel de resolvedores da ItPlatPSP antes dos métodos desenvolvidos neste trabalho.



Fonte: Elaborado pelo autor.

Figura 30 – Painel de resolvedores da ItPlatPSP depois de adicionar os métodos desenvolvidos neste trabalho.



Fonte: Elaborado pelo autor.

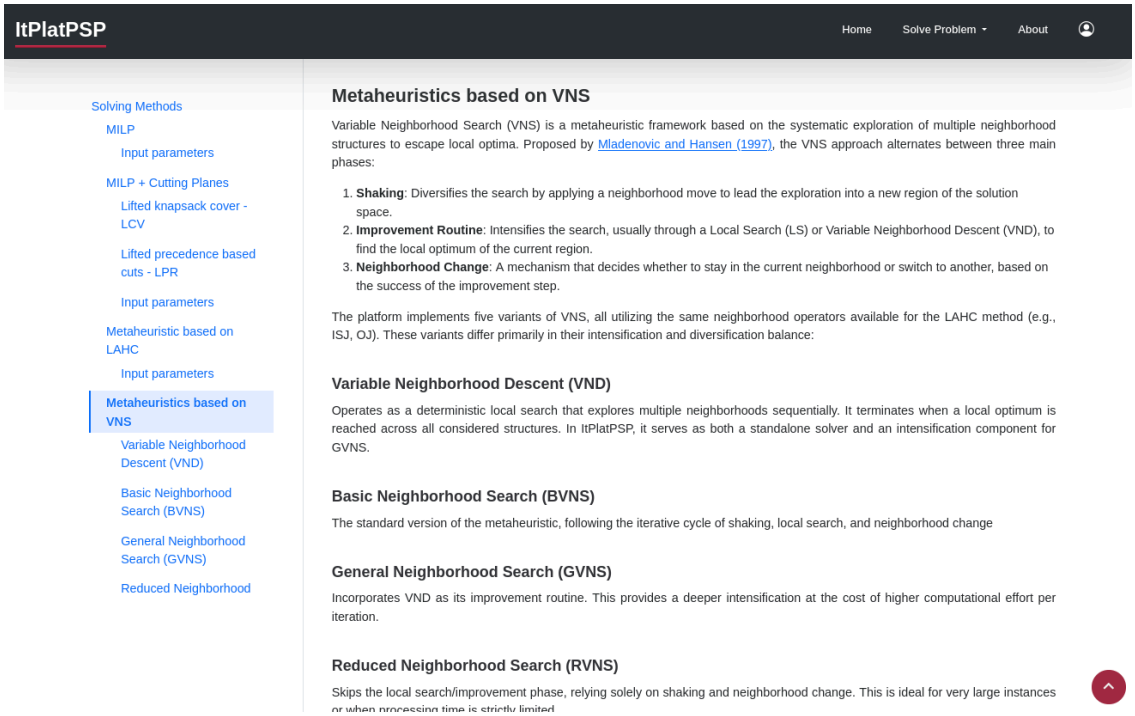
Os parâmetros de entrada dos novos métodos disponibilizados na interface incluem tempo limite e número máximo de iterações sem melhoria (critérios de parada), os operadores de vizinhança a serem utilizados na busca e a estratégia de troca de vizinhança adotada (conforme apresentado em 4.4).

Também estão disponíveis parâmetros relacionados ao comportamento interno dos métodos, tais como: aceitar soluções de mesmo valor objetivo, desde que o cronograma correspondente seja distinto; habilitar a seleção aleatória de vizinhos durante a exploração de um operador, em substituição à exploração sistemática; e, por fim, o parâmetro *seed*, que viabiliza a reprodutibilidade de procedimentos internos aleatórios.

Para os métodos SVNS e Smart VNS, o menu de parâmetros inclui ainda configurações específicas. No caso do SVNS, está disponível o fator de desvio utilizado na aceitação de soluções piores. O Smart VNS inclui o parâmetro que define o número máximo de perturbações permitidas em uma mesma iteração.

Por fim, foi realizada a atualização da página destinada à documentação geral da plataforma (rota `/about`), na qual foram adicionadas referências aos novos métodos integrados, bem como descrições dos parâmetros de entrada disponibilizados ao usuário. A Figura 31 apresenta um trecho da seção incorporada.

Figura 31 – Página about atualizada.



Fonte: Elaborado pelo autor.

6 Conclusão

Este trabalho apresentou a ampliação dos métodos de resolução disponibilizados pela ItPlatPSP, por meio da integração sistemática de novas abordagens meta-heurísticas para os RCPSP. A principal contribuição deste trabalho foi a consolidação de um conjunto mais robusto e parametrizável de ferramentas de apoio à decisão, aliada ao fortalecimento da modularidade da plataforma, por meio da padronização de componentes meta-heurísticos estruturados com base em contratos e interfaces bem definidos.

A metodologia contribuiu para a formalização de contratos entre módulos na organização dos métodos sob uma interface comum. Tal abordagem favoreceu a extensibilidade e o reuso de componentes, além de viabilizar a integração entre o ambiente de prototipação, a biblioteca de resolvedores e a plataforma web.

Os experimentos computacionais evidenciaram que a ampliação da biblioteca diversificou o comportamento exploratório da plataforma, proporcionando diferentes estratégias de intensificação e diversificação no processo de busca. Os métodos implementados demonstraram viabilidade computacional no ambiente proposto e desempenho consistente, com destaque para variantes baseadas em intensificação estruturada, como VND e GVNS, que apresentaram bons resultados em diferentes conjuntos de instâncias e cenários experimentais.

Quanto às limitações, a natureza da plataforma web exige um equilíbrio entre a eficácia dos métodos e os recursos computacionais disponíveis. Ao contrário de ambientes dedicados exclusivamente à experimentação intensiva, a plataforma opera como um serviço acessível a múltiplos usuários, o que demanda controle sobre tempo de execução, consumo de recursos computacionais e disponibilidade do sistema.

Como perspectivas futuras, sugere-se a incorporação de mecanismos de parametrização dinâmica ao longo da busca nas meta-heurísticas, bem como a integração de novos operadores de vizinhança. Tais avanços podem ampliar a capacidade exploratória dos métodos implementados e fortalecer a plataforma como ambiente integrado de apoio à pesquisa e ao ensino em escalonamento de projetos.

Referências

ARAUJO, J. A. et al. Strong bounds for resource constrained project scheduling: Preprocessing and cutting planes. *Computers & Operations Research*, v. 113, p. 104782, jan. 2020. ISSN 03050548. Citado 3 vezes nas páginas 17, 20 e 30.

ARAUJO, J. A. S. Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling problem. *PPGCC - Doutorado (Teses)*, 2019. Citado 6 vezes nas páginas 17, 25, 31, 34, 35 e 36.

ASTA, S. et al. *Combining Monte-Carlo and Hyper-heuristic Methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem*. [S.l.]: arXiv, 2016. Citado 4 vezes nas páginas 20, 22, 25 e 31.

BLAZEWICZ, J.; LENSTRA, J.; KAN, A. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, v. 5, n. 1, p. 11–24, jan. 1983. ISSN 0166218X. Citado na página 17.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308, set. 2003. ISSN 0360-0300, 1557-7341. Citado na página 23.

BRUCKER, P. et al. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, v. 112, n. 1, p. 3–41, jan. 1999. ISSN 03772217. Citado na página 21.

BURKE, E. K.; BYKOV, Y. The late acceptance Hill-Climbing heuristic. *European Journal of Operational Research*, v. 258, n. 1, p. 70–78, abr. 2017. ISSN 03772217. Citado 2 vezes nas páginas 17 e 24.

DEBLAERE, F.; DEMEULEMEESTER, E.; HERROELEN, W. RESCON: Educational project scheduling software. *Computer Applications in Engineering Education*, v. 19, n. 2, p. 327–336, jun. 2011. ISSN 1061-3773, 1099-0542. Citado na página 26.

DEMEULEMEESTER, E. L.; HERROELEN, W. S. *Project Scheduling: A Research Handbook*. 1st ed. 2002. ed. New York, NY: Imprint: Springer, 2002. (International Series in Operations Research & Management Science, 49). ISBN 978-0-306-48142-0. Citado 6 vezes nas páginas 17, 18, 21, 22, 23 e 31.

GMYREK, K. et al. iMOPSE: A Comprehensive Open Source Library for Single- and Multi-objective Metaheuristic Optimization. In: AFFENZELLER, M. et al. (Ed.). *Parallel Problem Solving from Nature – PPSN XVIII*. Cham: Springer Nature Switzerland, 2024. v. 15149, p. 170–184. ISBN 978-3-031-70067-5 978-3-031-70068-2. Citado na página 26.

HANSEN, P. et al. Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization*, v. 5, n. 3, p. 423–454, set. 2017. ISSN 21924406. Citado 3 vezes nas páginas 41, 42 e 43.

JOHNSON, D. S.; PAPADIMITRIOU, C. H.; YANNAKAKIS, M. How easy is local search? *Journal of Computer and System Sciences*, v. 37, n. 1, p. 79–100, ago. 1988. ISSN 00220000. Citado na página 24.

KELLEY, J. E.; WALKER, M. R. Critical-path planning and scheduling. In: *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference on - IRE-AIEE-ACM '59 (Eastern)*. Boston, Massachusetts: ACM Press, 1959. p. 160–173. Citado na página 21.

KOLISCH, R.; SPRECHER, A. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, v. 96, n. 1, p. 205–216, jan. 1997. ISSN 03772217. Citado 2 vezes nas páginas 18 e 25.

KOLISCH, R.; SPRECHER, A.; DREXL, A. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science*, v. 41, n. 10, p. 1693–1703, out. 1995. ISSN 0025-1909, 1526-5501. Citado 2 vezes nas páginas 17 e 21.

LAN, L.; BERKHOUT, J. *PyJobShop: Solving Scheduling Problems with Constraint Programming in Python*. [S.l.]: arXiv, 2025. Citado na página 26.

MARTÍ, R.; SEVAUX, M.; SÖRENSEN, K. Fifty years of metaheuristics. *European Journal of Operational Research*, v. 321, n. 2, p. 345–362, mar. 2025. ISSN 03772217. Citado na página 24.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, nov. 1997. ISSN 03050548. Citado 4 vezes nas páginas 17, 24, 33 e 40.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. 6. pr. ed. Englewood Cliffs, N.J: Prentice-Hall, 1982. ISBN 978-0-13-152462-0. Citado na página 23.

PEARL, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Mass: Addison-Wesley Pub. Co, 1984. (The Addison-Wesley Series in Artificial Intelligence). ISBN 978-0-201-05594-8. Citado na página 23.

PETEGHEM, V. V.; VANHOUCKE, M. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, v. 235, n. 1, p. 62–72, maio 2014. ISSN 03772217. Citado na página 25.

RARDIN, R. L.; UZSOY, R. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, v. 7, n. 3, p. 261–304, maio 2001. ISSN 1381-1231, 1572-9397. Citado na página 23.

SOUZA, M. et al. A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, v. 207, n. 2, p. 1041–1051, dez. 2010. ISSN 03772217. Citado na página 44.

TOULOUSE, M.; THULASIRAMAN, K.; GLOVER, F. Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In: AMESTOY, P. et al. (Ed.). *Euro-Par'99 Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. v. 1685, p. 533–542. ISBN 978-3-540-66443-7 978-3-540-48311-3. Citado na página 24.

VOUDOURIS, C.; TSANG, E. P. K. Guided Local Search. In: GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook of Metaheuristics*. Boston: Kluwer Academic Publishers, 2003. v. 57, p. 185–218. ISBN 978-1-4020-7263-5. Citado na página 24.

WAUTERS, T. et al. The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem: The MISTA 2013 challenge. *Journal of Scheduling*, v. 19, n. 3, p. 271–283, jun. 2016. ISSN 1094-6136, 1099-1425. Citado 5 vezes nas páginas 18, 21, 22, 24 e 25.

WĘGLARZ, J.; HILLIER, F. S. (Ed.). *Project Scheduling*. Boston, MA: Springer US, 1999. v. 14. (International Series in Operations Research & Management Science, v. 14). ISBN 978-1-4613-7529-6 978-1-4615-5533-9. Citado na página 22.